DataSimilarity: Quantifying Similarity of Datasets and Multivariate Two- and k-Sample Testing

Marieke Stolte
TU Dortmund University and LMU Munich

Luca Sauer
TU Dortmund University

Jörg Rahnenführer
TU Dortmund University

Andrea Bommert
TU Dortmund University

Abstract

Quantifying the similarity of two or more datasets is a common task in many applications in statistics and machine learning, such as two- or k-sample testing and meta- or transfer learning. The **DataSimilarity** package contains a variety of methods for quantifying the similarity of datasets. The package includes 40 methods of which 17 are implemented for the first time in R. The remaining are wrapper functions for methods with already existing implementations that unify and simplify the various input and output formats of different methods and bundle the methods of many existing R packages in a single package. Here, we give details on the methods and implementations and show some application examples.

Keywords: dataset similarity, two-sample testing, multi-sample testing.

1. Methods

In the following, we describe the general setup in the two- or k-sample problem that most of the implemented methods have in common. Moreover, we discuss the selection of the implemented methods and present one example method for each application domain in more detail.

1.1. The two- and k-sample problem

Most methods for quantifying the similarity of datasets are proposed in the literature as test statistics for two- or k-sample testing. For this, a dataset is seen as a sample from a set of random variables that follow some true underlying distribution. Often, the similarity or distance of these underlying distributions is estimated.

In the following, we assume that at least two different datasets $X^{(1)}$ and $X^{(2)}$ are given consisting of n_1 and n_2 samples $X_1^{(1)}, \ldots, X_{n_1}^{(1)} \sim F_1$ and $X_1^{(2)}, \ldots, X_{n_2}^{(2)} \sim F_2$, respectively. We assume $X_i^{(1)}, X_j^{(2)} : \mathcal{X} \to \mathbb{R}^p \; \forall i \in \{1, \ldots, n_1\}, j \in \{1, \ldots, n_2\}$ and call the p components of each sample features or variables. The two-sample problem is defined as the testing problem

$$H_0 : F_1 = F_2 \text{ vs. } H_1 : F_1 \neq F_2. \tag{1}$$

This testing problem is sometimes also called testing for homogeneity of the two distributions.

In some cases, one of the variables in each dataset has the role of a target variable, which we will then denote as Y. While data in many domains (e.g., clinical applications) typically has a clearly defined endpoint, the literature on quantifying dataset similarity has mostly neglected the existence of target variables. Most dataset similarity methods (at least implicitly) assume that all variables can be treated the same way. However, there are some methods that can handle the target variable differently and take the relationship of the remaining variables with the target variable into account. Therefore, whenever necessary in the following, we distinguish between methods that can handle a target variable appropriately and methods that treat all variables in the same way.

Analogously to the two-sample problem, the k-sample or multi-sample problem is defined for $k \geqslant 2, k \in \mathbb{N}$, datasets $X^{(1)}, \ldots, X^{(k)}$ with sample sizes $n_i, i = 1, \ldots, k$, as

$$H_0 : F_1 = F_2 = \cdots = F_k \text{ vs. } H_1 : \exists i \neq j \in \{1, \ldots, k\} : F_i \neq F_j,$$

where F_i denotes the distribution from which each observation in the ith dataset is drawn.

Each of the considered methods can be seen as a measure of similarity or distance between the F_i, $i = 1, \ldots, k$. Not all of these methods include a hypothesis test.

We use the hat symbol to denote estimators. We denote the pooled sample as $\{Z_1, \ldots, Z_N\} = \{X_1^{(1)}, \ldots, X_{n_1}^{(1)}, \ldots, X_1^{(k)}, \ldots, X_{n_k}^{(k)}\}$, where $N = \sum_{i=1}^{k} n_i$ is the total sample size. Additionally, we assume that all Z_i are distributed independently.

1.2. Selection of methods

Previously, in a comprehensive literature review (Stolte, Kappenberg, Rahnenführer, and Bommert 2024), 118 methods were described and divided into the ten classes:

1. Comparison of cumulative distribution functions, density functions, or characteristic functions,
2. Methods based on multivariate ranks,
3. Discrepancy measures for distributions,
4. Graph-based methods,
5. Methods based on inter-point distances,
6. Kernel-based methods,
7. Methods based on binary classification,
8. Distance and similarity measures for datasets,
9. Comparison based on summary statistics, and
10. Testing approaches.

Moreover, the methods were compared with respect to 22 criteria judging their applicability, interpretability, and theoretical properties. The criteria are listed in Table 1. Here, methods are selected from the literature review using the theoretical criteria to perform a pre-selection of promising methods. The **DataSimilarity** package comprises 36 methods that fulfill at least one of the following properties:

1. The method is implemented in R.
2. The method is one of the top methods ordered by the highest number of fulfilled criteria and fulfills at least 11 criteria of the theoretical criteria, excluding the two consistency criteria.
3. The method is the best in its subclass in the theoretical comparison, and no other method from this subclass was chosen based on the first two criteria.

To avoid preferring methods that define a test over methods that do not, and therefore can by definition not fulfill the consistency criteria, consistency is not counted for determining the top methods. We chose 11 as the cutoff for the number of fulfilled criteria, as this is the median number of fulfilled criteria (excluding consistency) for the already implemented methods, and it ensures that at least more than half of the criteria are fulfilled. All methods from the theoretical comparison fulfilling the above-mentioned properties are included except for the method of Weiss (1960), for which no concrete test is defined but only the general idea. Moreover, that test lacks symmetry, i.e., the test result depends on the order in which the datasets are supplied, which is highly undesirable in practice.

There are additional methods (DMMD, DFDA by Kirchler, Khorasani, Kloft, and Lippert (2020), and ME, SCF by Chwialkowski, Ramdas, Sejdinovic, and Gretton (2015); Jitkrittum, Szabó, Chwialkowski, and Gretton (2016)) implemented in Python, but these are not compatible with current versions of Python and the used packages and can therefore no longer be run directly. As these methods also performed poorly in the theoretical comparison and other methods based on similar ideas (MMD) are implemented in this package, we did not take the effort to re-implement the methods in R from scratch. The same holds for the block MMD (Zaremba 2022) for which a MATLAB implementation exists. Since it is just a block-wise estimation of the already implemented MMD, it is not included here.

Note that only nonparametric methods that are applicable to multivariate data, can be used as a measure of dataset (dis)similarity, and do not focus on comparing specific characteristics (like location or scale) were included here. There exist many additional methods for the univariate case, for specific distribution families or more specific test problems. A very general testing framework that includes many of these as special cases is, e.g., provided by the **coin** package (Hothorn, Hornik, van de Wiel, and Zeileis 2008). Here, we kept to these restrictions since univariate data is in many applications an exception, and distributional assumptions are often hard to check in practice.

	Criterion
	Applicability
1.	Is the method able to handle a target variable present in the datasets appropriately?
2.	Is the method applicable to numeric datasets?
3.	Is the method applicable to categorical datasets?
4.	Is the method applicable to datasets with unequal sample sizes?
5.	Is the method applicable to datasets with more variables than observations?
6.	Is the method applicable to $k > 2$ datasets at the same time?
7.	Can the method be applied without the need of a training dataset?
8.	Is the method defined without making further assumptions about the dataset characteristics or the underlying distributions?
9.	Can the method be applied without choosing or tuning hyperparameters?
10.	Is the method implemented?
11.	What is the computational complexity of the method?
	Interpretability
12.	Can a one unit increase in the statistic values that the method returns be interpreted?
13.	Is there a known lower bound of the statistic values?
14.	Is there a known upper bound of the statistic values?
	Theoretical Properties
15.	Is the method invariant to rotating all datasets in the same way?
16.	Is the method invariant to location change applied to all datasets in the same way?
17.	Is the method invariant to change of scale applied to all datasets in the same way?
18.	Is the result of the method $\geqslant 0$ and zero iff the distributions of the datasets coincide (positive definite)?
19.	Is the method symmetric (regarding the order of two datasets)?
20.	Does distance given by the method fulfill the triangle inequality?
21.	Is the corresponding two- or k-sample test consistent (for $N \to \infty$)?
22.	Is the corresponding two- or k-sample test HDLSS consistent (for $p \to \infty$,)?

Table 1: Criteria for theoretical method comparison (Stolte *et al.* 2024).

2. Implementation overview

In the following, we give an overview of the implemented methods and the general structure of the implementations. Moreover, we discuss potential challenges in the application of the methods in practice.

2.1. Implemented methods

For the **DataSimilarity** package, existing implementations are used where possible. If methods already have a name in the article where they were proposed or in the secondary literature, the corresponding functions are named after that, e.g., `Wasserstein()` for the Wasserstein distance (Vaserstein 1969), `MMD()` for the maximum mean discrepancy (MMD) (Gretton, Borgwardt, Rasch, Schölkopf, and Smola 2006), or `CMDistance()` for the constrained minimum (CM) distance (Tatti 2007). Otherwise, the function names are composed of the first letters of the surnames of all authors of the article where the respective method was originally proposed, e.g., `FR()` for the Friedman-Rafsky test proposed by Friedman and Rafsky (1979), or the full surname in case of a single author, e.g., `Bahr()` for the test proposed by Bahr (1996). The input and output of the methods from different existing packages and of the newly implemented methods are unified. To achieve this, for some existing methods, it was sufficient to implement a wrapper calling the original function.

In other cases, we re-implemented the method from scratch if the R package was archived and additional issues with the original implementation occurred. This was the case for the DiPro-Perm test (Wei, Lee, Wichers, and Marron 2016) for which the original implementation in the **diproperm** package (Allmon, Marron, and Hudgens 2021) yields non-reproducible results. Moreover, the implementations of the multi-sample cross-match test of Petrie (2016) and the multi-sample Mahalanobis cross-match test (MMCM) of Mukherjee, Agarwal, Zhang, and Bhattacharya (2022) in the **multicross** package (Agarwal, Bhattacharya, and Zhang 2020) could not be used due to the output format that made it impossible to access the test statistic and p value. More details on the new implementations compared to the aforementioned versions can be found in Section 4.

Table 2 gives an overview of all wrapper functions included in the package. For each method, the original implementation, the new function name, and the applicability to data with a target variable, numerical data, categorical data, and multiple samples are given. Note that the applicability statements refer to the specific implementation of the method. Some of the methods are, in theory, applicable to a broader range of data types than those implemented. More details on the methods and their implementation can be found in Section 4.

Table 3 gives an overview of the newly implemented methods and their applicability. A few of these methods were already implemented in another programming language, as described in the implementation details in Sectioin 4.

Method	Original function	New function	y	Num	Cat	k>2	Sec.
Bahr (1996)	`cramer::cramer.test()` (Franz 2024)	`Bahr()`	✗	✓	✗	✗	4.1
Ball divergence (Pan, Tian, Wang, and Zhang 2018)	`Ball::bd.test()` (Zhu, Pan, Zheng, and Wang 2021)	`BallDivergence()`	✗	✓	✗	✓	4.2

Method	Original function	New function	y	Num	Cat	k>2	Sec.
Baringhaus and Franz (2010)	`cramer::cramer.test()` (Franz 2024)	`BF()`	✗	✓	✗	✗	4.1
Classifier Two-Sample Test (Lopez-Paz and Oquab 2017)	`Ecume::classifier_test()` (Roux de Bezieux 2024)	`C2ST()`	✗	✓	✓	✓	4.3
Chen, Chen, and Su (2018)	`gTests::g.tests()` (Chen and Zhang 2017)	`CCS()`	✗	✓	✗	✗	4.4
Chen and Friedman (2017)	`gTests::g.tests()` (Chen and Zhang 2017)	`CF()`	✗	✓	✗	✗	4.4
Cramér test (Baringhaus and Franz 2004)	`cramer::cramer.test()` (Franz 2024)	`Cramer()`	✗	✓	✗	✗	4.1
DISCO (Rizzo and Székely 2010)	`energy::eqdist.test()` (Rizzo and Szekely 2024)	`DISCOB()`, `DISCOF()`	✗	✓	✗	✓	4.6
Energy statistic (Székely and Rizzo 2017)	`energy::eqdist.test()` (Rizzo and Szekely 2024)	`Energy()`	✗	✓	✗	✓	4.1
Friedman and Rafsky (1979)	`gTests::g.tests()` (Chen and Zhang 2017)	`FR()`	✗	✓	✗	✗	4.4
Chen, Dou, and Qiao (2013)	`gTests::g.tests_cat()` (Chen and Zhang 2017)	`FR_cat()`, `CF_cat()`, `CCS_cat()`, `ZC_cat()`	✗	✗	✓	✗	4.5
FS test (Paul, De, and Ghosh 2022b)	`HDLSSkST::FStest()` (Paul, De, and Ghosh 2022a)	`FStest()`	✗	✓	✗	✓	4.7
Song and Chen (2023a)	`kerTests::kertests()` (Song and Chen 2023c)	`GPK()`	✗	✓	✗*	✗	4.8
Hediger, Michel, and Näf (2022)	`hypoRF::hypoRF()` (Hediger, Michel, and Naef 2024)	`HMN()`	✗	✓	✓	✗	4.9
KMD (Huang and Sen 2024)	`KMD::KMD()`, `KMD::KMD_test()` (Huang 2022)	`KMD()`	✗	✓	✗*	✓	4.10
Maximum Mean Discrepancy (MMD) (Gretton *et al.* 2006)	`kernlab::kmmd()` (Karatzoglou, Smola, Hornik, and Zeileis 2004)	`MMD()`	✗	✓	✗*	✗	4.11
Mukhopadhyay and Wang (2020b)	`LPKsample::GLP()` (Mukhopadhyay and Wang 2020a)	`MW()`	✗	✓	✗*	✓	4.12
RISE (Zhou and Chen 2023)	`GraphRankTest::RISE()` (Zhou and Chen 2025)	`RISE()`	✗	✓	✗	✗	4.13
RI test (Paul *et al.* 2022b)	`HDLSSkST::RItest()` (Paul *et al.* 2022a)	`RItest()`	✗	✓	✗	✓	4.7
Cross-match test (Rosenbaum 2005)	`crossmatch::crossmatch()` (Heller, Small, and Rosenbaum 2024)	`Rosenbaum()`	✗	✓	✗	✗	4.14
Song and Chen (2022)	`gTestsMulti::gtestsmulti()` (Song and Chen 2023b)	`SC()`	✗	✓	✗	✓	4.15
Wasserstein distance	`Ecume::wasserstein_permut()` (Roux de Bezieux 2024)	`Wasserstein()`	✗	✓	✗	✗	4.16

Zhang and Chen (2022)	`gTests::g.tests()` (Chen and Zhang 2017)	`ZC()`	✗	✓	✗	✗	4.4

Table 2: Implemented wrapper functions. y: Can the method deal with a target variable in the dataset? Num: Is the method as implemented applicable to numeric data? Cat: Is the method as implemented applicable to categorical data? $k > 2$: Is the method as implemented applicable to more than two datasets at a time? Sec.: Reference to the corresponding section. ✗*: Method is, in theory, applicable, but implementation does not work in this case. ✓*: Implementation can be used, although this case is not described in the literature.

Method	New function	y	Num	Cat	k>2	Sec.
Biau and Gyorfi (2005)	`BG()`	✗	✓	✗	✓	4.17
Biswas and Ghosh (2014)	`BG2()`	✗	✓	✗	✗	4.18
Biswas, Mukhopadhyay, and Ghosh (2014)	`BMG()`	✗	✓	✗	✓	4.19
Barakat, Quade, and Salama (1996)	`BQS()`	✗	✓	✗	✗	4.20
Constrained Minimum Distance (Tatti 2007)	`CMDistance()`	✗	✗	✓	✗	4.21
DiProPerm test (Wei *et al.* 2016)	`DiProPerm()`	✗	✓	✗	✗	4.22
Deb and Sen (2021)	`DS()`	✗	✓	✗	✗	4.23
Engineer metric	`engineerMetric()`	✗	✓	✗	✗	4.24
Ganti, Gehrke, Ramakrishnan, and Loh (1999)	`GGRL()`	✓	✓	✗*	✗	4.25
Jeffreys divergence	`Jeffreys()`	✗	✓	✗	✗	4.26
Li, Hu, and Zhang (2022)	`LHZ()`	✗	✓	✗	✗	4.27
Mukherjee *et al.* (2022)	`MMCM()`	✗	✓	✓*	✓	4.28
Ntoutsi, Kalousis, and Theodoridis (2008)	`NKT()`	✓	✓	✗	✗	4.25
Alvarez-Melis and Fusi (2020a)	`OTDD()`	✓	✓	✓	✗	4.29
Petrie (2016)	`Petrie()`	✗	✓	✓*	✓	4.28
Schilling (1986) and Henze (1988)	`SH()`	✗	✓	✗	✗	4.30
Yu, Martin, Rothman, Zheng, and Lan (2007)	`YMRZL()`	✗	✓	✓	✗	4.31

Table 3: Newly implemented functions. y: Can the method deal with a target variable in the dataset? Num: Is the method as implemented applicable to numeric data? Cat: Is the method as implemented applicable to categorical data? $k > 2$: Is the method as implemented applicable to more than two datasets at a time? Sec.: Reference to the corresponding section. ✗*: Method is, in theory, applicable, but implementation does not work in this case. ✓*: Implementation can be used, although this case is not described in the literature.

2.2. Specific data types

Note that most implementations listed in Table 2 and Table 3 are only applicable to either numerical or categorical data. Exceptions to this are the classifier-based methods `HMN()` and

`C2ST()`, which can handle both data types simultaneously as long as the selected classifier can do so. The `MMD()` implementation can also handle both data types, but a matching kernel function has to be implemented. For the graph-based tests (FR, CF, CCF, ZC, MMCM, Petrie) as well as for OTDD, an appropriate distance function has to be supplied. For FR, CF, CCF, and ZC, depending on whether there are ties or not in the distances, either the implementation for numeric data (in case of no ties) or for categorical data (in case of ties) should be selected. Some of the other methods might also be applicable if no ties are present in the distance matrix of the datasets. A detailed list of which methods are applicable in which cases is also available on the help page `?DataSimilarity`.

The restriction to either numeric or categorical data can be a limitation in real-world applications, where data is often mixed-scaled. It is, however, inherited from the underlying literature, where the mixed-scale case is rarely discussed, and data is typically (at least implicitly) assumed to be either numeric or categorical. In some cases, the choice of an appropriate distance or kernel function is left up to the user, which allows the use of one that is appropriate for mixed-scale datasets. However, to our knowledge, there are no reliable empirical or theoretical results on the performance of the methods for that case yet.

Similarly, the literature mostly only distinguishes between numeric (continuous) and categorical data without a further distinction between nominal and ordinal data in the categorical case. A distinction between ordinal and nominal data might, however, also be critical for the choice of appropriate distance or kernel functions or other parameters of the method, as for example demonstrated by a simulation study using categorical data (Stolte, Rahnenführer, and Bommert 2026a). There, for one scenario, data was generated with five classes and the original implementations for `MMCM()` and `Petrie()` were used, which only allowed the Euclidean distance as the distance function. It could be shown that with this choice, the methods were considerably more sensitive in detecting alternatives, where the probabilities increased with increasing class labels, than for increasing the probability for one class and decreasing that of the neighboring class accordingly. This might be desirable if data is assumed to be ordinal, as then a change that only affects neighboring classes can be seen as minor compared to the overall change. But if we assume the classes to be unordered, we should select a distance function that treats changes in the probability of the classes independently of the assigned class labels.

Another issue that is common in practice but not well discussed in the related literature is missing values. To our knowledge, there are no recommendations specific to any of the implemented methods on how to handle missing values. The implementations of the **DataSimilarity** package omit missing values but warn the user in that case. Depending on the application, users might consider alternative strategies for handling missing values like supplying adapted distance or graph functions, or using imputation before comparing the datasets.

2.3. In- and output formats

In the implementation within the **DataSimilarity** package, each method gets two (or more) datasets as its first input parameters. After that, arguments specific to the method follow. For example, many methods perform a permutation test for which the number of permutations (`n.perm`) has to be specified. The defaults for those parameters of the methods are chosen based on the simulation results of Stolte *et al.* (2026a) and Stolte, Rahnenführer, and Bommert (2026b). Some of the existing implementations already include setting a random seed, and

some do not. Therefore, for uniformity, the new methods all include a random seed argument. The default for this seed is always set to `NULL`, which corresponds to not setting a new seed. If instead a numeric value is supplied, the random seed is set to the supplied value for reproducibility.

The output of each function is of class '`htest`' and includes

- `statistic`: The test statistic
- `parameter` (optional): A parameter specifying the null distribution (e.g., degrees of freedom for a χ^2 distribution).
- `p.value`: The p value (if an asymptotic or permutation / bootstrap test is performed).
- `estimate`: The sample estimate(s) (if available, e.g., the unstandardized edge count for edge-count tests, `NULL` for many methods).
- `alternative`: The alternative hypothesis. For two datasets, this is $F_1 \neq F_2$, for k datasets it is $\exists i \neq j \in \{1, \ldots, k\} : F_i \neq F_j$.
- `data.name`: Names of the supplied datasets.
- Further elements specific to the method (optional), e.g., the variable importances for the test of Hediger *et al.* (2022).

We use the '`htest`' class as it is widely adopted for storing results of hypothesis tests in R, and most of the implemented methods are two- or k-sample tests. Objects of class '`htest`' will be automatically printed in an appealing format using the `print.htest()` function from the **stats** package. For methods for which no test is performed, the `p.value` is set to `NULL`. This allows pretty printing of the results and a unified output format for the corresponding functions. For many of the newly implemented permutation tests, we use the `boot()` function from the **boot** package that is included in R for implementing the permutation.

There is one exception from the unified output format for the `DISCOF()` function. In that case, we decided to keep the original output format of class '`disco`' that is structured similarly to the remaining '`htest`' objects. The reason for this was that a specific printing method exists for '`disco`' objects that displays the results in an ANOVA table style, which we considered more appealing than the normal '`htest`' formatting in this specific case. The test statistic and p value can be accessed as for the '`htest`' objects, such that we do not see any disadvantages for the uniform usability. Moreover, the three convenience functions `gTests()`, `gTestsMulti()`, `kerTests()`, and the multiscale versions of `FStest()` and `RItest()` return multiple test statistics and therefore the output is a list of the same structure but not of class '`htest`'.

In typical applications, users should choose a test a priori and not based on test results. Therefore, the new functions perform exactly one test and return only the results corresponding to that single test. Some of the existing implementations used to perform multiple tests based on the same metrics or always returned the asymptotic p value in addition to a permutation p value. This could lead to unscientific practices like choosing the test based on the desired result. As an exception, for implementations that output multiple related tests, we offer wrapper functions that also perform these multiple tests (see the above-listed convenience functions). Often, conducting them at once is computationally faster than performing

each test individually when large parts of the calculation are the same. This option might be useful in certain situations where multiple tests need to be applied to the same data, e.g., when performing method comparison studies. The wrapper functions return lists structured in the same way as the '`htest`' objects that, instead of the single test statistic and p value include vectors of test statistics and p values. We advise against applying multiple related tests for the same hypothesis and deciding on one test afterwards when conducting inference for a specific real-life application. The next sections will provide some helpful considerations for the method choice in practice.

2.4. The `DataSimilarity()` and `findSimilarityMethod()` functions

To facilitate the choice and usage of an appropriate method, two convenience functions, `findSimilarityMethod()` and `DataSimilarity()`, are supplied. `findSimilarityMethod()` can be used to find the set of methods that satisfy all given requirements on the applicability of the methods. The `DataSimilarity()` function can be used to call each method by supplying the method name via the `method` argument. Like the functions for the individual methods, it takes two or more datasets as its first input. Moreover, the `method` argument has to be supplied to specify the method, and arguments specific to the method can be passed.

The arguments of `findSimilarityMethod()` specify which of the 22 theoretical criteria (see Section 1, Stolte *et al.* (2024)) must be fulfilled. These can be passed to the function by setting the corresponding arguments to `TRUE`. `findSimilarityMethod()` then returns by default the function names for all implemented and suitable methods. By setting `only.names = FALSE`, the full information on which criteria the method fulfills can be retrieved.

These convenience functions are useful in situations where users have datasets at hand that they want to compare, but do not know which method to use, or if users want to conveniently compare multiple methods. In the case where users want to apply a specific method, they can also use the corresponding function directly and find additional information on that method in the last section of this vignette. The typical workflow for using the `findSimilarityMethod()` and `DataSimilarity()` functions is demonstrated in the next section and also described by a 'Getting Started' vignette within the package.

3. Method choice

There are different aspects to take into account when choosing a suitable method. In the following, we give a roadmap for the method choice using the **DataSimilarity** package, and specifically the `findSimilarityMethod()` and `DataSimilarity()` functions. The description here is kept general and illustrated in the following section for a concrete example. Table 4 gives a compact step-by-step overview of the proposed COMPARE procedure (**C**haracteristics of datasets, **O**bjective of comparison, **M**ethod requirements, **P**roduction of method list, **A**ssessment of computational complexity, **R**anking of methods, **E**stimation and testing) for finding the best-suited method. Each step is described in more detail in the following. The first three steps are guiding through all considerations necessary for specifying the arguments of the `findSimilarityMethod()` function which is applied in step 5., and step 6. and 7. specify how to get from the output of `findSimilarityMethod()` to the final result. An illustration and further details on the individual steps can be found in the Getting Started vignette.

	Step	Description
1.	Characteristics of datasets	Evaluate number of datasets, scale level of variables, presence of target variable, dimensions of datasets
2.	Objective of comparison	Is the focus mainly on a test decision, or is the (dis)similarity value itself of interest and should be interpretable?
3.	Method requirements	Are additional properties of the method regarding invariances, metric properties, or consistency of a test of relevance?
4.	Production of method list	Call `findSimilarityMethod()` with the criteria identified in steps 1.–3.
5.	Assessment of computational complexity	Are the identified methods feasible with regard to their runtime and memory consumption?
6.	Ranking of methods	Compare the remaining methods regarding their performance in simulation studies and select the best
7.	Estimation and testing	Apply the chosen method(s)

Table 4: Step-by-step procedure for choosing the best method for a given application.

4. Implementation details

4.1. `Bahr()`, `BF()`, `Energy()`

The energy statistic is a popular two- and k-sample statistic based on interpoint distances. The k-sample statistic is defined as

$$T_{\text{Energy}} = \sum_{1 \leqslant i < j \leqslant k} \frac{n_i n_j}{n_i + n_j} \left(\frac{2}{n_i n_j} \sum_{u=1}^{n_i} \sum_{v=1}^{n_j} \|X_u^{(i)} - X_v^{(j)}\|_2 - \frac{1}{n_i^2} \sum_{u=1}^{n_i} \sum_{v=1}^{n_i} \|X_u^{(i)} - X_v^{(i)}\|_2 - \frac{1}{n_j^2} \sum_{u=1}^{n_j} \sum_{v=1}^{n_j} \|X_u^{(j)} - X_v^{(j)}\|_2 \right).$$

For a comprehensive review of the literature on the energy statistic and its applications, please refer to Székely and Rizzo (2017). A permutation test can be performed based on the energy statistic. In the two-sample case, the energy statistic is equal to two times the Cramér test statistic of Baringhaus and Franz (2004), and therefore, the tests are equivalent. However, a bootstrap instead of a permutation test is proposed for the Cramér test. Baringhaus and Franz (2010) propose a test statistic that generalizes the Cramér test statistic by using a continuous function ϕ such that $\phi(\|x-y\|^2)$ is a negative definite kernel instead of the Euclidean distances. Different examples for ϕ are given, including as special cases the Cramér test, the test by Bahr (1996), and the test by Szabo, Boucher, Carroll, Klebanov, Tsodikov, and Yakovlev (2002). Overall, $\phi(z) = \log(1+z)$ is recommended for general alternatives based on a simulation study, and the Cramér test is recommended for location alternatives. The tests of Baringhaus and Franz (2010) are implemented in the **cramer** package (Franz 2024). The new implementation is a simple wrapper to unify input and output naming and types. The energy statistic is implemented in the **energy** package (Rizzo and Szekely 2024). For the corresponding wrapper, the input type was changed to a greater extent since the original implementation had the pooled sample and the sample sizes as the input. The **energy** implementation outsourced the calculation of the energy statistic to `C`, which gives it a notable advantage with regard to computing time over the **cramer** implementation.

4.2. `BallDivergence()`

The Ball divergence (Pan *et al.* 2018) measures the difference between two probability measures. It is defined as the square of the measure difference over a given closed ball collection. It can be estimated as

$$\widehat{\text{BD}} = A + C,$$

where

$$A = \frac{1}{n_1^2} \sum_{i,j=1}^{n_1} \left(A_{ij}^{(1)} - A_{ij}^{(2)} \right)^2,$$

$$C = \frac{1}{n_2^2} \sum_{l,m=1}^{n_2} \left(C_{lm}^{(1)} - C_{lm}^{(2)} \right)^2,$$

and

$$A_{ij}^{(1)} = \frac{1}{n_1} \sum_{u=1}^{n_1} \mathbb{1}(X_u^{(1)} \in \bar{B}(X_i^{(1)}, d(X_i^{(1)}, X_j^{(1)}))),$$
$$A_{ij}^{(2)} = \frac{1}{n_2} \sum_{v=1}^{n_2} \mathbb{1}(X_v^{(2)} \in \bar{B}(X_i^{(1)}, d(X_i^{(1)}, X_j^{(1)}))),$$
$$C_{lm}^{(1)} = \frac{1}{n_1} \sum_{u=1}^{n_1} \mathbb{1}(X_u^{(1)} \in \bar{B}(X_l^{(2)}, d(X_l^{(2)}, X_m^{(2)}))),$$
$$C_{lm}^{(2)} = \frac{1}{n_2} \sum_{v=1}^{n_2} \mathbb{1}(X_v^{(2)} \in \bar{B}(X_l^{(2)}, d(X_l^{(2)}, X_m^{(2)}))),$$

with $\bar{B}(X_i^{(l)}, d(X_i^{(l)}, X_j^{(l)}))$ denoting the closed Ball around $X_i^{(l)}$ with radius equal to the distance d of the points $X_i^{(l)}$ and $X_j^{(l)}, l \in \{1, 2\}$. Therefore, the first part of the Ball divergence, A, consists of squared distances of proportions of data points from the first sample lying within closed balls around data points from the first sample and of data points from the second sample lying within closed balls around data points from the first sample. The second part, C, consists of squared distances of proportions of data points from the first sample lying within closed balls around data points from the second sample and of data points from the second sample lying within closed balls around data points from the second sample. For both parts, the mean over all such Balls with radii equal to the distances of the center point of the ball to all other points from the same sample is taken. For multiple samples, the pairwise test statistics can be summarized by summing up the pairwise divergences, or by taking the maximum of sums of the Ball divergences from each sample to all other samples, or by summing up the largest $k - 1$ pairwise Ball divergences.
The implementation here is a wrapper around the `bd.test()` function from the **Ball** package (Zhu *et al.* 2021). In contrast to the original implementation, the new wrapper returns an object of class '`htest`' in the multi-sample case, although, in that case, no test is conducted. Moreover, only the summarized statistic according to the specified `kbd.type`, which determines how the pairwise Ball divergences are summarized, is returned.

4.3. `C2ST()`

The general idea of Lopez-Paz and Oquab (2017) is to use a classifier to determine which of two or more datasets an observation belongs to. The *classifier two-sample test (C2ST)* uses the classification accuracy of this classifier as its test statistic.
The C2ST consists of five steps:

1. Construct the dataset consisting of the samples from all datasets labeled with their membership to each dataset.
2. Assign the observations of the dataset constructed in 1. randomly to a training and test set.
3. Train a classifier that predicts to which of the datasets $X^{(j)}, j = 1, \ldots, k$, an observation belongs.

4. Calculate the C2ST statistic, which is the accuracy on the test set. The accuracy should be close to the chance level for $F_1 = \cdots = F_k$, and it should be greater than the chance level if $\exists i \neq j \in \{1, \ldots, k\} : F_i \neq F_j$ since in the latter case the classifier should identify distributional differences between the samples.

5. Calculate a p value using a binomial test for comparing the accuracy to the chance level.

Maximizing the power of a C2ST is a trade-off between using a large training set to optimize the classifier and a large test set to better evaluate the performance of the classifier.
The test statistic is interpretable as the percentage of samples that are correctly classified on the unseen test data. The above-mentioned test of Hediger *et al.* (2022) can be seen as a special case of the general framework proposed by Lopez-Paz and Oquab (2017). One difference in the implementation of the tests is that for the C2ST, categorical data is dummy coded, while for the test of Hediger *et al.* (2022) the categorical variables are passed to `ranger::ranger()` directly. Moreover, the use of OOB predictions and feature importance is specific to the random forest-based test and cannot be used for all of the available classifiers for the C2ST. Further, the C2ST uses the accuracy as its test statistic, while the test of Hediger *et al.* (2022) uses the classification error, i.e., 1 − accuracy.

For the C2ST, the classifier can be specified by the user and defaults to K-nearest neighbors. Possible options are all models accepted by `caret::train()`. For a list of these classification models, call e.g.

```
R> names(caret::getModelInfo())[sapply(caret::getModelInfo(), function(x) {
+    "Classification" %in% x$type
+  })]
```

4.4. `CCS()`, `CF()`, `FR()`, `ZC()`

The tests by Friedman and Rafsky (1979), Chen and Friedman (2017), Chen *et al.* (2018), and Zhang and Chen (2022) are graph-based two-sample tests that use the edge counts in a similarity graph like the (K-)MST on the pooled sample. They make use of the number of edges that connect points within the first sample, R_1, the number of edges that connect points within the second sample, R_2, and the number of edges that connect points from different samples R_{12}. The original edge-count test by Friedman and Rafsky (1979) takes the standardized between-sample edge-count

$$T_{\text{FR}} = \frac{R_{12} - \mathsf{E}_{H_0}(R_{12})}{\sqrt{\mathsf{VAR}_{H_0}(R_{12})}}$$

as its test statistic. The expectation and variance under the null can be calculated analytically. Chen and Friedman (2017) noted that this has low power against scale alternatives and proposed the *generalized edge-count test* using

$$T_{\text{CF}} = (R_1 - \mathsf{E}_{H_0}(R_1), R_2 - \mathsf{E}_{H_0}(R_2))\, \mathsf{COV}_{H_0}^{-1}\left(\begin{pmatrix} R_1 \\ R_2 \end{pmatrix}\right) \begin{pmatrix} R_1 - \mathsf{E}_{H_0}(R_1) \\ R_2 - \mathsf{E}_{H_0}(R_2) \end{pmatrix}.$$

Chen *et al.* (2018) found some problems with the original edge-count test for unequal sample sizes of the two datasets, based on which they proposed the *weighted edge-count test* using the weighted edge-counts

$$R_w = \frac{n_1}{N} R_1 + \frac{n_2}{N} R_2,$$

where n_1 denotes the sample size of the first dataset, n_2 the sample size of the second dataset, and $N = n_1 + n_2$ the total sample size in the pooled sample. The *weighted edge-count test* statistic is then defined as the standardized weighted edge count

$$T_{\text{CCS}} = \frac{R_w - \mathsf{E}_{H_0}(R_w)}{\sqrt{\mathsf{VAR}_{H_0}(R_w)}}.$$

Lastly, the *max-type edge count* (Zhang and Chen 2022) test (based on the method of Chu and Chen (2019)) additionally uses the difference of the edge counts in the samples, i.e.,

$$R_d = R_1 - R_2.$$

Its test statistic is defined as

$$T_{\text{ZC}} = \max\left(\kappa\frac{R_w - \mathsf{E}_{H_0}(R_w)}{\sqrt{\mathsf{VAR}_{H_0}(R_w)}}, \left|\frac{R_d - \mathsf{E}_{H_0}(R_d)}{\sqrt{\mathsf{VAR}_{H_0}(R_d)}}\right|\right),$$

where κ is a constant that has to be chosen before performing the test. $\kappa \in \{1, 1.14, 1.31\}$ is recommended based on a small power simulation for normal data with shift or scale alternatives.
Wrapper functions around `g.tests()` from the **gTests** package (Chen and Zhang 2017) are implemented. These do not need a pre-calculated graph as input but allow specifying a distance function (`dist.fun`) and a function for calculating a similarity graph (`graph.fun`) and then calculating the similarity graph internally. The new input also includes both datasets. We find this more intuitive and less error-prone than supplying an edge matrix and two vectors of indices specifying the dataset membership as for the original `g.tests()` function. The new implementation forces the user to choose one of the tests first and then perform it, instead of performing all tests at once. Moreover, the users have to decide whether they want to perform the permutation test or the approximate test.

For the Friedman-Rafsky test, there is an additional implementation in the **GSAR** package (Rahmatallah, Zybailov, Emmert-Streib, and Glazko 2017), but there, the test statistic is standardized by the empirical mean and standard deviation rather than the theoretical mean and standard deviation of the test statistic under the null hypothesis as proposed in the original article. Therefore, we use the **gTests** implementation here.

4.5. `CCS_cat()`, `CF_cat()`, `FR_cat()`, `ZC_cat()`

These methods are adaptations of the previously mentioned edge-count tests for categorical data. With categorical data, the problem of ties in the distance matrix arises. Ties lead to non-unique solutions for the similarity graph construction and, therefore, also to non-unique values of the proposed test statistics. This can be solved by either taking the union of all optimal graphs and calculating the respective statistic on this union graph, or by averaging the test statistic values over all optimal graphs. The new implementation of the categorical graph-based tests is again a wrapper function that includes the calculation of the edge matrix. For this, the function `getGraph()` from the **gTests** package is used. Therefore, the choice of the similarity graph is restricted to the K-nearest neighbors and the K-MST. Still, a distance function can be supplied. By default, this is the sum of unequal classes. The calculation of the frequency table of all observations and the similarity graph on this are performed internally;

thus, again, only the datasets have to be supplied by the user. Moreover, the method for aggregating the graphs has to be supplied. Possible options are averaging (`"a"`) and union (`"u"`) over graphs.

4.6. `DISCOB()`, `DISCOF()`

Rizzo and Székely (2010) show that the energy test can be seen as the treatment sum of squares in an ANOVA interpretation of the k-sample problem. As the measure of dispersion for univariate or multivariate responses based on all pairwise distances between sample elements for ANOVA

$$d_\alpha(X^{(i)}, X^{(j)}) = \frac{n_i n_j}{n_i + n_j}[2g_\alpha(X^{(i)}, X^{(j)}) - g_\alpha(X^{(i)}, X^{(i)}) - g_\alpha(X^{(j)}, X^{(j)})]$$

is proposed with

$$g_\alpha(X^{(i)}, X^{(j)}) = \frac{1}{n_i n_j} \sum_{u=1}^{n_i} \sum_{v=1}^{n_j} \|X_u^{(i)} - X_v^{(j)}\|_2^\alpha, i, j \in \{1, \ldots, k\}.$$

With this, Rizzo and Székely (2010) derive their so-called *distance components (DISCO) decomposition* for $\alpha \in (0, 2]$. It partitions the total dispersion in the samples

$$T_\alpha = \frac{N}{2} \sum_{i,j=1}^{N} g_\alpha(X^{(i)}, X^{(j)}),$$

into components

$$T_\alpha = S_\alpha + W_\alpha$$

analogous to the variance components in ANOVA. The between-sample measure of dispersion S_α and the within-sample measure of dispersion W_α, respectively, are defined as

$$S_\alpha = \sum_{1 \leqslant i < j \leqslant k} \frac{n_i + n_j}{2N} d_\alpha(X^{(i)}, X^{(j)}),$$
$$W_\alpha = \sum_{i=1}^{k} \frac{n_i}{2} g_\alpha(X^{(i)}, X^{(i)}).$$

The between-sample measure of dispersion S_α can be used directly in a k-sample permutation test (`DISCOB()`). Alternatively, the statistic

$$F_\alpha = \frac{S_\alpha/(k-1)}{W_\alpha/(N-k)}$$

can be used in a k-sample permutation test (`DISCOF()`). For each index $\alpha \in (0, 2)$, this determines a nonparametric test for the multi-sample problem that is statistically consistent against general alternatives. For $\alpha = 2$, it equals the usual ANOVA F-test. The choice of the index α is difficult. In general, the computational costs for calculating Gini means g_α, in terms of which the test statistic can be formulated, are $\mathcal{O}(N^2)$. For $\alpha = 1$, it can be linearized, and computation time reduces to $\mathcal{O}(N \log N)$. The simplest and most natural choice for α is one. For heavy-tailed distributions, a small α is recommended.

The test is implemented by permutation bootstrap in the R package `energy` (Rizzo and Szekely 2024). The new implementations of the between-sample and of the DISCO F-test are wrappers, which mainly unify the inputs and outputs that differed between the two tests in the original implementation. Moreover, the input format is again changed from the pooled sample and the dataset labels to the individual datasets.

4.7. `FStest()`, `RItest()`

Paul *et al.* (2022b) propose distribution-free k-sample tests intended for the high dimension low sample size (HDLSS) setting. The tests are based on clustering the pooled sample and comparing the resulting clustering to the true dataset membership via a contingency table. If the datasets come from the same distribution, the cluster and dataset membership are independent, while if the datasets come from different distributions, the clustering depends on the true dataset membership. As a clustering algorithm, Paul *et al.* (2022b) suggest using K-means based on the generalized version of the *mean absolute difference of distances (MADD)*

$$\rho_{h,\varphi}(z_i, z_j) = \frac{1}{N-2} \sum_{m \in \{1,\dots,N\} \setminus \{i,j\}} \left| \varphi_{h,\psi}(z_i, z_m) - \varphi_{h,\psi}(z_j, z_m) \right|,$$

as proposed by Sarkar and Ghosh (2020) for the HDLSS setting. Here, $z_i, i = 1, \dots, N$, denote realizations from the pooled sample and

$$\varphi_{h,\psi}(z_i, z_j) = h\left(\frac{1}{p} \sum_{l=1}^{p} \psi |z_{il} - z_{jl}| \right),$$

where $h : \mathbb{R}^+ \to \mathbb{R}^+$ and $\psi : \mathbb{R}^+ \to \mathbb{R}^+$ are continuous and strictly increasing functions. $\psi(t) = t^2$, $\psi(t) = 1 - \exp(-t)$, $\psi(t) = 1 - \exp(-t^2)$, $\psi(t) = \log(1 + t)$, and $\psi(t) = t$ are considered in combination with $h(t) = \sqrt{t}$ and $h(t) = t$. The number of clusters has to be chosen in advance. A natural choice is to set the number of clusters to the number of datasets k. For the RI test, the Rand index of the clustering is used as a test statistic. It is zero when the clustering is perfect, i.e., when the cluster membership is a permutation of the true dataset membership. The test rejects for low values since the Rand index should take higher values when all clusters have similar distributions of class labels. The critical value can be calculated using a generalized hypergeometric distribution. Due to the discreteness of the Rand index, Paul *et al.* (2022b) propose to use a randomized test. For the FS test, the generalized Fisher's test statistic for testing for independence in a $k \times \ell$ contingency table is used. Again, a randomized test using the generalized hypergeometric distribution to find the critical values is proposed.

Paul *et al.* (2022b) additionally propose modified versions of the tests (MRI, MFS test). For these, the number of clusters is estimated from the data using the Dunn index, since setting the number of clusters to k might fail in case of multimodal distributions where a larger number of clusters might be required, where then multiple clusters can correspond to one dataset.

Moreover, multiscale versions of the tests are presented (MSRI, MSFS test) for the case where the number of clusters is unclear. The respective tests are then performed for different numbers of clusters, and the results are aggregated using a Bonferroni adjustment for the individual tests. Still, an upper limit for the number of clusters to be considered must be chosen. The implementation also includes aggregated tests (AFS / ARI test) that perform all

pairwise FS / MFS or RI / MRI tests, respectively, on the samples and aggregate the results by taking the minimum test statistic value and applying a multiple testing procedure.
The tests are implemented in the R package **HDLSSkST** (Paul *et al.* 2022a). The main difference between the new wrapper functions and the original implementation is that the modified and multiscale versions of the RI and FS tests can be performed with the same function as the original tests. The test can be chosen via the newly introduced `version` argument of the `FStest()` and `RItest()` functions. One advantage of this is that the input and output formats are unified between the versions of the test. In the original implementation of the test, the elements of the output list differ both content-wise and also in their names between the tests. Moreover, the input of the tests differs slightly between the original functions for the different tests. The input is also unified to match the input of the other functions in the **DataSimilarity** package and, therefore, consists simply of the datasets instead of a pooled data matrix, a vector with the dataset affiliation of each observation, and a vector of the sample sizes. We think this is easier to understand and less error-prone from a user perspective.

4.8. `GPK()`

Song and Chen (2023a) propose another kernel-based test for which they decompose the squared MMD estimator as

$$\widehat{\mathrm{MMD}}^2 = \alpha + \beta - 2\gamma,$$

where

$$\alpha = \frac{1}{n_1(n_1-1)} \sum_{i=1}^{n_1} \sum_{\substack{j=1 \\ j\neq i}}^{n_1} K(X_i^{(1)}, X_j^{(1)}),$$

$$\beta = \frac{1}{n_2(n_2-1)} \sum_{i=1}^{n_2} \sum_{\substack{j=1 \\ j\neq i}}^{n_2} K(X_i^{(2)}, X_j^{(2)}),$$

$$\gamma = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} K(X_i^{(1)}, X_j^{(2)}).$$

As a new statistic, they propose to use

$$\mathrm{GPK} = (\alpha - \mathsf{E}_{H_0}(\alpha), \beta - \mathsf{E}_{H_0}(\beta))\mathsf{COV}_{H_0}^{-1}\left(\begin{pmatrix}\alpha \\ \beta\end{pmatrix}\right)\begin{pmatrix}\alpha - \mathsf{E}_{H_0}(\alpha) \\ \beta - \mathsf{E}_{H_0}(\beta)\end{pmatrix}.$$

The GPK can be decomposed into $\mathrm{GPK} = Z_W^2 + Z_D^2$, where Z_W and Z_D are the standardized versions (with expectation and variance under H_0) of

$$W = \frac{n_1}{N}\alpha + \frac{n_2}{N}\beta,$$
$$D = n_1(n_1-1)\alpha - n_2(n_2-1)\beta.$$

Based on this observation, they further generalize W to

$$W_r = r\frac{n_1}{N}\alpha + \frac{n_2}{N}\beta$$

and Z_W to $Z_{W,r}$. Fast tests based on $Z_{W,r}$ are proposed as the asymptotic distribution of $Z_W = Z_{W,1}$ is complicated, but that of $Z_{W,r}, r \neq 1$, is a standard normal under mild assumptions. One fast test fGPK uses the Bonferroni adjusted test result of the tests based on Z_D, $Z_{W,1.2} =: ZW_1$ and $Z_{W,0.8} =: ZW_2$, the other fast test fGPK$_M$ uses the Bonferroni adjusted test result of the tests based on $Z_{W,1.2}$ and $Z_{W,0.8}$. For GPK (as well as for fGPK and fGPK$_M$), a permutation test can be performed.
The new implementation `GPK()` based on the `kertests()` function from the **kerTests** package (Song and Chen 2023c) performs by default the fast test version instead of a permutation test, and the bandwidth parameter σ of the RBF kernel that is used as the kernel K is chosen via the median heuristic using the function `med_sigma()` of the **kerTests** package. The median heuristic sets the bandwidth of the kernel to the median value of all pairwise distances in the pooled sample (Sriperumbudur, Fukumizu, Gretton, Lanckriet, and Schölkopf 2009). When the fast test is performed, all three test statistics, ZW_1, ZW_2, and Z_D, are returned together with the asymptotic p value if `n.perm = 0` or the permutation p value if `n.perm > 0`, respectively. For the GPK statistic, only the permutation test is available as its null distribution cannot be accessed. Therefore, if the number of permutations is set to zero, the fast test is always performed. This holds even if `fast` is set to `FALSE` (with a warning).

4.9. `HMN()`

Hediger *et al.* (2022) provide a two-sample test based on random forests. It is applicable for categorical data and can also be used with numeric variables or a mix of categorical and numeric variables. For this, a pooled dataset is created where each observation is labeled according to its original dataset membership, and a random forest is trained to distinguish between the dataset labels. The idea is that if the datasets are generated from the same distribution, the classification error of the random forest should be close to the chance level; otherwise, the classifier should be able to distinguish between the two distributions, and hence the classification error should be lower than the chance level. One advantage of using random forests as the classifier is that it requires almost no tuning. An asymptotic test is proposed. For this, the pooled dataset has to be split into a training set on which the random forest is trained and a test set on which its classification error is evaluated to ensure that the test is performed on data that is independent of the data on which the classifier was trained. In the implementation, both datasets are split in half to create a training and a test dataset. Alternatively, an out-of-bag (OOB) based permutation test can be performed that does not require data splitting. OOB statistics can be used to increase the sample efficiency compared to the test based on a holdout sample. Both the OOB-based permutation test and the asymptotic version of the test using data splitting are implemented. The test statistic is either the mean of the per-class OOB or test classification errors, or the overall OOB or test classification error over both classes, respectively. In the asymptotic case, a binomial test is performed in case of the overall classification error, or a Z test is performed in case of the mean per-class classification error. Otherwise, a permutation test is performed. The variable importance measures of the random forest can provide additional insights into sources of distributional differences.

The function here is a wrapper around the `hypoRF()` function from the **hypoRF** package (Hediger *et al.* 2024) that only renames arguments for consistency with the other methods. Note that the implemented per-class OOB statistics differ for the permutation test and the approximate test: for the permutation test, the sum of the per-class OOB errors is returned;

for the asymptotic version, the standardized sum is returned.

4.10. `KMD()`

The *kernel measure of multi-sample dissimilarity* (KMD) introduced by Huang and Sen (2024) is a kernel-based test using the association between the variables and the sample membership to quantify the dissimilarity of multiple samples. Denote the dataset membership of each point in the pooled sample $\{Z_1, \ldots, Z_N\}$ by $\{\Delta_1, \ldots, \Delta_N\}$. $\{(\Delta_i, Z_i)\}_{i=1}^N$ can be seen as an i.i.d. sample from $(\tilde{\Delta}, \tilde{Z})$ with distribution μ defined by $\mathrm{P}(\tilde{\Delta} = i) = \pi_i$ and $\tilde{Z}|\tilde{\Delta} = i \sim F_i$, $i = 1, \ldots, k$. Let $(\tilde{Z}_1, \tilde{\Delta}_1), (\tilde{Z}_2, \tilde{\Delta}_2)$ be i.i.d. samples from μ and $(\tilde{Z}, \tilde{\Delta}), (\tilde{Z}, \tilde{\Delta}') \sim \mu$ with $\tilde{\Delta}, \tilde{\Delta}'$ conditionally independent given $\tilde{Z}$. Denote by K a kernel function over $\{1, \ldots, k\}$, e.g., the discrete kernel $K(x, y) := \mathbb{1}(x = y)$. Then, the KMD is defined as

$$\eta(P_1, \ldots, P_k) := \frac{\mathsf{E}\left[K(\tilde{\Delta}, \tilde{\Delta}')\right] - \mathsf{E}\left[K(\tilde{\Delta}_1, \tilde{\Delta}_2)\right]}{\mathsf{E}\left[K(\tilde{\Delta}, \tilde{\Delta})\right] - \mathsf{E}\left[K(\tilde{\Delta}_1, \tilde{\Delta}_2)\right]}.$$

It can be estimated using a similarity graph $\mathcal{G}$, e.g., the K-nearest neighbor (NN) graph or the minimum spanning tree (MST) on the pooled sample. Denote by $(Z_i, Z_j) \in \mathcal{E}(\mathcal{G})$ that there is an edge in $\mathcal{G}$ connecting Z_i and Z_j. Moreover, let o_i be the out-degree of Z_i in $\mathcal{G}$. Then, an estimator for η is defined as

$$\hat{\eta} := \frac{\frac{1}{N}\sum_{i=1}^N \frac{1}{o_i}\sum_{j:(Z_i,Z_j)\in\mathcal{E}(\mathcal{G})} K(\Delta_i, \Delta_j) - \frac{1}{N(N-1)}\sum_{i\neq j} K(\Delta_i, \Delta_j)}{\frac{1}{N}\sum_{i=1}^N K(\Delta_i, \Delta_i) - \frac{1}{N(N-1)}\sum_{i\neq j} K(\Delta_i, \Delta_j)}.$$

An asymptotic and a permutation k-sample test are proposed based on the KMD.
The implementation of the new function `KMD()` combines the calculation of KMD and the corresponding p value using the functions `KMD()` and `KMD_test()`, respectively, from the **KMD** package (Huang 2022). Moreover, the inputs of the new function are simply the individual datasets instead of the pooled data matrix and sample IDs. By default, the asymptotic test is performed (`n.perm = 0`) using a discrete kernel and the K-NN graph with $K = \lceil N/10 \rceil$, where N denotes the total sample size of the pooled sample. The options for the graph are restricted to `"knn"` and `"mst"` by the implementations from the **KMD** package. A user-specified kernel can be used only when a kernel matrix is supplied instead of the keyword `"discrete"` for the `kernel` argument of the new function.

4.11. `MMD()`

The *maximum mean discrepancy* (MMD) uses a kernel mean embedding to define a metric for probability distributions. Kernel mean embeddings extend feature maps ϕ to the space of probability distributions by representing each distribution F as a mean function

$$\phi(F)(\cdot) = \mu_F(\cdot) := \int_{\mathcal{X}} K(x, \cdot)\, \mathrm{d}F(x) = \mathsf{E}_F(K(X, \cdot)),$$

where $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric and positive definite kernel function. A reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ of functions on the domain $\mathcal{X}$ with kernel K is a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$ with dot product $\langle \cdot, \cdot \rangle$ that satisfies the reproducing property

$$\langle f(\cdot), K(x, \cdot)\rangle = f(x) \;\Rightarrow\; \langle K(x, \cdot), K(x', \cdot)\rangle = K(x, x'),$$

such that the linear map from a function to its value at x can be seen as an inner product. Then the kernel mean embedding as given above is a transformation of the distribution F to an element in the reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ corresponding to the kernel K (Muandet, Fukumizu, Sriperumbudur, and Schölkopf 2017). For characteristic kernels, the kernel mean representation captures all information about the distribution F, which implies $\|\mu_{F_1} - \mu_{F_2}\|_{\mathcal{H}} = 0 \Leftrightarrow F_1 = F_2$ (Fukumizu, Bach, and Jordan 2004; Sriperumbudur, Gretton, Fukumizu, Lanckriet, and Schölkopf 2008; Sriperumbudur, Gretton, Fukumizu, Schölkopf, and Lanckriet 2010). Therefore, the MMD measures the difference between two distributions as

$$\text{MMD}(\mathcal{H}, F_1, F_2) = \|\mu_{F_1} - \mu_{F_2}\|_{\mathcal{H}}.$$

Here, the implementation `kmmd()` from the **kernlab** package (Karatzoglou *et al.* 2004) is used. The alternative implementation from the **Ecume** package (Roux de Bezieux 2024) does not include an automatic choice of the kernel parameter. The new implementation adds a permutation test to the **kernlab** implementation.

4.12. `MW()`

For the test of Mukhopadhyay and Wang (2020b), a nonparametrically designed set of orthogonal functions (LP polynomials) is obtained by orthonormalizing a set of functions constructed as orthonormal polynomials of mid-distribution transforms. These are used for the construction of a polynomial kernel of degree 2 that encodes the similarity between two data points in the LP-transformed domain. The values of the kernel Gram matrix are then used as weights on a graph with the pooled sample as vertices. The idea is to cluster points for the graph into k groups that have higher connectivity and compare how closely this clustering is related to the true memberships of the k distributions. Then, the problem reduces to testing independence, which can be accomplished by determining whether all of the LP comeans are zero.
The test is implemented in the **LPKsample** package (Mukhopadhyay and Wang 2020a). The new implementation offers the additional option to sum over all components instead of summing over the significant components only. This might be of interest when using the statistic as a data similarity measure without testing. By default, this is disabled (`sum.all = FALSE`). When only summing over the significant components, the returned test statistic is always equal to zero when no component is significant.

4.13. `RISE()`

Zhou and Chen (2023) define a rank-based two-sample test based on similarity graphs, which can be applied to high-dimensional and non-Euclidean data. They define a sequence of simple similarity graphs $\{G_l\}_{l=0}^{K}$ on the pooled sample via

$$G_{l+1} = G_l \cup G_{l+1}^*$$

with

$$G_{l+1}^* = \arg\max_{G' \in \mathcal{G}_{l+1}} \sum_{(i,j) \in G'} S(Z_i, Z_j),$$

where G_0 has no edges, $\mathcal{G}_{l+1} = \{G' \in \mathcal{G} : G' \cap G_l = \varnothing\}$ with $\mathcal{G}$ the set of graphs that fulfill specific user-defined constraints, and $S(\cdot, \cdot)$ a similarity measure, e.g. the negative Euclidean

distance for Euclidean data, and $Z_1, \ldots, Z_N$ denoting the pooled sample. This includes as special cases the K-nearest neighbor graph, the K-minimum spanning tree, the K-minimum distance non-bipartite pairing, and the K-shortest Hamiltonian path. Then, Zhou and Chen (2023) define the following two graph-based rank matrices $R = (R_{ij})_{i,j=1}^{N}$ using this sequence of graphs. The *graph-induced ranks* are defined as

$$R_{ij} = \sum_{l=1}^{K} \mathbb{1}\left((i,j) \in G_l\right).$$

They can be interpreted as the number of graphs that contain the edge (i, j) in the sequence of graphs. The *overall ranks* are defined as

$$R_{ij} = \text{rank}\left(S\left(Z_i, Z_j\right), G_K\right),$$

where $\text{rank}\left(S\left(Z_i, Z_j\right), G_K\right)$ denotes the rank of $S\left(Z_i, Z_j\right)$ among the values $\{S\left(Z_u, Z_v\right)\}_{(u,v)\in G_K}$ if $(i, j) \in G_k$ and zero otherwise. The overall rank can be interpreted as the rank of the similarity of edges in the graph G_K. Both rank definitions depend on the choice of K. For the test, the symmetrized rank matrix $1/2(R + R^T)$ is used, which is also denoted by R for convenience.

For the test statistic, the within-sample rank sums of the first and second sample are defined as

$$U_x = \sum_{i,j=1}^{n_1} R_{ij}, U_y = \sum_{i,j=n_1+1}^{N} R_{ij}.$$

Using these, the *rank in similarity graph edge-count two-sample test (RISE)* statistic is defined as

$$T_R = (U_x - \mu_x, U_y - \mu_y)\Sigma^{-1}(U_x - \mu_x, U_y - \mu_y)^T,$$

where $\mu_x = \mathsf{E}(U_x)$, $\mu_y = \mathsf{E}(U_y)$, and $\Sigma = \mathbb{C}\text{ov}((U_x, U_y)^T)$. The quantities μ_x, μ_y, and Σ can be calculated explicitly under the permutation null hypothesis. Zhou and Chen (2023) give sufficient conditions under which Σ is invertible and therefore T_R is well-defined. For small samples, the exact permutation null distribution can be used for testing. For large samples and under several assumptions on the similarity graphs, the asymptotic χ^2_2-distribution of T_R can be used for testing.

The implementation here is a wrapper around the `RISE()` function from the **GraphRankTest** package (Zhou and Chen 2025).

4.14. `Rosenbaum()`

The Rosenbaum (2005) cross-match test uses a similar approach as the Friedman-Rafsky test, but is based on the optimal non-bipartite matching instead of the MST as a similarity graph, i.e., a graph where pairs of two observations in the pooled sample are connected such that the sum over the edge lengths (= Euclidean distances of connected observations) is minimized. For a visualization, see the "Getting Started" vignette. In the case of an odd pooled sample size, a ghost observation is added that has the maximal distance to all real observations. This ghost observation and the observation that was matched with it are then discarded in the calculation of the test statistic.

The test statistic of the cross-match test is given by the standardized cross-match count

$$\frac{\text{CMC} - \mathsf{E}_{H_0}(\text{CMC})}{\sqrt{\mathsf{VAR}_{H_0}(\text{CMC})}},$$

where CMC denotes the cross-match count and E_{H_0} and VAR_{H_0} its expectation and variance, respectively, under $H_0 : F_1 = F_2$. The cross-match count is the number of edges connecting points from different datasets. The exact distribution of the test statistic under H_0 is known. For small samples, it can be used for computing an exact p value.

The new function `Rosenbaum()` is a wrapper around the `crossmatchtest()` function from the **crossmatch** package (Heller *et al.* 2024). Again, a distance function can be supplied. By default, this is `stats::dist()`, i.e., the Euclidean distance. The new function then calculates the distance matrix internally. Again, we find this more straightforward from a user perspective than supplying a distance matrix on the pooled sample and a vector specifying the dataset membership of each observation. The output of the function includes the raw edge count, its standard error, and expectation under the null, like for the **crossmatch** implementation. In contrast, only either the exact or the approximate p value is returned. By default (`exact = TRUE`), the exact p value is returned. This is appropriate for samples that are not too large. Note that with a pooled sample size of 340 or more, it is numerically impossible to derive the exact distribution due to the factorials involved in the calculation, and `crossmatchtest()` will return a missing value for the exact p value.

4.15. `SC()`

Song and Chen (2022) propose three new tests for the k-sample problem that use the between-sample edges and the within-sample edges of a similarity graph on the pooled sample. Let R^W denote the vector containing the numbers of within-sample edges for each of the k samples and R^B denote the vector containing the numbers of between-sample edges for all $k(k-1)$ pairs of different samples. Then, the first test statistic is given by

$$\begin{aligned} S &= S^W + S^B, \text{ where} \\ S^W &= \left(R^W - \mathsf{E}_{H_0}(R^W)\right)^\top \mathsf{COV}_{H_0}^{-1}\left(R^W\right)\left(R^W - \mathsf{E}_{H_0}(R^W)\right), \\ S^B &= \left(R^B - \mathsf{E}_{H_0}(R^B)\right)^\top \mathsf{COV}_{H_0}^{-1}\left(R^B\right)\left(R^B - \mathsf{E}_{H_0}(R^B)\right). \end{aligned}$$

The second test statistic is based on the vector R^A of all linearly independent numbers of edges between and within samples, i.e., all numbers of edges between all pairs of samples, including the pairs of a sample with itself, except for the pair of sample $(k-1)$ and sample k. The test statistic is then defined as

$$S^A = \left(R^A - \mathsf{E}_{H_0}(R^A)\right)^\top \mathsf{COV}_{H_0}^{-1}\left(R^A\right)\left(R^A - \mathsf{E}_{H_0}(R^A)\right).$$

All expectations and covariances under the null can be calculated analytically again. While $\mathsf{COV}_{H_0}\left(R^W\right)$ is shown to be always invertible, no such proof exists for $\mathsf{COV}_{H_0}\left(R^B\right)$ and $\mathsf{COV}_{H_0}\left(R^A\right)$. Therefore, Song and Chen (2022) suggest checking the invertibility numerically before applying the test and using a generalized inverse if necessary. This is already done within their implementation. Based on S^A, an asymptotic test can easily be performed. The asymptotic distribution of S is more complicated and hard to compute in practice; therefore,

a fast test is suggested instead. It combines the tests using S^W and S^B and takes the Bonferroni-adjusted p value of both these tests. Alternatively, a permutation test can be performed for either S^A or S.
The implementation here for the test of Song and Chen (2022) is a wrapper around the `gtestsmulti()` function from **gTestsMulti** (Song and Chen 2023b). The input is simplified as for the wrapper around `g.tests()`. The user has to choose whether the original (S) or the fast (S^A) version of the test should be performed. If the number of permutations for the permutation test (`n.perm`) is set to 0, the approximate test is performed; otherwise, the permutation p value is reported

4.16. `Wasserstein()`

The q-Wasserstein distance (Vaserstein 1969) of two distributions F_1 and F_2 on $\mathcal{X}$ is defined as

$$\mathrm{W}(F_1, F_2) := \left(\min_{\pi \in \Pi(F_1, F_2)} \int_{\mathcal{X} \times \mathcal{X}} d_{\mathcal{X}}(x, y)^q \, \mathrm{d}\pi(x, y) \right)^{1/q},$$

where $d_{\mathcal{X}}$ is the metric that $\mathcal{X}$ is provided with, and

$$\Pi(F_1, F_2) := \{\pi_{1,2} \in \mathcal{P}(\mathcal{X} \times \mathcal{X}) | \pi_1 = F_1, \pi_2 = F_2\}$$

is the set of joint distributions over the product space $\mathcal{X} \times \mathcal{X}$ with marginal distributions F_1 and F_2.
In the **Ecume** package (Roux de Bezieux 2024), a permutation test based on the Wasserstein distance is implemented.

4.17. `BG()`

Biau and Gyorfi (2005) test for homogeneity of two (multivariate) datasets by calculating the L_1-distance between the two empirical distributions restricted to a finite partition. For this, a finite partition of the subspace spanned by the two datasets has to be defined. By default, we define a rectangular partition under the assumption of approximately equal cell probabilities. The number of elements of the partition m_n is chosen according to the convergence criteria in Biau and Gyorfi (2005) as $n^{0.8}$, where the exponent can be varied as an argument (`exponent`). For each dimension, $m_n^{1/p} + 1$ equidistant cut-points are created along the range of both datasets to define the partition. It must be ensured that there are at least three cut-points per dimension (min, max, and one point splitting the data into two bins). The argument `eps` ensures that the partition covers all data points by adding some small value to the data range. Alternative partition functions can be provided via the `partition` argument. After calculating the partition, all data points are assigned to an element of the partition along the defined cut-points. Lastly, the L_1 distance between the empirical distribution functions restricted to the elements of the partition is calculated.

4.18. `BG2()`

The statistic of Biswas and Ghosh (2014) uses inter-point distances and is defined as

$$T = \|\hat{\mu}_{D_{F_1}} - \hat{\mu}_{D_{F_2}}\|_2^2, \text{ where}$$

$$\hat{\mu}_{D_{F_1}} = \left[\hat{\mu}_{F_1F_1} = \frac{2}{n_1(n_1-1)} \sum_{i=1}^{n_1} \sum_{j=i+1}^{n_1} \|X_i^{(1)} - X_j^{(1)}\|, \hat{\mu}_{F_1F_2} = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \|X_i^{(1)} - X_j^{(2)}\|\right]^\top,$$

$$\hat{\mu}_{D_{F_2}} = \left[\hat{\mu}_{F_1F_2} = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \|X_i^{(1)} - X_j^{(2)}\|, \hat{\mu}_{F_2F_2} = \frac{2}{n_2(n_2-1)} \sum_{i=1}^{n_2} \sum_{j=i+1}^{n_2} \|X_i^{(2)} - X_j^{(2)}\|\right]^\top.$$

For testing, the scaled statistic

$$\begin{aligned}
T^* &= \frac{N\hat{\lambda}(1-\hat{\lambda})}{2\hat{\sigma}_0^2} T \text{ with} \\
\hat{\lambda} &= \frac{n_1}{N}, \\
\hat{\sigma}_0^2 &= \frac{n_1 S_1 + n_2 S_2}{N}, \text{ where} \\
S_1 &= \frac{1}{\binom{n_1}{3}} \sum_{1 \leqslant i < j < l \leqslant n_1} \|X_i^{(1)} - X_j^{(1)}\| \cdot \|X_i^{(1)} - X_l^{(1)}\| - \hat{\mu}_{F_1F_1}^2 \text{ and} \\
S_2 &= \frac{1}{\binom{n_2}{3}} \sum_{1 \leqslant i < j < l \leqslant n_2} \|X_i^{(2)} - X_j^{(2)}\| \cdot \|X_i^{(2)} - X_l^{(2)}\| - \hat{\mu}_{F_2F_2}^2
\end{aligned}$$

is used as it is asymptotically χ_1^2-distributed. The new function `BG2()` implements the Biswas and Ghosh (2014) test from scratch. `stats::dist()` is used to calculate the Euclidean distance matrix on the pooled sample. The statistic T and the scaled test statistic T^* are implemented according to the formulas above. A permutation test is implemented by permuting the distance matrix, recalculating the test statistic T for the permuted distances, and calculating the p value as the proportion of permuted test statistics larger than the observed test statistic. An asymptotic test is implemented using the asymptotic result from Theorem 4.1 of Biswas and Ghosh (2014), i.e., calculating the p value as `stats::pchisq(`T^*`, lower.tail = FALSE)`.

4.19. `BMG()`

Biswas *et al.* (2014) suggest a graph-based test similar to those of Friedman and Rafsky (1979) and Rosenbaum (2005), but using the shortest Hamiltonian path as the similarity graph. Since calculating the Hamiltonian path is an NP hard problem, the implementation of `BMG()` is based on Kruskal's algorithm, which is a heuristic approach to find the shortest Hamilton Path within the pooled dataset as suggested in Biswas *et al.* (2014). Here, it is implemented as follows:

1. Create an edge list of the fully connected graph on the pooled sample, sorted by increasing Euclidean distance of the corresponding vertices.
2. For each edge, check if (i) an addition of this edge leads to a cyclic graph (using `IsAcyclic()` from the **rlemon** package (Agarwal, Tewari, and Errickson 2023)) and (ii) an addition of this edge leads to a degree larger than two in any (used) vertex. If both criteria are not met, keep the corresponding edge.

3. Return the reduced edge list, containing only edges needed to construct the Hamilton path.

For pooled sample sizes $N < 1030$, an exact test can be performed. For $N \geqslant 1030$, calculation of the exact runs statistic cannot be performed due to terms involved in the calculation becoming too large for representing them as floating point numbers in R. In the exact case, the p values using the null distribution of the univariate runs statistic (Biswas *et al.* 2014) are calculated. If an asymptotic test is performed, the asymptotic null distribution is used instead.

4.20. `BQS()`

Barakat *et al.* (1996) generalize the Schilling-Henze nearest neighbor test (see 4.30) to circumvent choosing the number of nearest neighbors. Their test statistic is the sum of edge counts for all values of K for the K-nearest neighbor graph. The resulting test is equivalent to a sum of Wilcoxon rank sums. It requires samples in the Euclidean space $\mathbb{R}^p$, and it is assumed that there are no ties in ranking with respect to nearness.
Within our implementation, we do not explicitly calculate the K-nearest neighbor graph for all possible values of K as this would be highly inefficient. Instead, the distance matrix on the pooled sample is calculated with a user-specified distance function (Euclidean distance calculated via `stats::dist()` by default), and the column-wise orderings of the distances, excluding the diagonal elements, are calculated. Then, the cumulative numbers of the elements smaller than n_1 are calculated for the first n_1 columns of the orderings, corresponding to the numbers of within-sample edges in the first sample in the K-nearest neighbor graph for $K = 1, \ldots, N-1$. Analogously, the cumulative numbers of the elements greater than n_1 are calculated for the remaining n_2 columns of the orderings, corresponding to the numbers of within-sample edges in the second sample in the K-nearest neighbor graph for $K = 1, \ldots, N-1$. Lastly, all these cumulative numbers are summed up, which corresponds to the Barakat *et al.* (1996) test statistic. A permutation test is implemented using the `boot::boot()` function. For that, the distances are permuted directly, and the calculation is repeated for the permuted distance matrix, which circumvents the costly recalculation of the distances for each permutation.

4.21. `CMDistance()`

The *constrained minimum (CM) distance* (Tatti 2007) uses a *feature function* $S : \mathcal{X} \to \mathbb{R}^m$ that maps points from the sample space $\mathcal{X}$ to a real vector. The *frequency* $\theta \in \mathbb{R}^m$ of S with respect to dataset $X^{(j)}$ is the average of the values of S

$$\theta_j = \frac{1}{N} \sum_{i=1}^{n_j} S(X_i^{(j)}), j = 1, 2.$$

The CM distance is then defined as

$$D_{\mathrm{CM}}(X^{(1)}, X^{(2)}|S)^2 = (\theta_1 - \theta_2)^\top \mathsf{COV}^{-1}(S)(\theta_1 - \theta_2)$$

with

$$\mathsf{COV}(S) = \frac{1}{|\mathcal{X}|} \sum_{\omega \in \mathcal{X}} S(\omega) S(\omega)^\top - \left(\frac{1}{|\mathcal{X}|} \sum_{\omega \in \mathcal{X}} S(\omega) \right) \left(\frac{1}{|\mathcal{X}|} \sum_{\omega \in \mathcal{X}} S(\omega) \right)^\top .$$

It has to be assumed that the feature space $\mathcal{X}$ is finite and can be enumerated. For binary data and S chosen as the conjunction function, i.e., S is one if all components of an observation are one, and zero otherwise, or as the parity function, i.e., S is one if an odd number of components of an observation are one, and zero otherwise, the CM distance reduces to

$$D_{\text{CM}}(X^{(1)}, X^{(2)}|S) = 2\|\theta_1 - \theta_2\|_2.$$

This special case for binary data is implemented first. It includes the option to use either the means as features (example 3 in Tatti (2007)) or the means and covariances (example 4 in Tatti (2007)). Note that there is an error in the calculation of the covariance matrix in A.4 Proof of Lemma 8 in Tatti (2007). The correct covariance matrix has the form $\mathsf{COV}[T_{\mathcal{F}}] = 0.25I$ since $\mathsf{VAR}[T_A] = \mathsf{E}[T_A^2] - \mathsf{E}[T_A]^2 = 0.5 - 0.5^2 = 0.25$ following from the correct statement that $\mathsf{E}[T_A^2] = \mathsf{E}[T_A] = 0.5$. Therefore, formula (4) changes to $d_{CM}(D_1, D_2|S_{\mathcal{F}}) = 2\|\theta_1 - \theta_2\|_2$ and the formula in example 3 changes to $d_{CM}(D_1, D_2|S_1) = 2\|\theta_1 - \theta_2\|_2$. Our implementation is based on these corrected formulas. If the original formula was used, the results on the same data calculated with the formula for the binary special case and the results calculated with the general formula differ by a factor of $\sqrt{2}$. For the general case for categorical data, the user has to specify a feature function S mapping a point in the sample space to a real vector. Additionally, either the covariance matrix $\mathsf{COV}[S]$, if known, or the sample space has to be given. If both are given, the supplied covariance matrix is used and not recalculated. The constrained minimum distance is calculated using Theorem 1 in Tatti (2007), i.e., the formulas given above. Therefore, the supplied or calculated $\mathsf{COV}[S]$, respectively, has to be invertible.

4.22. `DiProPerm()`

Wei *et al.* (2016) propose their *direction-projection-permutation (DiProPerm)* test for which a univariate two-sample statistic is applied to the projection of the datasets onto the normal vector of a separating hyperplane. For this, a linear classification method like a support vector machine (SVM) or distance weighted discrimination (DWD) is used to calculate such a separating hyperplane. A permutation test is then performed for the univariate statistic applied to the projection onto the normal vector. Possible options for the univariate statistic would be the mean difference, the two-sample t-statistic, or the area under the curve (AUC). There is an implementation in the **diproperm** package (Allmon *et al.* 2021), which is currently archived. Our implementation is independent of that implementation. It has the following advantages.

- All suggested univariate two-sample statistics from the paper, i.e., mean difference, t test statistic, and AUC, are implemented. An additional two-sample statistic can be used if a suitable function is supplied via the `stat.fun` argument.
- Additional binary linear classifiers other than the DWD and SVM suggested in the original paper can easily be used by supplying a suitable function via the `dipro.fun` argument.
- The results of the new function are reproducible by setting a random seed.
- The new implementation does not rely on global variables.

- The p value is returned as numeric instead of character.
- The output is an object of class '`htest`' for pretty displaying of the results.

One restriction of the new function is that it no longer supports balanced permutation. That was necessary to ensure the reproducibility, which we consider to be a trade-off worth making since the use of balanced permutation is controversial anyway, see Southworth, Kim, and Owen (2009), and reproducibility is essential for permutation tests.

4.23. `DS()`

The test of Deb and Sen (2021) is a rank version of the Energy statistic. The multivariate ranks are assigned using optimal transport. The implementation is based on R code accompanying the original article (`https://github.com/NabarunD/MultiDistFree`). It wraps up tidied-up versions of the `computestatistic()` and `gensamdist()` given there. The implementation uses the **randtoolbox** package (Christophe and Petr 2024) for random number generation, the **clue** package (Hornik 2005, 2024) to solve the assignment problem for ranking, and the **energy** package (Rizzo and Szekely 2024) for implementation of the Energy statistic.

4.24. `engineerMetric()`

The L_q-*engineer metric* is defined as

$$\mathrm{EN}(X, Y; q) = \left[\sum_{i=1}^{p} \left| \mathsf{E}\left(X_i\right) - \mathsf{E}\left(Y_i\right) \right|^q \right]^{\min(q, 1/q)} \quad \text{with } q > 0,$$

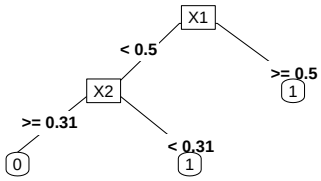
where X_i, Y_i denote the ith component of the p-dimensional random vectors $X \sim F_1$ and $Y \sim F_2$. A new function `engineerMetric()` is implemented. Since the Engineer metric is simply the L_q-distance of the expectations of two random vectors, it is estimated as the L_q-distance of the column means of the datasets. For the distance calculation, the **base** function `norm()` is used, and different options for the L_q norm are available via the `type` argument.

4.25. `GGRL()`, `GGRL_cat()`, `NKT()`

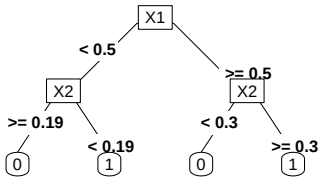
The methods of Ganti *et al.* (1999) and Ntoutsi *et al.* (2008) work by determining the partition induced by a decision tree fit to each dataset and then intersecting these partitions and calculating certain probability estimates on the resulting intersection.

3. Methods applicable to exactly two numeric datasets with target variables

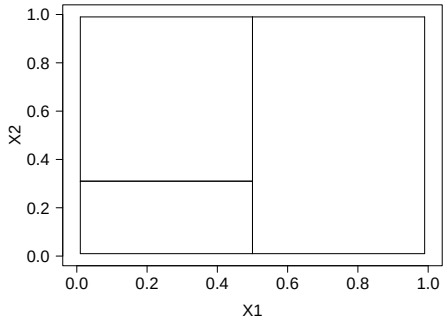
Ntoutsi *et al.* (2008) propose measuring dataset similarity based on probability density estimates derived from decision trees. For this, it is assumed that in addition to both covariate datasets $X^{(1)}$ and $X^{(2)}$, categorical target variables $Y^{(1)}$ and $Y^{(2)}$ with the same levels are given. On each dataset $X^{(j)}$, a classification tree is constructed with $Y^{(j)}$ as the target variable, $j = 1, 2$. The splits defined by the decision trees induce a partition of the feature space $\mathcal{X}$ such that each leaf node corresponds to one segment in the partition. Figure 1 demonstrates the procedure for two example datasets. First, trees are fit to each dataset (Figure 1a and 1b). Then, the sample space is divided into segments based on the splits



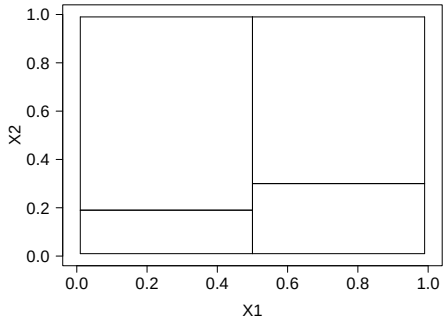
(a) Fitted Tree for Dataset 1.



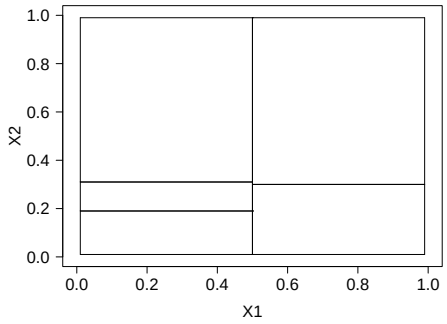
(b) Fitted Tree for Dataset 2.



(c) Partition of sample space derived from fitted tree for Dataset 1.



(d) Partition of sample space derived from fitted tree for Dataset 2.



(e) Intersected partition (greatest common refinement, GCR) from fitted trees for Datasets 1 and 2. Each dataset includes two covariates and a binary target variable.

Figure 1: Partitioning of sample space by fitting trees to two example datasets. Names in the tree nodes refer to variables in the respective datasets. 0 and 1 in the tree leaves refer to the levels of the target variables in the respective datasets.

performed in each tree (Figure 1c and 1d). These partitions are intersected (Figure 1e) and based on the joint partition, the probability densities $P_D(\mathcal{X})$ and $P_D(Y^{(j)}, \mathcal{X})$ are estimated for $D \in \{X^{(1)}, X^{(2)}, Z\}$.

Let n_r denote the number of segments in the joint partition and n_c the number of classes of $Y^{(j)}, j = 1, 2$. $\hat{P}_D(\mathcal{X}) \in \mathbb{R}^{n_r}$ uses the proportion of observations in D that fall into each segment of the joint partition. This means that for each of the n_r segments of the partition,

the number of observations from dataset D that fall into that segment is counted and divided by the total number of observations in D. For the estimation of the joint density $P_D(Y, \mathcal{X})$, the proportion of observations that fall into each segment of the joint partition and belong to each class is determined, $\hat{P}_D(Y, \mathcal{X}) \in \mathbb{R}^{n_r \times n_c}$. Here, for each of the n_r segments of the partition and each of the n_c classes, the number of observations in D where the corresponding target variable has the respective class value and that fall into the respective segment is counted and divided by the total number of observations in D. The conditional density $P_D(Y|\mathcal{X})$ is estimated by calculating the proportion of observations belonging to each class separately for each segment, $\hat{P}_D(Y|\mathcal{X}) \in \mathbb{R}^{n_r \times n_c}$. Here, for each of the n_r segments of the partition and each of the n_c classes, the number of observations in D where the corresponding target variable has the respective class value and that fall into the respective segment is counted and divided by the total number of observations in D that fall into the respective segment.

Then, Ntoutsi *et al.* (2008) consider the similarity index

$$s(p, q) = \sum_i \sqrt{p_i \cdot q_i}$$

for vectors p and q, where $(n_r \times n_c)$-matrices are interpreted as $(n_r \cdot n_c)$-dimensional vectors. For the conditional distribution, the similarity vector $S(Y|\mathcal{X}) \in \mathbb{R}^{n_r}$ is computed with $S(Y|\mathcal{X})_i = s(\hat{P}_{X^{(1)}}(Y|\mathcal{X})_{i\bullet}, \hat{P}_{X^{(2)}}(Y|\mathcal{X})_{i\bullet})$ and index $i\bullet$ denoting the i-th row. Based on this, three similarity measures for datasets are proposed:

1. NTO1 $= s(\hat{P}_{X^{(1)}}(\mathcal{X}), \hat{P}_{X^{(2)}}(\mathcal{X}))$
2. NTO2 $= s(\hat{P}_{X^{(1)}}(Y, \mathcal{X}), \hat{P}_{X^{(2)}}(Y, \mathcal{X}))$
3. NTO3 $= S(Y|\mathcal{X})^\top \hat{P}_Z(\mathcal{X})$.

All three measures have values in the interval $[0, 1]$, where high values correspond to high similarity.

Ganti *et al.* (1999) calculate a decision tree model for each of the two datasets and calculate the greatest common refinement (GCR) induced by these trees. That is the intersection of the partitions of the sample space induced by each tree. A visualization of the computation of the GCR is given in Figure 1. Ganti *et al.* (1999) then compare the distribution of both datasets over this GCR. Let n_r denote the number of segments of the GCR, p_i the proportion of observations of $X^{(1)}$ that map to the i-th segment, and q_i the respective proportion of observations of $X^{(2)}$ mapping to the i-th segment. Then Ganti *et al.* (1999) compare the vectors p and q by a difference function $f : \mathbb{R}^{2n_r} \to \mathbb{R}^{n_r}$ and aggregate the results from that by an aggregate function $g : \mathbb{R}^{n_r} \to \mathbb{R}$ to obtain a measure of distance between the two datasets

$$\text{GAN} = g(f(p, q)).$$

Large values then indicate differences between the datasets. They propose the absolute difference function

$$f_a(p, q)_i = |p_i - q_i|,$$

and the scaled difference function

$$f_s(p, q)_i = \begin{cases} \frac{|p_i - q_i|}{(p_i + q_i)/2}, & \text{if } (p_i + q_i) > 0, i = 1, \ldots, n_r \\ 0, & \text{otherwise} \end{cases}.$$

For the aggregate function, they propose the sum or maximum of the values from the difference function. For using the sum as the aggregate function together with either f_a or f_s, it can be shown that the GCR is optimal in the sense that it gives the lowest value over all common refinements. For using the maximum, this property is not fulfilled. Ganti *et al.* (1999) propose using a bootstrap test procedure for assessing whether or not the two datasets are generated by the same data-generating process.
We use **rpart** package (Therneau and Atkinson 2025) for tree estimation. In the frame of a tree object fit with `rpart()`, the nodes are numbered starting with 1 at the root, following the rule that the left child node gets the ID of the parent times 2, and the right child node gets the ID of the parent times 2 plus 1. This allows us to easily trace back the decision rules from a leaf node to the root using integer division by 2. Moreover, the split rules can be easily accessed using the `labels()` function on the tree object. We iterate over leaves and collect all split rules on each path from the leaf to the root. Suppose no upper or lower limit is specified by any split rule for a certain variable in this way. In that case, we set this limit to the minimum or maximum, respectively, of this variable over both datasets. This ensures that each observation in either of the two datasets falls into some part of the intersected partition later on. The resulting set of ranges for all variables for each leaf node gives us the partition induced by the tree. The resulting partitions are intersected as described in Ganti *et al.* (1999) and Ntoutsi *et al.* (2008). For Ntoutsi *et al.* (2008), all three methods presented in the original article (see above) are implemented. No test is performed. For Ganti *et al.* (1999), the difference and aggregation functions can be supplied by the users. The suggested choices f_a and f_s, i.e., taking the absolute differences between the joint probabilities calculated on the GCR or normalizing this difference with the sum of both probabilities, are readily implemented. The default difference function is set to f_a, and the default aggregation function is set to the sum. A permutation test can be performed.
Neither Ntoutsi *et al.* (2008) nor Ganti *et al.* (1999) discuss the hyperparameter choice for the decision trees. Here, we offer the options to use the default parameter settings of `rpart()` or to tune the hyperparameters. For tuning the hyperparameters, we use the `best.rpart()` function of the **e1071** package (Meyer, Dimitriadou, Hornik, Weingessel, and Leisch 2024). The parameters `minsplit`, `minbucket`, and `cp` of the tree can be tuned. The ranges that are used here for tuning are chosen based on the recommendations by Bischl, Binder, Lang, Pielok, Richter, Coors, Thomas, Ullmann, Becker, Boulesteix, Deng, and Lindauer (2021). Tuning is enabled by default but can be disabled by setting `tune` to `FALSE`. Cross-validation is used for tuning. The number of evaluations (`n.eval`) is set to 100 as a default, and the number of folds (`k`) is set to 5. Both values can be customized by the user. The remaining calculation works the same for a tuned or untuned tree model. By default, the number of permutations is set to 0, corresponding to not performing any test.

An implementation for categorical data for the method of Ganti *et al.* (1999) is also supplied. This comes with the following difficulties. If a category is only observed in one dataset and not in the other, or even if just not all combinations of categories are observed, it might happen that at a certain split, not all levels of the respective variable are observed in the remaining data at that split. Then, it is unclear to which child node the missing level is assigned. In the `rpart::rpart()` implementation that we use here, the label is not assigned at all. If now in the other dataset, the combination with this label is present, the respective data points do not fit anywhere in the intersected partition. Therefore, the calculated probabilities in the joint distribution do not sum to one anymore. In these cases, a warning is printed. The

function might still return a useful measure of dataset distance, but the interpretation and theoretical results might not hold anymore. Also note that for deep trees, the intersection in practice often reduces to all combinations of categories of the variables. Therefore, the measure reduces to the differences in frequency of all category combinations in these cases, but is far more complicated and time-consuming to calculate.

4.26. `Jeffreys()`

Jeffreys divergence (Jeffreys 1997) is the symmetrized version

$$J(F_1, F_2) = \mathrm{KL}(F_1, F_2) + \mathrm{KL}(F_2, F_1)$$

of the Kullback Leibler (KL) divergence (Kullback and Leibler 1951)

$$\mathrm{KL}(F_1, F_2) := \int \log\left(\frac{f_1(x)}{f_2(x)}\right) f_1(x)\,\mathrm{d}x.$$

Within the `Jeffreys()` function, Jeffreys divergence is calculated as the sum of the two KL-divergences (Kullback and Leibler 1951) where each dataset is used as the first once. The KL-divergences are calculated using density ratio estimation as recommended in Sugiyama, Liu, du Plessis, Yamanaka, Yamada, Suzuki, and Kanamori (2013). For this, the `densratio()` function from the **densratio** package (Makiyama 2019) is used. By default, the method `KLIEP` is chosen as suggested by Sugiyama *et al.* (2013). At the time of implementation, the **densityratio** package was not yet available on CRAN, and the resulting KL divergences for some simple examples of data sampled from two univariate normal distributions were worse compared to the true values, which are known for that simple case, than the resulting KL divergences using the alternative package. By now, the two packages perform similarly but the **densityratio** does not show advantages that would make a change of the implementation necessary.

4.27. `LHZ()`

The *characteristic distance* (Li *et al.* 2022) is defined as

$$\begin{aligned}\mathrm{CD}(X, Y) = \mathsf{E}\big[\|\mathsf{E}\left(\exp\left(i\langle X'', X - X'\rangle\right)|X - X'\right) \\ - \mathsf{E}\left(\exp\left(i\langle Y, X - X'\rangle\right)|X - X'\right)\|^2\big] \\ + \mathsf{E}\big[\|\mathsf{E}\left(\exp\left(i\langle X, Y - Y'\rangle\right)|Y - Y'\right) \\ - \mathsf{E}\left(\exp\left(i\langle Y'', Y - Y'\rangle\right)|Y - Y'\right)\|^2\big],\end{aligned}$$

where X', X'' and Y', Y'' denote independent copies of $X \sim F_1$ and $Y \sim F_2$, respectively. An empirical version is obtained by replacing the conditional expectations with empirical means. The implementation calculates the empirical characteristic distance between two datasets. For both summands, Euler's formula is used for every entry of the inner product defined in Li *et al.* (2022). Both mean values are calculated, and the squared complex modulus of the difference between the two means is calculated. Since the inner product leads to a symmetric matrix, only an upper triangular matrix is calculated, and the final sum is multiplied by two. For reproducibility, a permutation test with `n.perm` permutations and random seed `seed` is performed.

4.28. `MMCM()`, `Petrie()`

The tests of Mukherjee *et al.* (2022) and Petrie (2016) generalize the Rosenbaum cross-match test to multiple samples by calculating the cross-counts for all pairs of samples based on the optimal non-bipartite matching on the pooled sample and taking the Mahalanobis distance or simply the sum of the cross-counts, respectively, as the test statistics. The cross-match counts $A = (a_{12}, a_{13}, \ldots, a_{1k}, a_{23}, \ldots, a_{2k}, \ldots, a_{k-1,k})^\top$ for all pairs of datasets are calculated using the optimal non-bipartite matching on the pooled sample. The test statistic for the MMCM test then is the Mahalanobis distance of the observed cross-counts under the null hypothesis $H_0 : F_1 = F_2 = \cdots = F_k$

$$\text{MMCM} = (A - \mathsf{E}_{H_0}(A))^\top \mathsf{COV}_{H_0}^{-1}(A)(A - \mathsf{E}_{H_0}(A)).$$

The expectation and covariance matrix of the cross-count vector A under H_0 can be calculated analytically and depend only on the sample sizes $n_i, i = 1, \ldots, k$. Small values of the multi-sample Mahalanobis cross-match (MMCM) statistic indicate similarity. However, as there is no known upper bound, it is hard to interpret the MMCM value. The MMCM statistic follows a $\chi^2_{\binom{k}{2}}$ distribution asymptotically under the null, which can be used for testing. The statistic for the test or Petrie is simply the sum of the cross-matches. When standardized by its expectation and variance under the null, the test statistic is asymptotically normally distributed. Small values of the test statistic correspond to dissimilarity. New functions `MMCM()` and `Petrie()` were implemented. There exist implementations of these methods in the R package **multicross** (Agarwal *et al.* 2020), but the package is archived on CRAN, and the implementation makes it impossible to access the test statistic and p value as numeric values. Therefore, here, the functions were re-implemented from scratch. To ensure that the new functions are not derivations of the **multicross** versions, they were implemented by an author who had not looked at the **multicross** implementations before. The functions implement the formulas from Section 2 of Mukherjee *et al.* (2022). The new output is again of class '`htest`' and contains the test statistic value and the p value as a numeric value. The **nbpMatching** package (Beck, Lu, and Greevy 2024) is used for calculating the optimal non-bipartite matching. Note that in case of ties in the distance matrix, the optimal non-bipartite matching might not be defined uniquely. In the current implementation, the observations in the pooled sample are ordered by default as supplied by the user. When searching for a match, the **nbpMatching** implementation of the optimal non-bipartite matching algorithm starts at the end of the pooled sample. Therefore, with many ties (e.g., for categorical data), observations from the first dataset are often matched with ones from the last dataset, and so on. This might affect the validity of the test negatively since, even under the null, more cross counts than expected are observed. As a workaround, the current implementation allows for a random ordering of the pooled sample in case of ties. A small simulation study was conducted to examine the impact of the random ordering for the `Petrie()` function. In this study, the number of cross-counts under the null hypothesis was computed both with and without random ordering of the pooled sample across different settings of sample size, data dimension and the data-generating distribution. For low-dimensional data (i.e. $p = 2$) the random ordering appears to resolve this problem for binary and categorical data. Figures 2 and 3 show the simulation results for low-dimensional data generated from binomial and discrete uniform distributions, respectively. Simultaneously, for high-dimensional data (i.e. $p = 10$) without ties or with only very few ties, both versions of the implementation yield

consistent results. Figures 4 and 5 show the results for high-dimensional data generated from binomial and discrete uniform distributions, respectively.

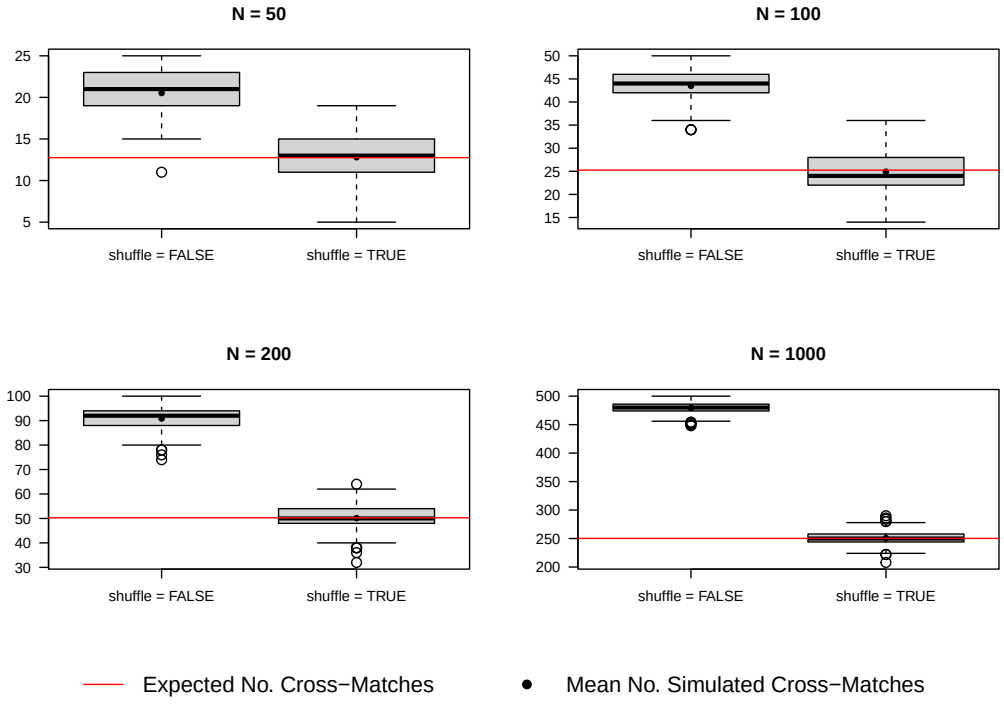


Figure 2: Simulation results for bivariate data generated by a binomial distribution to examine the effect of random ordering within the `Petrie()` function in case of ties.

4.29. `OTDD()`

Alvarez-Melis and Fusi (2020a) define a distance based on optimal transport between datasets that include a target (class) variable Y. The *optimal transport dataset distance* (OTDD) is defined as

$$d_{\mathrm{OT}}(X^{(1)}, X^{(2)}) = \min_{\pi \in \Pi(F_1, F_2)} \int_{\mathcal{Z} \times \mathcal{Z}} d_{\mathcal{Z}}(z, z')^q \, \mathrm{d}\pi(z, z')$$

where $X^{(1)}, X^{(2)}$ denote the two datasets,

$$\Pi(F_1, F_2) := \{\pi_{1,2} \in \mathcal{P}(\mathcal{Z} \times \mathcal{Z}) | \pi_1 = F_1, \pi_2 = F_2\}$$

is the set of joint distributions over the product space $\mathcal{Z} \times \mathcal{Z}$ over the sample space of the pooled sample with marginal distributions F_1 and F_2, and

$$d_{\mathcal{Z}}(z, z') := (d_{\mathcal{X}}(x, x')^{q'} + W_{q'}(\alpha_y, \alpha_{y'})^{q'})^{1/q'}.$$

defines a distance of two points $z^\top = (x^\top, y)$, and $z'^\top = (x'^\top, y')$ in the pooled sample. $d_{\mathcal{X}}$ defines a distance on the covariate space, e.g., the Euclidean distance, and $W_{q'}(\alpha_y, \alpha_{y'})$ is the

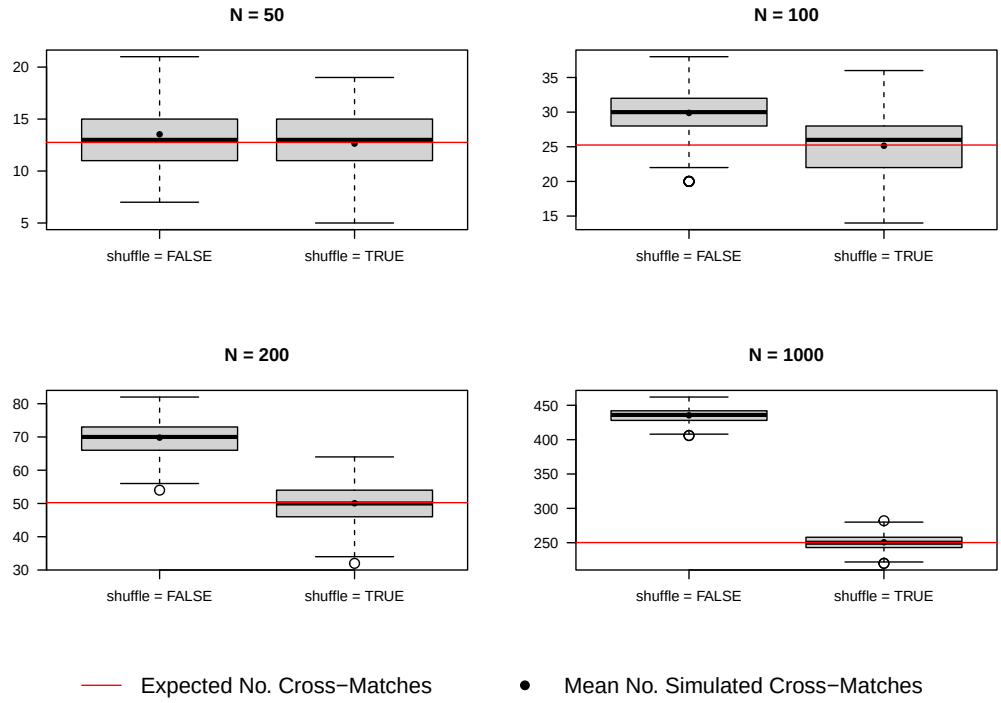


Figure 3: Simulation results for bivariate data generated by a discrete uniform distribution to examine the effect of random ordering within the `Petrie()` function in case of ties.

q'-Wasserstein distance of the distribution of the subset of covariate data with corresponding response value y and the distribution of the subset of covariate data with corresponding response value y'. The powers q and q' have to be chosen in advance to calculate the OTDD. The optimal transport problem can intuitively be motivated by imagining each probability density as a pile of dirt. Then, the cost function corresponds to the cost of transporting the dirt from one point to another, which is proportional to the distance between the two points. The optimal transport then corresponds to the lowest cost required for moving one pile of dirt fully to the shape and location of the other. Therefore, distributions can be regarded as more similar if the optimal transport between them is lower. For an intuitive explanation and visualization of the OTDD, also refer to Alvarez-Melis and Fusi (2020b).

There is a Python implementation of the method (`https://github.com/microsoft/otdd`) that was used as a rough orientation here. Compared to that, the JDOT option is deprecated. The new implementation uses the Wasserstein distance implementation from the **approxOT** package (Dunipace 2024) and the matrix square root from the **expm** package (Maechler, Dutang, and Goulet 2024). Note that the solution of the optimal transport between two distributions is given by their q-Wasserstein distance to the power of q. There are different options for the method to calculate the optimal transport based on the dataset distance. First case: chosen method is `"augmentation"`. In this case, the variable means and the covariance matrix of each dataset, reduced to each target observation value in that dataset, are calculated. The mean vector and the vectorized covariance matrix (column-wise) corresponding to

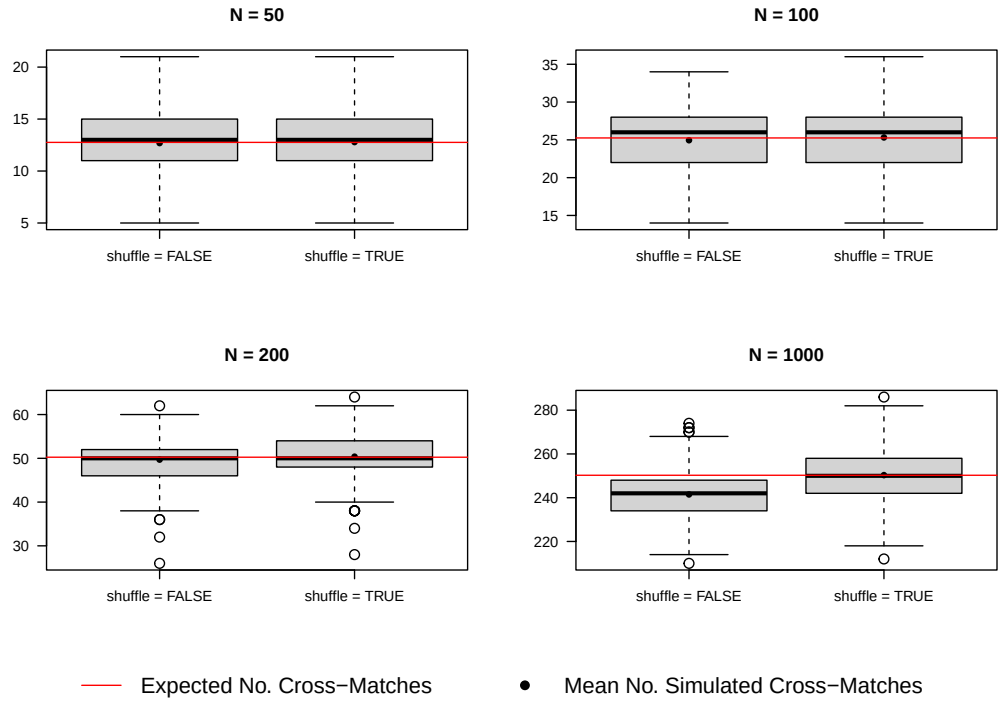


Figure 4: Simulation results for multivariate data ($p = 10$) generated by a binomial distribution to examine the effect of random ordering within the `Petrie()` function in case of ties.

the target value are appended to each observation in each dataset. Then, the q-Wasserstein distance to the power of q of these augmented datasets is calculated. Note that this calculation assumes commuting covariance matrices of all label distributions (rarely fulfilled in practice) and that the feature space metric coincides with the ground cost of the optimal transport problem on the labels (Alvarez-Melis and Fusi 2020a). Second case: chosen method is `"precomputed.labeldist"`. In this case, both the distance matrix for the label distributions and the distance matrix for the features are calculated, and the corresponding distances are added with weights `lambda.x` and `lambda.y`, respectively, to calculate a cost matrix of all observations. In case of `sinkhorn = FALSE`, i.e., for the exact calculation, only the costs from each observation from the first dataset to each observation from the second dataset are needed. When using the debiased Sinkhorn approximation, additionally, the costs within each dataset are needed. For calculating the distance matrices of the label distributions, there are again different options:

1. `inner.ot.method = "exact"`. The Wasserstein distance for each label pair is calculated between the datasets reduced to the observations where the target value equals the corresponding label. There are options for using the (debiased) Sinkhorn approximation and changing the parameters of the Wasserstein distance and the ground cost metric.

2. `inner.ot.method = "gaussian.approx"`. The label distributions are approximated

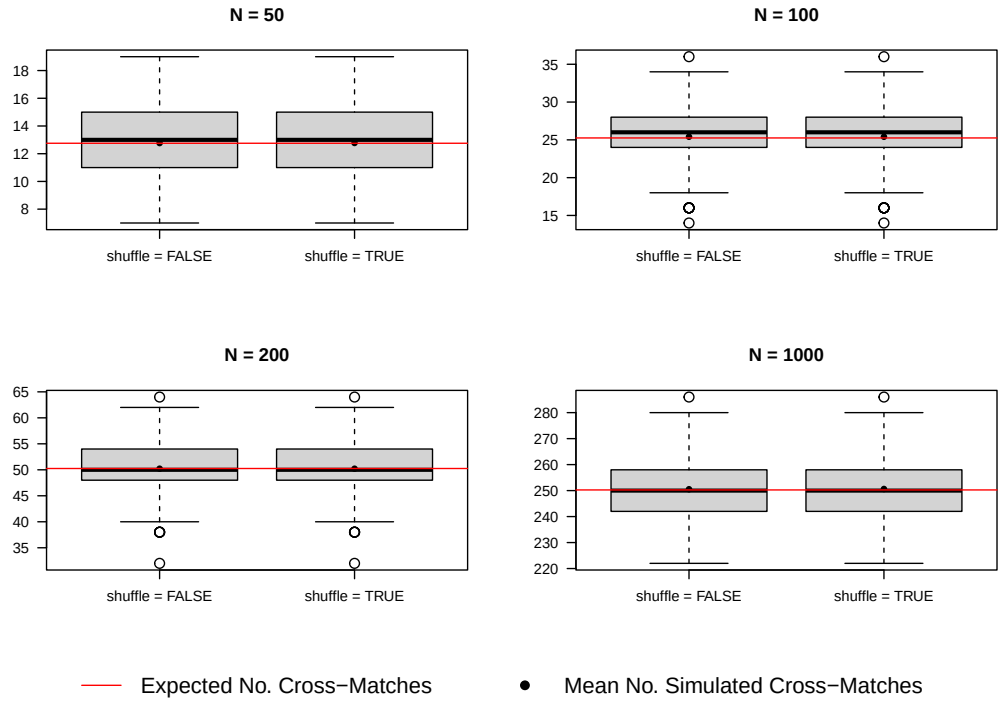


Figure 5: Simulation results for multivariate data ($p = 10$) generated by a discrete uniform distribution to examine the effect of random ordering within the `Petrie()` function in case of ties.

by Gaussians, which leads to a simple closed-form solution of the optimal transport problem that uses only the means and covariances. The calculation includes calculating multiple matrix square roots of covariance matrices, which might get costly if the number of variables is high. Moreover, this calculation fails if the estimated covariance matrix is not numerically positive semi-definite. This might happen especially for $N < p$ settings.

3. `inner.ot.method = "only.means"`. The former is further simplified by using only the means (i.e., assuming equal covariance matrices in all label distributions).

4. `inner.ot.method = "naive.upperbound"`. A distribution-agnostic upper bound for the optimal transport between the label distributions is calculated that again relies only on the means and covariance matrices of these distributions.

4.30. `SH()`

The Schilling-Henze (Schilling 1986; Henze 1988) test uses the mean within-sample edge-count, i.e.,

$$\text{SH} := L := \frac{1}{KN}(R_1 + R_2)$$

in a K-nearest neighbor graph as the test statistic. It is implemented from scratch as follows.

1. Calculate K-nearest neighbor (NN) edge matrix on the pooled sample (distance function returning a distance matrix and K are inputs of the function), i.e., create a matrix where the first column is each observation number repeated K times, and the second column is the corresponding K nearest neighbors of that observation. For the calculation of the K-NN graph, a function can be supplied by the user. Pre-implemented options include a brute-force search, a wrapper for the `kNN()` function from the **dbscan** package (Hahsler, Piekenbrock, and Doran 2019), and the fast (approximative) K-NN algorithm implemented in the `get.knn()` function from the **FNN** package (Beygelzimer, Kakadet, Langford, Arya, Mount, and Li 2024).

2. Count the number L of rows where both observations come from the same sample (i.e., either both have observation number $\leqslant n_1$ or both have observation number $> n_1$).

3. Calculate the quantities $\mathsf{E}_{H_0}(L)$ and $\mathsf{VAR}_{H_0}(L)$ from proposition 2.1 in Henze (1988).

4. Calculate the standardized test statistic $L^* = (L - \mathsf{E}_{H_0}(L))/\sqrt{\mathsf{VAR}_{H_0}(L)}$.

5. When performing a permutation test, permute the distance matrix on the pooled sample, recalculate L, and calculate the proportion of permuted test statistics that are larger than the observed value of L.

6. When performing an asymptotic test, use the asymptotic normal distribution of Z as proposed in Remark 5.1 of Henze (1988).

7. The observed value of L^* is returned in the result as the `statistic`, the observed L is returned as the `estimate`.

The default for K is set to one. This is rather arbitrary based on computational speed as there is no good rule for choosing K so far proposed in the literature (Aslan and Zech 2005).

4.31. `YMRZL()`

Yu *et al.* (2007) propose a permutation test that uses the classification error of a classification tree that distinguishes between the two datasets. The implementation of the test is based on the `C2ST()` function, as the methods work very similarly. Here, we set the classifier to `"rpart"`, i.e., a CART. Instead of the classification accuracy, as for the C2ST, the classification error, i.e., $1-$ accuracy, is returned. A permutation test is implemented using the `boot::boot()` framework, and the permutation p value is calculated as the proportion of the number $+ 1$ of permuted test statistics smaller than or equal to the observed value divided by the number of permutations. Yu *et al.* (2007) do not propose any asymptotic test, but since their test fits into the framework of Lopez-Paz and Oquab (2017), the binomial test proposed there and implemented in the `Ecume::classifier_test()` function utilized by `C2ST()` is still valid and therefore kept in the implementation.

Acknowledgments

This work has been supported (in part) by the Research Training Group "Biostatistical Methods for High-Dimensional Data in Toxicology" (RTG 2624, Project P1) funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation - Project Number

427806116).
We would like to thank Nabarun Deb and Bodhisattva Sen for allowing us to use their R implementation of their test for the package. Moreover, we would like to thank David Alvarez-Melis, whose Python implementation of the OTDD was the basis for our R implementation.

References

Agarwal A, Tewari A, Errickson J (2023). *rlemon: R Access to LEMON Graph Algorithms*. R package version 0.2.1, URL https://CRAN.R-project.org/package=rlemon.

Agarwal SMD, Bhattacharya B, Zhang NR (2020). *multicross: A Graph-Based Test for Comparing Multivariate Distributions in the Multi Sample Framework*. R package version 2.1.0, URL https://CRAN.R-project.org/package=multicross.

Allmon AG, Marron J, Hudgens MG (2021). *diproperm: Conduct Direction-Projection-Permutation Tests and Display Plots*. R package version 0.2.0, URL https://CRAN.R-project.org/package=diproperm.

Alvarez-Melis D, Fusi N (2020a). "Geometric Dataset Distances via Optimal Transport." In *Advances in Neural Information Processing Systems*, volume 33, pp. 21428–21439. Curran Associates, Inc.

Alvarez-Melis D, Fusi N (2020b). "Measuring dataset similarity using optimal transport." URL https://www.microsoft.com/en-us/research/blog/measuring-dataset-similarity-using-optimal-transport/.

Aslan B, Zech G (2005). "New Test for the Multivariate Two-Sample Problem Based on the Concept of Minimum Energy." *Journal of Statistical Computation and Simulation*, **75**(2), 109–119. ISSN 0094-9655. doi:10.1080/00949650410001661440.

Bahr R (1996). *Ein neuer Test für das mehrdimensionale Zwei-Stichproben-Problem bei allgemeiner Alternative*. Ph.D. thesis, Universität Hannover.

Barakat AS, Quade D, Salama IA (1996). "Multivariate Homogeneity Testing Using an Extended Concept of Nearest Neighbors." *Biometrical Journal*, **38**(5), 605–612. ISSN 1521-4036. doi:10.1002/bimj.4710380509.

Baringhaus L, Franz C (2004). "On a New Multivariate Two-Sample Test." *Journal of Multivariate Analysis*, **88**(1), 190–206. ISSN 0047-259X. doi:10.1016/S0047-259X(03)00079-4.

Baringhaus L, Franz C (2010). "Rigid Motion Invariant Two-Sample Tests." *Statistica Sinica*, **20**(4), 1333–1361. ISSN 1017-0405.

Beck C, Lu B, Greevy R (2024). *nbpMatching: Functions for Optimal Non-Bipartite Matching*. R package version 1.5.6, URL https://CRAN.R-project.org/package=nbpMatching.

Beygelzimer A, Kakadet S, Langford J, Arya S, Mount D, Li S (2024). *FNN: Fast Nearest Neighbor Search Algorithms and Applications*. R package version 1.1.4, URL https://CRAN.R-project.org/package=FNN.

Biau G, Gyorfi L (2005). "On the Asymptotic Properties of a Nonparametric L_1-Test Statistic of Homogeneity." *IEEE Transactions on Information Theory*, **51**(11), 3965–3973. ISSN 1557-9654. doi:10.1109/TIT.2005.856979.

Bischl B, Binder M, Lang M, Pielok T, Richter J, Coors S, Thomas J, Ullmann T, Becker M, Boulesteix AL, Deng D, Lindauer M (2021). "Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges." *arXiv:2107.05847 [cs, stat]*.

Biswas M, Ghosh AK (2014). "A Nonparametric Two-Sample Test Applicable to High Dimensional Data." *Journal of Multivariate Analysis*, **123**, 160–171. ISSN 0047-259X. doi:10.1016/j.jmva.2013.09.004.

Biswas M, Mukhopadhyay M, Ghosh AK (2014). "A Distribution-Free Two-Sample Run Test Applicable to High-Dimensional Data." *Biometrika*, **101**(4), 913–926. ISSN 0006-3444. doi:10.1093/biomet/asu045.

Chen H, Chen X, Su Y (2018). "A Weighted Edge-Count Two-Sample Test for Multivariate and Object Data." *Journal of the American Statistical Association*, **113**(523), 1146–1155. ISSN 0162-1459. doi:10.1080/01621459.2017.1307757.

Chen H, Friedman JH (2017). "A New Graph-Based Two-Sample Test for Multivariate and Object Data." *Journal of the American Statistical Association*, **112**(517), 397–409. ISSN 0162-1459. doi:10.1080/01621459.2016.1147356.

Chen H, Zhang J (2017). *gTests: Graph-Based Two-Sample Tests*. R package version 0.2, URL https://CRAN.R-project.org/package=gTests.

Chen L, Dou WW, Qiao Z (2013). "Ensemble Subsampling for Imbalanced Multivariate Two-Sample Tests." *Journal of the American Statistical Association*, **108**(504), 1308–1323. ISSN 0162-1459. doi:10.1080/01621459.2013.800763.

Christophe D, Petr S (2024). *randtoolbox: Generating and Testing Random Numbers*. R package version 2.0.5.

Chu L, Chen H (2019). "Asymptotic Distribution-Free Change-Point Detection for Multivariate and Non-Euclidean Data." *The Annals of Statistics*, **47**(1), 382–414. ISSN 0090-5364, 2168-8966. doi:10.1214/18-AOS1691.

Chwialkowski KP, Ramdas A, Sejdinovic D, Gretton A (2015). "Fast Two-Sample Testing with Analytic Representations of Probability Measures." In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Deb N, Sen B (2021). "Multivariate Rank-Based Distribution-Free Nonparametric Testing Using Measure Transportation." *Journal of the American Statistical Association*, **118**(541), 1–16. ISSN 0162-1459. doi:10.1080/01621459.2021.1923508.

Dunipace EA (2024). *approxOT: approximate optimal transport*. R package version 1.1, URL https://github.com/ericdunipace/approxOT.

Franz C (2024). *cramer: Multivariate Nonparametric Cramer-Test for the Two-Sample-Problem*. R package version 0.9-4, URL https://CRAN.R-project.org/package=cramer.

Friedman JH, Rafsky LC (1979). "Multivariate Generalizations of the Wald-Wolfowitz and Smirnov Two-Sample Tests." *The Annals of Statistics*, **7**(4), 697–717. ISSN 0090-5364.

Fukumizu K, Bach FR, Jordan MI (2004). "Dimensionality Reduction for Supervised Learning with Reproducing Kernel Hilbert Spaces." *Journal of Machine Learning Research*, **5**, 73–99.

Ganti V, Gehrke J, Ramakrishnan R, Loh WY (1999). "A Framework for Measuring Changes in Data Characteristics." In *Proceedings of the 18th Symposium on Principles of Database Systems*, pp. 126–137.

Gretton A, Borgwardt K, Rasch M, Schölkopf B, Smola A (2006). "A Kernel Method for the Two-Sample-Problem." In *Advances in Neural Information Processing Systems*, volume 19. MIT Press.

Hahsler M, Piekenbrock M, Doran D (2019). "dbscan: Fast Density-Based Clustering with R." *Journal of Statistical Software*, **91**(1), 1–30. doi:10.18637/jss.v091.i01.

Hediger S, Michel L, Naef J (2024). *hypoRF: Random Forest Two-Sample Tests*. R package version 1.0.1, URL https://CRAN.R-project.org/package=hypoRF.

Hediger S, Michel L, Näf J (2022). "On the Use of Random Forest for Two-Sample Testing." *Computational Statistics & Data Analysis*, **170**, 107435. ISSN 0167-9473. doi:10.1016/j.csda.2022.107435. URL https://www.sciencedirect.com/science/article/pii/S0167947322000159.

Heller R, Small D, Rosenbaum P (2024). *crossmatch: The Cross-Match Test*. R package version 1.4-0, URL https://CRAN.R-project.org/package=crossmatch.

Henze N (1988). "A Multivariate Two-Sample Test Based on the Number of Nearest Neighbor Type Coincidences." *The Annals of Statistics*, **16**(2), 772–783. ISSN 0090-5364.

Hornik K (2005). "A CLUE for CLUster Ensembles." *Journal of Statistical Software*, **14**(12). doi:10.18637/jss.v014.i12.

Hornik K (2024). *clue: Cluster Ensembles*. R package version 0.3-66, URL https://CRAN.R-project.org/package=clue.

Hothorn T, Hornik K, van de Wiel MA, Zeileis A (2008). "Implementing a class of permutation tests: The coin package." *Journal of Statistical Software*, **28**(8), 1–23. doi:10.18637/jss.v028.i08.

Huang Z (2022). *KMD: Kernel Measure of Multi-Sample Dissimilarity*. R package version 0.1.0, URL https://CRAN.R-project.org/package=KMD.

Huang Z, Sen B (2024). "A Kernel Measure of Dissimilarity between M Distributions." *Journal of the American Statistical Association*, **119**(548), 3020–3032. doi:10.1080/01621459.2023.2298036.

Jeffreys H (1997). "An Invariant Form for the Prior Probability in Estimation Problems." *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, **186**(1007), 453–461. doi:10.1098/rspa.1946.0056.

Jitkrittum W, Szabó Z, Chwialkowski KP, Gretton A (2016). "Interpretable Distribution Features with Maximum Testing Power." In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). "kernlab – An S4 Package for Kernel Methods in R." *Journal of Statistical Software*, **11**(9), 1–20. doi:10.18637/jss.v011.i09.

Kirchler M, Khorasani S, Kloft M, Lippert C (2020). "Two-sample Testing Using Deep Learning." In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 1387–1398. PMLR. ISSN 2640-3498.

Kullback S, Leibler RA (1951). "On Information and Sufficiency." *The Annals of Mathematical Statistics*, **22**(1), 79–86. ISSN 0003-4851, 2168-8990. doi:10.1214/aoms/1177729694.

Li X, Hu W, Zhang B (2022). "Measuring and Testing Homogeneity of Distributions by Characteristic Distance." *Statistical Papers*. ISSN 1613-9798. doi:10.1007/s00362-022-01327-7.

Lopez-Paz D, Oquab M (2017). "Revisiting Classifier Two-Sample Tests." In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=SJkXfE5xx.

Maechler M, Dutang C, Goulet V (2024). *expm: Matrix Exponential, Log, 'etc'*. R package version 1.0-0, URL https://CRAN.R-project.org/package=expm.

Makiyama K (2019). *densratio: Density Ratio Estimation*. R package version 0.2.1, URL https://CRAN.R-project.org/package=densratio.

Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2024). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.7-16, URL https://CRAN.R-project.org/package=e1071.

Muandet K, Fukumizu K, Sriperumbudur B, Schölkopf B (2017). "Kernel Mean Embedding of Distributions: A Review and Beyond." *Foundations and Trends® in Machine Learning*, **10**(1-2), 1–141. ISSN 1935-8237, 1935-8245. doi:10.1561/2200000060.

Mukherjee S, Agarwal D, Zhang NR, Bhattacharya BB (2022). "Distribution-Free Multisample Tests Based on Optimal Matchings With Applications to Single Cell Genomics." *Journal of the American Statistical Association*, **117**(538), 627–638. ISSN 0162-1459. doi:10.1080/01621459.2020.1791131.

Mukhopadhyay S, Wang K (2020a). *LPKsample: LP Nonparametric High Dimensional K-Sample Comparison*. R package version 2.1, URL https://CRAN.R-project.org/package=LPKsample.

Mukhopadhyay S, Wang K (2020b). "A Nonparametric Approach to High-Dimensional k-Sample Comparison Problems." *Biometrika*, **107**(3), 555–572. ISSN 0006-3444. doi:10.1093/biomet/asaa015.

Ntoutsi I, Kalousis A, Theodoridis Y (2008). "A General Framework for Estimating Similarity of Datasets and Decision Trees: Exploring Semantic Similarity of Decision Trees." In

Proceedings of the 2008 SIAM International Conference on Data Mining (SDM), pp. 810–821. Society for Industrial and Applied Mathematics. ISBN 978-0-89871-654-2. doi:10.1137/1.9781611972788.73.

Pan W, Tian Y, Wang X, Zhang H (2018). "Ball Divergence: Nonparametric Two Sample Test." *The Annals of Statistics*, **46**(3), 1109–1137. ISSN 0090-5364. doi:10.1214/17-AOS1579.

Paul B, De SK, Ghosh AK (2022a). *HDLSSkST: Distribution-Free Exact High Dimensional Low Sample Size k-Sample Tests*. R package version 2.1.0, URL https://CRAN.R-project.org/package=HDLSSkST.

Paul B, De SK, Ghosh AK (2022b). "Some Clustering-Based Exact Distribution-Free k-Sample Tests Applicable to High Dimension, Low Sample Size Data." *Journal of Multivariate Analysis*, **190**, 104897. ISSN 0047-259X. doi:10.1016/j.jmva.2021.104897. URL https://www.sciencedirect.com/science/article/pii/S0047259X21001743.

Petrie A (2016). "Graph-Theoretic Multisample Tests of Equality in Distribution for High Dimensional Data." *Computational Statistics & Data Analysis*, **96**, 145–158. ISSN 0167-9473. doi:10.1016/j.csda.2015.11.003.

Rahmatallah Y, Zybailov B, Emmert-Streib F, Glazko G (2017). "GSAR: Bioconductor package for gene set analysis in R." *BMC Bioinformatics*, **18**, 61.

Rizzo M, Szekely G (2024). *energy: E-Statistics: Multivariate Inference via the Energy of Data*. R package version 1.7-12, URL https://CRAN.R-project.org/package=energy.

Rizzo ML, Székely GJ (2010). "DISCO Analysis: A Nonparametric Extension of Analysis of Variance." *The Annals of Applied Statistics*, **4**(2), 1034–1055. ISSN 1932-6157, 1941-7330. doi:10.1214/09-AOAS245.

Rosenbaum PR (2005). "An Exact Distribution-Free Test Comparing Two Multivariate Distributions Based on Adjacency." *Journal of the Royal Statistical Society B*, **67**(4), 515–530. ISSN 1369-7412.

Roux de Bezieux H (2024). *Ecume: Equality of 2 (or k) Continuous Univariate and Multivariate Distributions*. R package version 0.9.2, URL https://CRAN.R-project.org/package=Ecume.

Sarkar S, Ghosh AK (2020). "On Perfect Clustering of High Dimension, Low Sample Size Data." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **42**(9), 2257–2272. ISSN 1939-3539. doi:10.1109/TPAMI.2019.2912599. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, URL https://ieeexplore.ieee.org/document/8695805.

Schilling MF (1986). "Multivariate Two-Sample Tests Based on Nearest Neighbors." *Journal of the American Statistical Association*, **81**(395), 799–806. ISSN 0162-1459. doi:10.2307/2289012.

Song H, Chen H (2022). "New Graph-Based Multi-Sample Tests for High-Dimensional and Non-Euclidean Data." doi:10.48550/arXiv.2205.13787. ArXiv:2205.13787 [stat].

Song H, Chen H (2023a). "Generalized Kernel Two-Sample Tests." *Biometrika*, pp. 755–770. ISSN 1464-3510. doi:10.1093/biomet/asad068.

Song H, Chen H (2023b). *gTestsMulti: New Graph-Based Multi-Sample Tests*. R package version 0.1.1, URL https://CRAN.R-project.org/package=gTestsMulti.

Song H, Chen H (2023c). *kerTests: Generalized Kernel Two-Sample Tests*. R package version 0.1.4, URL https://CRAN.R-project.org/package=kerTests.

Southworth LK, Kim SK, Owen AB (2009). "Properties of Balanced Permutations." *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, **16**(4), 625–638. ISSN 1557-8666. doi:10.1089/cmb.2008.0144.

Sriperumbudur B, Fukumizu K, Gretton A, Lanckriet G, Schölkopf B (2009). "Kernel Choice and Classifiability for RKHS Embeddings of Probability Distributions." In *Advances in Neural Information Processing Systems 22*, pp. 1750–1758. Max-Planck-Gesellschaft, Curran, Red Hook, NY, USA.

Sriperumbudur BK, Gretton A, Fukumizu K, Lanckriet G, Schölkopf B (2008). "Injective Hilbert space embeddings of probability measures." In *21st Annual Conference on Learning Theory (COLT 2008)*, pp. 111–122. Omnipress.

Sriperumbudur BK, Gretton A, Fukumizu K, Schölkopf B, Lanckriet GRG (2010). "Hilbert Space Embeddings and Metrics on Probability Measures." *Journal of Machine Learning Research*, **11**(50), 1517–1561. ISSN 1533-7928.

Stolte M, Kappenberg F, Rahnenführer J, Bommert A (2024). "Methods for Quantifying Dataset Similarity: A Review, Taxonomy and Comparison." *Statistics Surveys*, **18**, 163–298. ISSN 1935-7516. doi:10.1214/24-SS149.

Stolte M, Rahnenführer J, Bommert A (2026a). "An Empirical Comparison of Methods for Quantifying the Similarity of Categorical Datasets." doi:10.48550/arXiv.2604.11458. URL https://arxiv.org/abs/2604.11458.

Stolte M, Rahnenführer J, Bommert A (2026b). "An Empirical Comparison of Methods for Quantifying the Similarity of Numeric Datasets." doi:10.48550/arXiv.2604.12327. URL https://arxiv.org/abs/2604.12327.

Sugiyama M, Liu S, du Plessis MC, Yamanaka M, Yamada M, Suzuki T, Kanamori T (2013). "Direct Divergence Approximation between Probability Distributions and Its Applications in Machine Learning." *Journal of Computing Science and Engineering*, **7**(2), 99–111. ISSN 1976-4677. doi:10.5626/JCSE.2013.7.2.99.

Szabo A, Boucher K, Carroll WL, Klebanov LB, Tsodikov AD, Yakovlev AY (2002). "Variable selection and pattern recognition with gene expression data generated by the microarray technology." *Mathematical Biosciences*, **176**(1), 71–98. ISSN 0025-5564. doi:10.1016/s0025-5564(01)00103-1.

Székely GJ, Rizzo ML (2017). "The Energy of Data." *Annual Review of Statistics and Its Application*, **4**(1), 447–479. doi:10.1146/annurev-statistics-060116-054026.

Tatti N (2007). "Distances between Data Sets Based on Summary Statistics." *Journal of Machine Learning Research*, **8**(1).

Therneau T, Atkinson B (2025). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1.24, URL https://CRAN.R-project.org/package=rpart.

Vaserstein LN (1969). "Markov Processes Over Denumerable Products of Spaces, Describing Large Systems of Automata." *Problemy Peredachi Informatsii*, **5**(3), 64–72.

Wei S, Lee C, Wichers L, Marron JS (2016). "Direction-Projection-Permutation for High-Dimensional Hypothesis Tests." *Journal of Computational and Graphical Statistics*, **25**(2), 549–569. ISSN 1061-8600. doi:10.1080/10618600.2015.1027773.

Weiss L (1960). "Two-Sample Tests for Multivariate Distributions." *The Annals of Mathematical Statistics*, **31**(1), 159–164. ISSN 0003-4851, 2168-8990.

Yu K, Martin R, Rothman N, Zheng T, Lan Q (2007). "Two-sample Comparison Based on Prediction Error, with Applications to Candidate Gene Association Studies." *Annals of Human Genetics*, **71**(1), 107–118. ISSN 1469-1809. doi:10.1111/j.1469-1809.2006.00306.x.

Zaremba W (2022). "B - test." URL https://github.com/wojzaremba/btest.

Zhang J, Chen H (2022). "Graph-Based Two-Sample Tests for Data with Repeated Observations." *Statistica Sinica*, **32**(1), 391–415. ISSN 1017-0405. Publisher: Institute of Statistical Science, Academia Sinica, URL https://www.jstor.org/stable/27108529.

Zhou D, Chen H (2023). "A new ranking scheme for modern data and its application to two-sample hypothesis testing." In G Neu, L Rosasco (eds.), *Proceedings of Thirty Sixth Conference on Learning Theory*, volume 195 of *Proceedings of Machine Learning Research*, pp. 3615–3668. PMLR. URL https://proceedings.mlr.press/v195/zhou23a.html.

Zhou D, Chen H (2025). *GraphRankTest: Rank in Similarity Graph Edge-Count Two-Sample Test (RISE)*. doi:10.32614/CRAN.package.GraphRankTest. R package version 0.1, URL https://CRAN.R-project.org/package=GraphRankTest.

Zhu J, Pan W, Zheng W, Wang X (2021). "Ball: An R Package for Detecting Distribution Difference and Association in Metric Spaces." *Journal of Statistical Software*, **97**(6), 1–31. doi:10.18637/jss.v097.i06.

Affiliation:

Marieke Stolte, Luca Sauer, Jörg Rahnenführer, Andrea Bommert
Department of Statistics
TU Dortmund University
Vogelpothsweg 87
44227 Dortmund, Germany
and
Marieke Stolte

Institute for Medical Information Processing, Biometry, and Epidemiology (IBE)
LMU Medizin
Ludwig-Maximilians-Universität München
Marchioninistrasse 15
81377 Munich, Germany
E-mail: marieke.stolte@ibe.med.uni-muenchen.de