Package: admix (via r-universe)

October 25, 2024

Title Package Admix for Admixture (aka Contamination) Models

Version 2.3.1

Description Implements techniques to estimate the unknown quantities related to two-component admixture models, where the two components can belong to any distribution (note that in the case of multinomial mixtures, the two components must belong to the same family). Estimation methods depend on the assumptions made on the unknown component density; see Bordes and Vandekerkhove (2010) <doi:10.3103/S1066530710010023>, Patra and Sen (2016) <doi:10.1111/rssb.12148>, and Milhaud, Pommeret, Salhi, Vandekerkhove (2024) <doi:10.3150/23-BEJ1593>. In practice, one can estimate both the mixture weight and the unknown component density in a wide variety of frameworks. On top of that, hypothesis tests can be performed in one and two-sample contexts to test the unknown component density (see Milhaud, Pommeret, Salhi and Vandekerkhove (2022) <doi:10.1016/j.jspi.2021.05.010>, and Milhaud, Pommeret, Salhi, Vandekerkhove (2024) <doi:10.3150/23-BEJ1593>). Finally, clustering of unknown mixture components is also feasible in a K-sample setting (see Milhaud, Pommeret, Salhi, Vandekerkhove (2024) <https://jmlr.org/papers/v25/23-0914.html>).

License GPL (>= 3)

URL https://github.com/XavierMilhaud/admix-Rpackage

BugReports https://github.com/XavierMilhaud/admix-Rpackage/issues

Encoding UTF-8

RoxygenNote 7.3.2

Imports base, cubature, EnvStats, fdrtool, graphics, Iso, MASS, orthopolynom, pracma, Rcpp, Rdpack, stats, utils

Suggests doParallel, doRNG, evd, flexsurv, foreach, gridExtra, knitr, lattice, logitnorm, plyr, reshape2, rmarkdown, rmutil, testthat (>= 3.0.0)

Depends R (>= 2.10)

LazyData true

LinkingTo Rcpp

RdMacros Rdpack

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation yes

Author Xavier Milhaud [aut, cre], Pierre Vandekerkhove [ctb], Denys Pommeret [ctb], Yahia Salhi [ctb]

Maintainer Xavier Milhaud <xavier.milhaud.research@gmail.com>

Repository CRAN

Date/Publication 2024-10-24 10:50:02 UTC

Contents

`admix_cluster` *Clustering of K populations following admixture models*

Description

Create clusters on the unknown components related to the K populations following admixture models. Based on the K-sample test using Inversion - Best Matching (IBM) approach, see 'Details' below for further information.

Usage

```
admix_cluster(
  samples,
  admixMod,
  conf_level = 0.95,
  n_sim_tab = 100,
  tune_penalty = FALSE,
  tabul_dist = NULL,
  echo = TRUE,
  parallel = FALSE,
  n_cpu = 2
)
```

Arguments

`samples`	(list) A list of the K (K>0) samples to be studied, all following admixture distributions.
`admixMod`	(list) A list of objects of class 'admix_model', containing useful information about distributions and parameters.
`conf_level`	The confidence level of the K-sample test used in the clustering procedure.

n_sim_tab	Number of simulated gaussian processes used in the tabulation of the inner convergence distribution in the IBM approach.
tune_penalty	A boolean that allows to choose between a classical penalty term or an optimized penalty embedding some tuning parameters (automatically optimized) for k-sample tests used within the clustering procedure. Optimized penalty is particularly useful for low sample size.
tabul_dist	(Only useful for comparisons of detected clusters at different confidence levels) A list of the tabulated distributions of the stochastic integral for each cluster previously detected.
echo	(default to TRUE) Display the remaining computation time.
parallel	(default to FALSE) Boolean to indicate whether parallel computations are performed (speed-up the tabulation).
n_cpu	(default to 2) Number of cores used when parallelizing.

Value

An object of class 'admix_cluster', containing 12 attributes: 1) the number of samples under study; 2) the sizes of samples; 3) the information about mixture components in each sample (distributions and parameters); 4) the number of detected clusters; 5) the list of p-values for each k-sample test at the origin of detected clusters; 6) the cluster affiliation for each sample; 7) the confidence level of statistical tests; 8) which samples in which cluster; 9) the size of clusters; 10) the estimated weights of the unknown component distributions inside each cluster (remind that estimated weights are consistent only if unknown components are tested to be identical, which is the case inside clusters); 11) the matrix of pairwise discrepancies across all samples; 12) the list of tabulated distributions used for statistical tests involved in building the clusters.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2024). "Contamination-source based K-sample clustering." *Journal of Machine Learning Research*, **25**(287), 1–32. https://jmlr.org/papers/v25/23-0914.html.

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 2600, weight = 0.8,
                      comp.dist = list("gamma", "exp"),
                      comp.param = list(list("shape" = 16, "scale" = 1/4),
                                        list("rate" = 1/3.5)))
mixt2 <- twoComp_mixt(n = 3000, weight = 0.7,
                      comp.dist = list("gamma", "exp"),
                      comp.param = list(list("shape" = 14, "scale" = 1/2),
                                        list("rate" = 1/5)))
mixt3 <- twoComp_mixt(n = 3500, weight = 0.6,
                      comp.dist = list("gamma", "gamma"),
```

```
                      comp.param = list(list("shape" = 16, "scale" = 1/4),
                                        list("shape" = 12, "scale" = 1/2)))
mixt4 <- twoComp_mixt(n = 4800, weight = 0.5,
                      comp.dist = list("gamma", "exp"),
                      comp.param = list(list("shape" = 14, "scale" = 1/2),
                                        list("rate" = 1/7)))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
data3 <- getmixtData(mixt3)
data4 <- getmixtData(mixt4)

## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
admixMod3 <- admix_model(knownComp_dist = mixt3$comp.dist[[2]],
                         knownComp_param = mixt3$comp.param[[2]])
admixMod4 <- admix_model(knownComp_dist = mixt4$comp.dist[[2]],
                         knownComp_param = mixt4$comp.param[[2]])

admix_cluster(samples = list(data1, data2, data3, data4),
              admixMod = list(admixMod1, admixMod2, admixMod3, admixMod4),
              conf_level = 0.95, n_sim_tab = 30, tune_penalty = TRUE,
              tabul_dist = NULL, echo = FALSE, parallel = FALSE, n_cpu = 2)
```

`admix_estim` *Estimate the unknown parameters of the admixture model(s)*

Description

Estimate the component weights, the location shift parameter (in case of a symmetric unknown component density), and the unknown component distribution using different estimation techniques. We remind that the i-th admixture model has probability density function (pdf) l_i such that: l_i = p_i * f_i + (1-p_i) * g_i, where g_i is the known component density. The unknown quantities p_i and f_i then have to be estimated.

Usage

```
admix_estim(
  samples,
  admixMod,
  est.method = c("BVdk", "PS", "IBM"),
  sym.f = FALSE
)
```

Arguments

samples	(list) A list of the K (K>0) samples to be studied, all following admixture distributions.
admixMod	(list) A list of objects of class 'admix_model', containing useful information about distributions and parameters.
est.method	The estimation method to be applied. Can be one of 'BVdk' (Bordes and Vandekerkhove estimator), 'PS' (Patra and Sen estimator), or 'IBM' (Inversion Best-Matching approach). The same estimation method is performed on each sample. Important note: estimation by 'IBM' is unbiased only under H0, meaning that choosing this method requires to perform previously the test hypothesis between the pairs of samples. For further details, see section 'Details' below.
sym.f	A boolean indicating whether the unknown component densities are assumed to be symmetric or not.

Details

For further details on the different estimation techniques, see references below i) Patra and Sen estimator ; ii) BVdk estimator ; iii) IBM approach.

Value

An object of class 'admix_estim', containing at least 5 attributes: 1) the number of samples under study; 2) the information about the mixture components (distributions and parameters); 3) the sizes of the samples; 4) the chosen estimation technique (one of 'BVdk', 'PS' or 'IBM'); 5) the estimated mixing proportions (weights of the unknown component distributions in the mixture model). In case of 'BVdk' estimation, one additional attribute corresponding to the estimated location shift parameter is included.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Patra RK, Sen B (2016). "Estimation of a two-component mixture model with applications to multiple testing." *Journal of the Royal Statistical Society Series B*, **78**(4), 869-893. Bordes L, Delmas C, Vandekerkhove P (2006). "Semiparametric Estimation of a Two-Component Mixture Model Where One Component Is Known." *Scandinavian Journal of Statistics*, **33**(4), 733–752. ISSN 03036898, 14679469, http://www.jstor.org/stable/4616955. Bordes L, Vandekerkhove P (2010). "Semiparametric two-component mixture model with a known component: An asymptotically normal estimator." *Mathematical Methods of Statistics*, **19**(1), 22–41. doi:10.3103/S1066530710010023. Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2024). "Two-sample contamination model test." *Bernoulli*, **30**(1), 170–197. doi:10.3150/23BEJ1593.

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 250, weight = 0.7,
```

```
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
mixt2 <- twoComp_mixt(n = 200, weight = 0.85,
                      comp.dist = list("norm", "exp"),
                      comp.param = list(list("mean" = 3, "sd" = 1),
                                        list("rate" = 1)))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)

## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
admix_estim(samples = list(data1), admixMod = list(admixMod1),
            est.method = 'BVdk', sym.f = TRUE)
admix_estim(samples = list(data1,data2),
            admixMod = list(admixMod1,admixMod2),
            est.method = 'PS')
```

`admix_model` *Create an object of class 'admix_model'*

Description

Create an admixture model, also known as (aka) a contamination model. Such a model is a two-component mixture model with one known component. Both the second component distribution and the mixing weight are unknown.

Usage

```
admix_model(knownComp_dist, knownComp_param)
```

Arguments

`knownComp_dist`	(Character) The name of the distribution (specified as in R glossary) of the known component of the admixture model
`knownComp_param`	(Character) A vector of the names of the parameters (specified as in R glossary) involved in the chosen known distribution, with their values.

Value

An object of class 'admix_model', containing 2 attributes: 1) a list that gives the information about the distributions involved in the two-component mixture model (the unknown and the known ones); 2) a list that gives the information about the corresponding parameters of those distributions.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
admix_model(knownComp_dist = "norm", knownComp_param = list("mean"=0, "sd"=1))
admix_model(knownComp_dist = "exp", knownComp_param = list("rate"=2))
admix_model(knownComp_dist = "multinom", knownComp_param = list("size"=1, "prob"=c(0.2,0.8,0.1)))
```

---

admix_test *Equality test for the unknown components of admixture models*

Description

Perform hypothesis test between unknown components of a list of admixture models, where we remind that the i-th admixture model has probability density function (pdf) l_i such that: l_i = p_i * f_i + (1-p_i) * g_i, with g_i the known component density. The unknown quantities p_i and f_i are thus estimated, leading to the test given by the following null and alternative hypothesis: H0: f_i = f_j for all i != j against H1 : there exists at least i != j such that f_i differs from f_j. The test can be performed using two methods, either the comparison of coefficients obtained through polynomial basis expansions of the component densities, or by the inner-convergence property obtained using the IBM approach. See 'Details' below for further information.

Usage

```
admix_test(
  samples,
  admixMod,
  test_method = c("poly", "icv"),
  sim_U = NULL,
  n_sim_tab = 50,
  ICV_tunePenalty = FALSE,
  ask_poly_param = FALSE,
  support = c("Real", "Integer", "Positive", "Bounded.continuous"),
  conf_level = 0.95,
  parallel = FALSE,
  n_cpu = 2
)
```

Arguments

samples	(list) A list of the K (K > 0) samples to be studied, each one assumed to follow a mixture distribution.
admixMod	(list) A list of objects of class 'admix_model', containing useful information about distributions and parameters.

`test_method` The testing method to be applied. Can be either 'poly' (polynomial basis expansion) or 'icv' (inner convergence from IBM). The same testing method is performed between all samples. In the one-sample case, only 'Poly' is available and the test is a gaussianity test. For further details, see section 'Details' below.

`sim_U` (Only with 'icv' testing method, otherwise useless) Random draws of the inner convergence part of the contrast as defined in the IBM approach (see 'Details' below).

`n_sim_tab` (Only with 'icv' testing method, otherwise useless) Number of simulated gaussian processes used in the tabulation of the inner convergence distribution in the IBM approach.

`ICV_tunePenalty` (Only with 'icv' testing with at least 3 samples to deal with, otherwise useless. Default to FALSE) Boolean used to tune the penalty term in the k-sample test (k = 3,...,K) when using Inversion Best Matching (IBM) approach coupled to Inner ConVergence (icv) property. Particularly useful when studying unbalanced samples (in terms of sample size) or small-sized samples.

`ask_poly_param` (Only with 'poly' testing method, otherwise useless. Boolean, default to FALSE) If TRUE, ask the user to choose both the order 'K' of expansion coefficients in the orthonormal polynomial basis, and the penalization rate 's' involved on the penalization rule for the test. Default values for these two parameters are 'K=3' and 's=0.25'.

`support` (Used with 'poly' testing method, otherwise useless) The support of the observations; one of "Real", "Integer", "Positive", or "Bounded.continuous".

`conf_level` The confidence level of the K-sample test.

`parallel` (default to FALSE, only used with 'icv' testing) Boolean indicating whether parallel computations are performed (to speed-up the tabulation).

`n_cpu` (default to 2, only used with 'icv' testing) Number of cores used when parallelizing.

Details

For further details on implemented hypothesis tests, see the references hereafter. .

Value

An object of class 'admix_test', containing 8 attributes: 1) the test decision (reject the null hypothesis or not); 2) the p-value of the test; 3) the confidence level of the test (1-alpha, where alpha denotes the level of the test or equivalently the type-I error); 4) the value of the test statistic; 5) the number of samples under study; 6) the respective size of each sample; 7) the information about mixture components (distributions and parameters); 8) the chosen testing method (either based on polynomial basis expansions, or on the inner convergence property; see given references).

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2024). "Contamination-source based K-sample clustering." *Journal of Machine Learning Research*, **25**(287), 1–32. https://jmlr.org/papers/v25/23-0914.html. Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2022). "Semi-parametric two-sample admixture components comparison test: The symmetric case." *Journal of Statistical Planning and Inference*, **216**, 135-150. ISSN 0378-3758, doi:10.1016/j.jspi.2021.05.010. Pommeret D, Vandekerkhove P (2019). "Semiparametric density testing in the contamination model." *Electronic Journal of Statistics*, 4743–4793. doi:10.1214/19EJS1650.

Examples

```
####### Example with 2 samples
mixt1 <- twoComp_mixt(n = 280, weight = 0.7,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(c("mean" = -2, "sd" = 0.5),
                                        c("mean" = 0, "sd" = 1)))
mixt2 <- twoComp_mixt(n = 250, weight = 0.85,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(c("mean" = -2, "sd" = 0.5),
                                        c("mean" = -1, "sd" = 1)))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
admix_test(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2),
           test_method = "poly", ask_poly_param = FALSE, support = "Real",
           conf_level = 0.95, parallel = FALSE, n_cpu = 2)
```

---

allGalaxies *Measurements of heliocentric velocities in four galaxies*

Description

An evolving data frame of velocities for 4 dSph galaxies (namely Carina, Sextans, Sculptor and Fornax), from SIMBAD astronomical database.

Usage

```
allGalaxies
```

Format

Currently contains 8,862 rows and 3 columns, with information on:

Target Target identification; Galaxy-ID number

HV Weighted mean Heliocentric rest frame velocity

Name The name of the galaxy

Source

https://vizier.u-strasbg.fr/viz-bin/VizieR-3?-source=J/AJ/137/3100/stars&-out.max=50&-out.form=HTML%20Table&-out.add=_r&-out.add=_RAJ,_DEJ&-out.add=_RA%2a-c.eq,_DE%2a-c.eq&-sort=_r&-oc.form=sexa

---

`BVdk_varCov_estimators` *Variance of estimators in an admixture model with symmetric unknown density.*

Description

Semiparametric estimation of the variance of the estimators related to the mixture weight p and the location shift parameter mu, considering the admixture model with probability density function l: l(x) = p*f(x-mu) + (1-p)*g(x), x in R, where g is the known component of the two-component mixture, p is the unknown proportion, f is the unknown component density and mu is the location shift. See 'Details' below for more information.

Usage

```
BVdk_varCov_estimators(estim, data, admixMod)
```

Arguments

`estim`	An object of class 'estim_BVdk', containing the estimators of unknown quantities in the admixture model.
`data`	The observed sample under study.
`admixMod`	An object of class 'admix_model', containing useful information about distributions and parameters.

Details

See formulas pp.28–30 in Appendix of Bordes, L. and Vandekerkhove, P. (2010).

Value

A list containing 1) the variance-covariance matrix of the estimators (assessed at the specific time points 'u' and 'v' such that u = v = mean(data)); 2) the variance of the estimator of the unknown mixture weight; 3) the variance of the estimator of the location shift parameter; 4) the variance of the estimator of the unknown component cumulative distribution function at points 'u' and 'v'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Bordes L, Vandekerkhove P (2010). "Semiparametric two-component mixture model with a known component: An asymptotically normal estimator." *Mathematical Methods of Statistics*, **19**(1), 22–41. doi:10.3103/S1066530710010023.

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 200, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(c("mean" = -2, "sd" = 0.5),
                                        c("mean" = 0, "sd" = 1)))
data1 <- getmixtData(mixt1)
## Define the admixture model:
admixMod <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                        knownComp_param = mixt1$comp.param[[2]])

## Perform the estimation of parameters in real-life:
estim <- estim_BVdk(data = data1, admixMod = admixMod, method = 'L-BFGS-B')
BVdk_varCov_estimators(estim = estim, data = data1, admixMod = admixMod)
```

`decontaminated_cdf` *Estimates the decontaminated CDF of the unknown component in an admixture*

Description

Estimates the decontaminated cumulative distribution function (CDF) of the unknown component in an admixture model, using inversion of the admixture CDF. Recall that an admixture model follows the cumulative distribution function (CDF) L, where L = p*F + (1-p)*G, with g a known CDF and p and f unknown quantities.

Usage

```
decontaminated_cdf(sample1, estim.p, admixMod)
```

Arguments

`sample1`	Observations of the sample under study.
`estim.p`	The estimated weight of the unknown component distribution, related to the proportion of the unknown component in the admixture model studied.
`admixMod`	An object of class 'admix_model', containing useful information about distributions and parameters.

Details

The decontaminated CDF is obtained by inverting the admixture CDF, given by L = p*F + (1-p)*G, to isolate the unknown component F after having estimated p. This means that F = (1/hat(p)) * (hat(L)-(1-p)*G).

Value

The decontaminated CDF F of the admixture model, of class 'stepfun' (step function).

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
####### Continuous support:
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 400, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
mixt2 <- twoComp_mixt(n = 300, weight = 0.6,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 5, "sd" = 2)))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2),
                   est.method = 'PS')
prop <- getmixingWeight(est)
## Determine the decontaminated version of the unknown CDF by inversion:
F1 <- decontaminated_cdf(sample1 = data1, estim.p = prop[1], admixMod = admixMod1)
F2 <- decontaminated_cdf(sample1 = data2, estim.p = prop[2], admixMod = admixMod2)
abs <- seq(from=-1, to=4, length.out=100)
plot(x=abs, y=F1(abs), xlim=c(-1,4), ylim=c(0,1), type="l")
par(new = TRUE)
plot(x=abs, y=F2(abs), xlim=c(-1,4), ylim=c(0,1), type="l", col="red")
```

---

decontaminated_density

Estimates the decontaminated density of the unknown component in an admixture

---

Description

Estimate the decontaminated density of the unknown component in the admixture model under study, after inversion of the admixture cumulative distribution function. Recall that an admixture model follows the cumulative distribution function (CDF) L, where L = p*F + (1-p)*G, with g a known CDF and p and f unknown quantities.

Usage

```
decontaminated_density(sample1, estim.p, admixMod)
```

Arguments

sample1	Sample under study.
estim.p	The estimated weight of the unknown component distribution, related to the proportion of the unknown component in the admixture model studied.
admixMod	An object of class 'admix_model', containing useful information about distributions and parameters.

Details

The decontaminated density is obtained by inverting the admixture density, given by l = p*f + (1-p)*g, to isolate the unknown component f after having estimated p.

Value

An object of class 'decontaminated_density', containing 2 attributes: 1) the decontaminated density function; 2) the type of support for the underlying distribution (either discrete or continuous, useful for plots).

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 400, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
data1 <- getmixtData(mixt1)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1), admixMod = list(admixMod1),
                   est.method = 'PS')
## Determine the decontaminated version of the unknown density by inversion:
decontaminated_density(sample1 = data1, estim.p = est$estimated_mixing_weights[1],
                       admixMod = admixMod1)
```

```
####### Discrete support:
mixt1 <- twoComp_mixt(n = 5000, weight = 0.6,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 2)))
mixt2 <- twoComp_mixt(n = 4000, weight = 0.8,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 4)))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1, data2),
                   admixMod = list(admixMod1, admixMod2), est.method = 'IBM')
## Determine the decontaminated version of the unknown density by inversion:
decontaminated_density(sample1 = data1, estim.p = est$estimated_mixing_weights[1],
                       admixMod = admixMod1)

####### Finite discrete support:
mixt1 <- twoComp_mixt(n = 12000, weight = 0.6,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.3,0.4,0.3)),
                                        list("size" = 1, "prob" = c(0.6,0.3,0.1))))
mixt2 <- twoComp_mixt(n = 10000, weight = 0.8,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.3,0.4,0.3)),
                                        list("size" = 1, "prob" = c(0.2,0.6,0.2))))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1, data2),
                   admixMod = list(admixMod1, admixMod2), est.method = 'IBM')
## Determine the decontaminated version of the unknown density by inversion:
decontaminated_density(sample1 = data1, estim.p = est$estimated_mixing_weights[1],
                       admixMod = admixMod1)
```

`estim_BVdk` *Estimation of the admixture parameters by Bordes & Vandekerkhove (2010)*

Description

Estimates parameters in an admixture model where the unknown component is assumed to have a symmetric density. More precisely, estimates the two parameters (mixture weight and location shift) in the admixture model with pdf: l(x) = p*f(x-mu) + (1-p)*g(x), x in R, where g is the known component, p is the proportion and f is the unknown component with symmetric density. The localization shift parameter is denoted mu, and the component weight p. See 'Details' below for further information.

Usage

```
estim_BVdk(data, admixMod, method = c("L-BFGS-B", "Nelder-Mead"))
```

Arguments

`data`	The observed sample under study.
`admixMod`	An object of class 'admix_model', containing useful information about distributions and parameters.
`method`	The method used throughout the optimization process, either 'L-BFGS-B' or 'Nelder-Mead' (see ?optim).

Value

An object of class 'estim_BVdk', containing 7 attributes: 1) the number of sample under study (set to 1 here); 2) the sample size; 3) the information about mixture components (distributions and parameters); 4) the estimation method (Bordes and Vandekerkhove here, see the given reference); 5) the estimated mixing proportion (weight of the unknown component distribution); 6) the estimated location parameter of the unknown component distribution (with symetric density); 7) the optimization method that was used.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Bordes L, Vandekerkhove P (2010). "Semiparametric two-component mixture model with a known component: An asymptotically normal estimator." *Mathematical Methods of Statistics*, **19**(1), 22–41. doi:10.3103/S1066530710010023.

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 200, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
## Retrieves the mixture data:
data1 <- getmixtData(mixt1)
## Define the admixture model:
```

```
admixMod <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                        knownComp_param = mixt1$comp.param[[2]])

## Perform the estimation of parameters in real-life:
estim_BVdk(data = data1, admixMod = admixMod, method = 'L-BFGS-B')

## Second example:
mixt2 <- twoComp_mixt(n = 200, weight = 0.65,
                      comp.dist = list("norm", "exp"),
                      comp.param = list(list("mean" = -1, "sd" = 0.5),
                                        list("rate" = 1)))
data2 <- getmixtData(mixt2)
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                        knownComp_param = mixt2$comp.param[[2]])

## Perform the estimation of parameters in real-life:
estim_BVdk(data = data2, admixMod = admixMod2, method = 'L-BFGS-B')
```

`estim_IBM` *Estimates weights of unknown components from 2 admixtures using IBM*

Description

Estimation of the component weights from the Inversion - Best Matching (IBM) method, related to two admixture models with respective probability density function (pdf) l1 and l2, such that: l1 = p1*f1 + (1-p1)*g1 and l2 = p2*f2 + (1-p2)*g2, where g1 and g2 are the known component densities. For further details about IBM approach, see 'Details' below.

Usage

```
estim_IBM(samples, admixMod, n.integ = 1000)
```

Arguments

`samples`	(List) List of the two considered samples.
`admixMod`	(List) List of objects of class 'admix_model', one for each sample.
`n.integ`	Number of data points generated for the distribution on which to integrate.

Value

An object of class 'estim_IBM', containing 7 attributes: 1) the number of samples under study; 2) the sizes of samples; 3) the information about mixture components (distributions and parameters) for each sample; 4) the estimation method (Inversion Best Matching here, see the given reference); 5) the estimated mixing proportions (weights of the unknown component distributions in each sample); 6) the arbitrary value of the mixing weight in the first admixture sample (in case of equal known components, see the given reference); 7) the support of integration that was used in the computations.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2024). "Two-sample contamination model test." *Bernoulli*, **30**(1), 170–197. doi:10.3150/23BEJ1593.

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 1500, weight = 0.5,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
data1 <- getmixtData(mixt1)
mixt2 <- twoComp_mixt(n = 2000, weight = 0.7,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 5, "sd" = 2)))
data2 <- getmixtData(mixt2)

## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])

## Estimate the mixture weights of the two admixture models (provide only hat(theta)_n):
estim_IBM(samples = list(data1,data2),
          admixMod = list(admixMod1,admixMod2), n.integ = 1000)

## Example 2: multinomial distribution:
mixt1 <- twoComp_mixt(n = 1500, weight = 0.8,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.2,0.3,0.5)),
                                        list("size" = 1, "prob" = c(0.1,0.6,0.3))))
data1 <- getmixtData(mixt1)
mixt2 <- twoComp_mixt(n = 2000, weight = 0.3,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.2,0.3,0.5)),
                                        list("size" = 1, "prob" = c(0.7,0.1,0.2))))
data2 <- getmixtData(mixt2)

## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])

## Estimate the mixture weights of the two admixture models (provide only hat(theta)_n):
estim_IBM(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2))
```

estim_PS *Estimates in an admixture using Patra and Sen approach*

Description

Estimation of both the weight and the distribution of the unknown component in an admixture model, by Patra and Sen approach. Remind that the admixture probability density function (pdf) l is given by l = p*f + (1-p)*g, where g is the known component of the two-component mixture, p is the unknown proportion of the unknown component distribution f. More information in 'Details' below concerning the estimation method.

Usage

```
estim_PS(
  data,
  admixMod,
  method = c("lwr.bnd", "fixed", "cv"),
  c.n = NULL,
  folds = 10,
  reps = 1,
  cn.s = NULL,
  cn.length = 100,
  gridsize = 600
)
```

Arguments

data	Sample to be studied.
admixMod	An object of class 'admix_model', containing information about the known component distribution and its parameter(s).
method	One of 'lwr.bnd', fixed' or 'cv': depending on whether compute some lower bound of the mixing proportion, the estimate based on the value of 'c.n' or use cross-validation for choosing 'c.n' (tuning parameter).
c.n	A positive number, with default value equal to 0.1 log(log(n)), where 'n' is the length of the observed sample.
folds	(optional, default to 10) Number of folds used for cross-validation.
reps	(optional, default to 1) Number of replications for cross-validation.
cn.s	(optional) A sequence of 'c.n' to be used for cross-validation (vector of values). Default is equally spaced grid of 100 values between .001 x log(log(n)) and 0.2 x log(log(n)).
cn.length	(optional, default to 100) Number of equally spaced tuning parameter (between .001 x log(log(n)) and 0.2 x log(log(n))). Values to search from.
gridsize	(default to 600) Number of equally spaced points (between 0 and 1) to evaluate the distance function. Larger values are more computationally intensive but also lead to more accurate estimates.

Value

An object of class 'estim_PS', containing 10 attributes: 1) the number of samples studied (1 in this case); 2) the sample size; 3) the information about component distributions of the admixture model; 4) the estimation method 5patra and Sen here); 5) the estimated mixing weight (estimate of the unknown component proportion); 6) the estimated decontaminated CDF; 7) an object of the class 'dist.fun' (that gives the distance); 8) the tuning parameter 'c.n'; 9) the lower bound of the estimated mixing proportion (if such an option has been chosen); 10) the number of observations.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Patra RK, Sen B (2016). "Estimation of a two-component mixture model with applications to multiple testing." *Journal of the Royal Statistical Society Series B*, **78**(4), 869-893.

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 800, weight = 0.2,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
data1 <- getmixtData(mixt1)

## Define the admixture model:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])

## Transform the known component of the admixture model into a Uniform(0,1) distribution:
estim_PS(data = data1, admixMod = admixMod1, method = 'fixed',
         c.n = 0.1*log(log(length(data1))), gridsize = 1000)
```

`gaussianity_test` *Gaussianity test in an admixture model*

Description

Performs an hypothesis test to check for the gaussianity of the unknown mixture component, given that the known component has support on the real line. Recall that an admixture model has probability density function (pdf) l = p*f + (1-p)*g, where g is the known pdf and l is observed (others are unknown). This test requires optimization (to estimate the unknown parameters) as defined by Bordes & Vandekerkhove (2010), which means that the unknown mixture component must have a symmetric density.

Usage

```
gaussianity_test(
  sample1,
  admixMod,
  conf_level = 0.95,
  K = 3,
  s = 0.25,
  support = c("Real", "Integer", "Positive", "Bounded.continuous")
)
```

Arguments

sample1	Sample under study.
admixMod	An object of class 'admix_model', containing useful information about distributions and parameters.
conf_level	(default to 0.95) The confidence level. Equals 1-alpha, where alpha is the level of the test (type-I error).
K	(K > 0) Number of coefficients considered for the polynomial basis expansion.
s	Normalization rate involved in the penalization rule for model selection (in]0,1/2[). See the reference below.
support	Support of the probability density functions, useful to choose the polynomial orthonormal basis. One of 'Real', 'Integer', 'Positive', or 'Bounded.continuous'.

Details

Extensions to the case of non-Gaussian known components can be overcome thanks to basic transformations using cdf.

Value

An object of class 'gaussianity_test', containing 10 elements: 1) the number of populations under study (1 in this case); 2) the sample size; 3) the information about the known component distribution; 4) the reject decision of the test; 5) the confidence level of the test, 6) the p-value of the test; 7) the value of the test statistic; 8) the variance of the test statistic at each order in the polynomial orthobasis expansion; 9) the selected rank (order) for the test statistic; 10) a list of estimates (mixing weight, mean and standard deviation of the Gaussian unknown distribution).

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Pommeret D, Vandekerkhove P (2019). "Semiparametric density testing in the contamination model." *Electronic Journal of Statistics*, 4743–4793. doi:10.1214/19EJS1650.

Examples

```
####### Under the null hypothesis H0.
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 250, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(c("mean" = -2, "sd" = 0.5),
                                        c("mean" = 0, "sd" = 1)))
data1 <- getmixtData(mixt1)

## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
## Performs the test:
gaussianity_test(sample1 = data1, admixMod = admixMod1,
                 conf_level = 0.95, K = 3, s = 0.1, support = 'Real')
```

---

getmixingWeight *Extractor for object of class 'admix_estim'*

Description

Get the estimated unknown mixing proportion related to the weight of the unknown component distribution of the admixture model.

Usage

```
getmixingWeight(x)
```

Arguments

x	An object of class 'admix_estim'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

getmixtData *Extractor for object of class 'twoComp_mixt'*

Description

Get the mixture data generated from method 'twoComp_mixt'.

Usage

```
getmixtData(x)
```

Arguments

x	An object of class 'twoComp_mixt'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

IBM_k_samples_test *Equality test of K unknown component distributions*

Description

Equality test of the unknown component distributions coming from K (K > 1) admixture models, based on the Inversion - Best Matching (IBM) approach. Recall that we have K populations following admixture models, each one with probability density functions (pdf) l_k = p_k*f_k + (1-p_k)*g_k, where g_k is the known pdf and l_k corresponds to the observed sample. Perform the following hypothesis test: H0 : f_1 = ... = f_K against H1 : f_i differs from f_j (i different from j, and i,j in 1,...,K).

Usage

```
IBM_k_samples_test(
  samples,
  admixMod,
  sim_U = NULL,
  n_sim_tab = 100,
  conf_level = 0.95,
  tune_penalty = TRUE,
  parallel = FALSE,
  n_cpu = 2
)
```

Arguments

samples	(list) A list of the K samples to be studied, all following admixture distributions.
admixMod	(list) A list of objects of class 'admix_model', containing useful information about distributions and parameters.
sim_U	(default to NULL) Random draws of the inner convergence part of the contrast as defined in the IBM approach (see 'Details' below).
n_sim_tab	Number of simulated gaussian processes when tabulating the inner convergence distribution in the IBM approach.
conf_level	The confidence level of the K-sample test.
tune_penalty	A boolean that allows to choose between a classical penalty term or an optimized penalty embedding some tuning parameters (automatically optimized). Optimized penalty is particularly useful for low sample size to detect alternatives.

parallel	(default to FALSE) Boolean indicating whether parallel computations are performed.
n_cpu	(default to 2) Number of cores used when parallelizing.

Value

An object of class 'IBM_test', containing 17 attributes: 1) the number of samples for the test; 2) the sizes of each sample; 3) the information about component distributions for each sample; 4) the reject decision of the test; 5) the confidence level of the test (1-alpha, where alpha refers to the first-type error); 6) the test p-value; 7) the 95th-percentile of the contrast tabulated distribution; 8) the test statistic value; 9) the selected rank (number of terms involved in the test statistic); 10) the terms (pairwise contrasts) involved in the test statistic; 11) A boolean indicating whether the penalization corresponds to the null hypothesis has been considered; 12) the value of penalized test statistics; 13) the selected optimal 'gamma' parameter (see reference below); 14) the selected optimal constant involved in the penalization process (see also the reference); 15) the tabulated distribution of the contrast; 16) the estimated mixing proportions (not implemented yet, since that makes sense only in case of equal unknown component distributions); 17) the matrix of pairwise contrasts (distance between two samples).

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2024). "Contamination-source based K-sample clustering." *Journal of Machine Learning Research*, **25**(287), 1–32. https://jmlr.org/papers/v25/23-0914.html.

Examples

```
####### Under the null hypothesis H0 (with K=3 populations):
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 450, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
mixt2 <- twoComp_mixt(n = 380, weight = 0.7,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 1, "sd" = 1)))
mixt3 <- twoComp_mixt(n = 400, weight = 0.8,
                      comp.dist = list("norm", "exp"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("rate" = 1)))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
data3 <- getmixtData(mixt3)

## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
```

```
                          knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                          knownComp_param = mixt2$comp.param[[2]])
admixMod3 <- admix_model(knownComp_dist = mixt3$comp.dist[[2]],
                          knownComp_param = mixt3$comp.param[[2]])
## Perform the 3-samples test:
IBM_k_samples_test(samples = list(data1, data2, data3),
                   admixMod = list(admixMod1, admixMod2, admixMod3),
                   sim_U = NULL, n_sim_tab = 8, conf_level = 0.95,
                   tune_penalty = FALSE, parallel = FALSE, n_cpu = 2)
```

`IBM_tabul_stochasticInteg` *Simulated distribution of the contrast using IBM*

Description

Tabulate the distribution related to the inner convergence part of the contrast, by simulating trajectories of Gaussian processes and applying some transformations. Useful to perform the test hypothesis, by retrieving the (1-alpha)-quantile of interest. See 'Details' below and the cited paper therein for further information.

Usage

```
IBM_tabul_stochasticInteg(
  n.sim = 200,
  n.varCovMat = 100,
  samples,
  admixMod,
  min_size = NULL,
  parallel = FALSE,
  n_cpu = 2
)
```

Arguments

`n.sim`	Number of trajectories for the simulated Gaussian processes (number of random draws for tabulation).
`n.varCovMat`	Number of time points at which the Gaussian processes are simulated.
`samples`	(list) A list of the two samples under study.
`admixMod`	(list) A list of two objects of class 'admix_model', with information about distributions and parameters.
`min_size`	(optional, NULL by default) In the k-sample case, useful to provide the minimal size among all samples (needed to take into account the correction factor for variance-covariance assessment). Otherwise, useless.

parallel	(default to FALSE) Boolean to indicate whether parallel computations are performed (speed-up the tabulation).
n_cpu	(default to 2) Number of cores used for computations when parallelizing.

Value

A list with four elements, containing: 1) random draws of the contrast as defined in the reference given here; 2) estimated unknown component weights for the two admixture models; 3) the value of the the empirical contrast; 4) support that was used to evaluate the variance-covariance matrix of the empirical processes.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2024). "Two-sample contamination model test." *Bernoulli*, **30**(1), 170–197. doi:10.3150/23BEJ1593.

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 1200, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
mixt2 <- twoComp_mixt(n = 1000, weight = 0.7,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 1, "sd" = 1)))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
IBM_tabul_stochasticInteg(n.sim = 2, n.varCovMat = 20, samples = list(data1, data2),
                          admixMod = list(admixMod1, admixMod2), min_size = NULL,
                          parallel = FALSE, n_cpu = 2)
```

milkyWay *Heliocentric velocity for the Milky Way*

Description

Heliocentric velocity for the Milky Way

Usage

```
milkyWay
```

Format

A data frame with 170,601 rows and 1 column:

V1 Heliocentric velocity measurements of the Milky way

Source

https://www.aanda.org/articles/aa/full_html/2018/08/aa32905-18/aa32905-18.html

References

Walker MG, Mateo M, Olszewski EW, Gnedin OY, Wang X, Sen B, Woodroofe M (2007). "Velocity Dispersion Profiles of Seven Dwarf Spheroidal Galaxies." *The Astrophysical Journal*, **667**(1), L53–L56. ISSN 1538-4357, doi:10.1086/521998, http://dx.doi.org/10.1086/521998.

mortality_sample *Deaths statistics in 11 european countries*

Description

Deaths statistics in 11 european countries

Usage

```
mortality_sample
```

Format

Dataset providing the exposure-to-death (population size) and number of deaths for males in 11 european countries, between 1908 and 2020, with ages ranging from 30 years old to 85 years old. Exported from the Human Mortality Database (HMD). The two first lists relate to some subsample of the population size and number of deaths in those countries, with random sampling from the original dataset.

An evolving data frame of exposure-to-death and number of deaths in Belgium, Switzerland, Denmark, Spain, Finland, France, United Kingdom, Italia, The Netherlands, Norway and Sweden.

XP A list of eleven elements (one for each country) giving a subset of the exposure-to-death (or reduced population size), each element having 56 rows (ages 30-85) and 113 columns (period 1908-2020)

DX A list of eleven elements (one for each country) giving a subset of the number of deaths, each element having 56 rows (ages 30-85) and 113 columns (period 1908-2020)

names A list of eleven elements giving the names of the countries, in the same order as the elements in other lists

Source

https://www.mortality.org

`orthobasis_test` *Equality test of two unknown component distributions using polynomial expansions*

Description

Tests the null hypothesis (H0: f1=f2) using the decomposition of unknown component densities of two admixture distributions in an adequate orthonormal polynomial basis. Recall that we have two admixture models with respective probability density functions (pdf) l1 = p1*f1 + (1-p1)*g1 and l2* = *p2*f2 + (1-p2)*g2, where g1 and g2 are the only known elements and l1 and l2 are observed. The admixture weights p1 and p2 thus have to be estimated. For further information on this method, see 'Details' below.

Usage

```
orthobasis_test(
  samples,
  admixMod,
  K = 3,
  s = 0.49,
  est_method = c("BVdk", "PS"),
  conf_level = 0.95,
  nb_echBoot = NULL,
  bounds_supp = NULL,
 support = c("Real", "Integer", "Positive", "Bounded.continuous", "Bounded.discrete")
)
```

Arguments

samples	(List) List of the two samples, each one following the mixture distribution given by l = p*f + (1-p)*g, with f and p unknown and g known.
admixMod	An object of class 'admix_model', containing useful information about distributions and parameters.
K	Number of coefficients considered for the polynomial basis expansion.
s	Rate at which the normalization factor is set in the penalization rule for model selection (in]0,1/2[), see 'Details'.
est_method	Estimation method to get the component weights, either 'PS' (Patra and Sen estimation) or 'BVdk' (Bordes and Vendekerkhove estimation). Choosing 'PS' requires to specify the number of bootstrap samples.
conf_level	The confidence level, default to 95 percent. Equals 1-alpha, where alpha is the level of the test (type-I error).
nb_echBoot	(default to NULL) Number of bootstrap samples, useful when choosing 'PS' estimation method.
bounds_supp	(default to NULL) useful if support = 'bounded', a list of minimum and maximum bounds, specified as following: list(list(min.f1,min.g1,min.f2,min.g2) , list(max.f1,max.g1,max.f2,max.g2))
support	support of the densities under consideration, useful to choose the polynomial orthonormal basis.

Value

An object of class 'orthobasis_test', containing ten attributes: 1) the number of populations under study (2 in this case); 2) the sizes of samples; 3) the information about the known component distribution; 4) the reject decision of the test; 5) the confidence level of the test, 6) the p-value of the test; 7) the value of the test statistic; 8) the variance of the test statistic at each order in the polynomial orthobasis expansion; 9) the selected rank (order) for the test statistic; 10) a vector of estimates, related to the estimated mixing proportions in the two samples.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2022). "Semiparametric two-sample admixture components comparison test: The symmetric case." *Journal of Statistical Planning and Inference*, **216**, 135-150. ISSN 0378-3758, doi:10.1016/j.jspi.2021.05.010.

Examples

```
#### Under the null hypothesis H0.
mixt1 <- twoComp_mixt(n = 300, weight = 0.77,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 1, "sd" = 1),
                                        list("mean" = 4, "sd" = 1)))
```

```
data1 <- getmixtData(mixt1)
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
mixt2 <- twoComp_mixt(n = 500, weight = 0.62,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 1, "sd" = 1),
                                        list("mean" = -2, "sd" = 0.5)))
data2 <- getmixtData(mixt2)
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])

orthobasis_test(samples = list(data1,data2),
                admixMod = list(admixMod1,admixMod2),
                K = 3, s = 0.49, nb_echBoot = NULL, support = 'Real',
                bounds_supp = NULL, est_method = 'BVdk')
```

`plot.decontaminated_density` *Plot method for class 'decontaminated_density'*

Description

Plot the decontaminated density of the unknown component from some admixture model, after inversion of the admixture cumulative distribution functions.

Usage

```
## S3 method for class 'decontaminated_density'
plot(x, x_val, add_plot = FALSE, ...)
```

Arguments

`x`	An object of class 'decontaminated_density' (see ?decontaminated_density).
`x_val`	(numeric) A vector of points at which to evaluate the probability mass/density function.
`add_plot`	(default to FALSE) A boolean specifying if one plots the decontaminated density over an existing plot. Used for visual comparison purpose.
`...`	Arguments to be passed to generic method 'plot', such as graphical parameters (see par).

Details

The decontaminated density is obtained by inverting the admixture density, given by l = p*f + (1-p)*g, to isolate the unknown component f after having estimated p and l.

Value

The plot of the decontaminated density.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
####### Continuous support:
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 400, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
mixt2 <- twoComp_mixt(n = 350, weight = 0.6,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 5, "sd" = 2)))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2),
                   est.method = 'PS')
prop <- getmixingWeight(est)
## Determine the decontaminated version of the unknown density by inversion:
res1 <- decontaminated_density(sample1 = data1, estim.p = prop[1], admixMod = admixMod1)
res2 <- decontaminated_density(sample1 = data2, estim.p = prop[2], admixMod = admixMod2)
## Use appropriate sequence of x values:
plot(x = res1, x_val = seq(from = 0, to = 6, length.out = 100), add_plot = FALSE)
plot(x = res2, col = "red", x_val = seq(from = 0, to = 6, length.out = 100), add_plot = TRUE)

####### Countable discrete support:
mixt1 <- twoComp_mixt(n = 4000, weight = 0.7,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 2)))
mixt2 <- twoComp_mixt(n = 3500, weight = 0.85,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 4)))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
```

```
                        knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2),
                   est.method = "IBM")
prop <- getmixingWeight(est)
## Determine the decontaminated version of the unknown density by inversion:
res1 <- decontaminated_density(sample1 = data1, estim.p = prop[1],
                               admixMod = admixMod1)
res2 <- decontaminated_density(sample1 = data2, estim.p = prop[2],
                               admixMod = admixMod2)
## Use appropriate sequence of x values:
plot(x = res1, x_val = seq(from = 0, to = 15, by = 1), add_plot = FALSE)
plot(x = res2, x_val = seq(from = 0, to = 15, by = 1), add_plot = TRUE, col = "red")

####### Finite discrete support:
mixt1 <- twoComp_mixt(n = 4000, weight = 0.7,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.3,0.4,0.3)),
                                        list("size" = 1, "prob" = c(0.6,0.3,0.1))))
mixt2 <- twoComp_mixt(n = 3500, weight = 0.85,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.3,0.4,0.3)),
                                        list("size" = 1, "prob" = c(0.2,0.6,0.2))))
data1 <- getmixtData(mixt1)
data2 <- getmixtData(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2),
                   est.method = "IBM")
prop <- getmixingWeight(est)
## Determine the decontaminated version of the unknown density by inversion:
res1 <- decontaminated_density(sample1 = data1, estim.p = prop[1],
                               admixMod = admixMod1)
res2 <- decontaminated_density(sample1 = data2, estim.p = prop[2],
                               admixMod = admixMod2)
## Use appropriate sequence of x values:
plot(x = res1, x_val = seq(from=1, to=3, by=1), add_plot = FALSE)
plot(x = res2, x_val = seq(from=1, to=3, by=1), add_plot = TRUE, col = "red")
```

`plot.twoComp_mixt` *Plots several mixture densities on the same graph*

Description

Plots the empirical densities of the samples with optional arguments to improve the visualization.

Usage

```
## S3 method for class 'twoComp_mixt'
plot(x, add.plot = FALSE, ...)
```

Arguments

x	Object of class 'twoComp_mixt' from which the density will be plotted.
add.plot	(default to FALSE) Option to plot another mixture distribution on the same graph.
...	further classical arguments and graphical parameters for methods plot and hist.

Value

a plot with the densities of the samples provided as inputs.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

print.admix_cluster *Print method for object of class 'admix_cluster'*

Description

Print the main results when clustering the unknown component distributions coming from various admixture samples, i.e. the obtained clusters.

Usage

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

Arguments

x	An object of class 'admix_cluster' (see ?admix_clustering).
...	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.admix_estim *Print the estimated parameters from K admixture models*

Description

Print the estimated parameters from K admixture models

Usage

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

Arguments

x	An object of class 'admix_estim' (see ?admix_estim).
...	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.admix_model *Print method for objects of class 'admix_model'*

Description

Print an object of class 'admix_mod'. An admixture model has probability density function (pdf) l_i such that: l_i = p_i * f_i + (1-p_i) * g_i, with g_i the known component density. The unknown quantities are therefore p_i and f_i.

Usage

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

Arguments

x	An object of class 'admix_model'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.admix_test *Print method for objects 'admix_test'*

Description

Print method for objects 'admix_test'

Usage

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

Arguments

x	An object of class 'admix_test'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.decontaminated_density *Print method for object of class 'decontaminated_density'*

Description

Print some overview of the decontaminated density function.

Usage

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

Arguments

x	An object of class 'decontaminated_density' (see ?decontaminated_density).
...	Arguments to be passed to generic method 'plot', such as graphical parameters (see par).

Value

More important information about the decontaminated density.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.estim_BVdk *Print method for objects 'estim_BVdk'*

Description

Print the results stored in an object of class 'estim_BVdk'.

Usage

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

Arguments

x	An object of class 'estim_BVdk'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.estim_IBM *Print method for objects of class 'estim_IBM'*

Description

Print the results stored in an object of class 'estim_IBM'.

Usage

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

Arguments

x	An object of class 'estim_IBM'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.estim_PS *Print method for objects of class 'estim_PS'*

Description

Print all the attributes of objects of class 'estim_PS'. Results of estimated quantities in an admixture using Patra and Sen approach

Usage

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

Arguments

x	An object of class 'estim_PS'.
...	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.gaussianity_test *Print method for objects 'gaussianity_test'*

Description

Print method for objects 'gaussianity_test'

Usage

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

Arguments

x	An object of class 'gaussianity_test'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.IBM_test *Print method for objects 'IBM_test'*

Description

Print method for objects 'IBM_test'

Usage

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

Arguments

x	An object of class 'IBM_test'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.orthobasis_test *Print method for objects of class 'orthobasis_test'*

Description

Print method for objects of class 'orthobasis_test'

Usage

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

Arguments

x	An object of class 'orthobasis_test'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.twoComp_mixt *Print method for objects 'twoComp_mixt'*

Description

Print an object of class 'twoComp_mixt'. A two-component mixture model has probability density function (pdf) l such that: l = p * f + (1-p) * g, where p is the mixing proportion, and f and g are the component distributions.

Usage

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

Arguments

x	An object of class 'twoComp_mixt'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

stmf_small *Short-term Mortality Fluctuations (STMF) data*

Description

Restricted to 6 countries: Belgium, France, Italy, Netherlands, Spain, Germany. Weekly death counts provide the most objective and comparable way of assessing the scale of short-term mortality elevations across countries and time. Extraction date from the Human Mortality Database (HMD): 09/21/2020.

Usage

```
stmf_small
```

Format

A data frame with 88146 rows and 19 variables:

CountryCode Mortality database country code

Year Year

Week Week number

Sex Gender ('m': male, 'f': female, 'b': both)

D0_14 Age range 0-14

D15_64 Age range 15-64

D65_74 Age range 65-74

D75_84 Age range 75-84

D85p Age range 85-+

DTotal Count of deaths for all ages combined

R0_14 Crude death rate for age range 0-14

R15_64 Crude death rate for age range 15-64

R65_74 Crude death rate for age range 65-74

R75_84 Crude death rate for age range 75-84

R85p Crude death rate for age range 85-+

RTotal Crude death rate for all ages combined

Split Indicates if data were split from aggregated age groups (0 if the original data has necessary detailed age scale). For example, if the original age scale was 0-4, 5-29, 30-65, 65+, then split will be equal to 1

SplitSex Indicates if the original data are available by sex (0) or data are interpolated (1)

Forecast Equals 1 for all years where forecasted population exposures were used to calculate weekly death rates

Source

https://www.mortality.org

---

summary.admix_cluster *Summary method for object of class 'admix_cluster'*

Description

Summarizes the results obtained when clustering the unknown component distributions coming from various admixture samples.

Usage

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

Arguments

object	An object of class 'admix_cluster' (see ?admix_clustering).
...	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.admix_estim *Results of estimated parameters from K admixture models*

Description

Summarize the estimated weight(s) of the unknown component(s) in the admixture model(s) under study. Recall that an admixture model follows the cumulative distribution function (CDF) L, where L = p*F + (1-p)*G, with G a known CDF, and p and F unknown quantities.

Usage

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

Arguments

object	An object of class 'admix_estim' (see ?admix_estim).
...	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.admix_test *Summary method for 'admix_test' objects*

Description

Print the decision (as well as other useful information) of the statistical test with null hypothesis corresponding to the equality of unknown component distributions in admixture models. More precisely, given two (or more) admixture models with cumulative distribution functions (CDF) L1 and L2, where Li = pi*Fi + (1-pi)*Gi i=1,2 and Gi are the known CDFs, the function performs the test: H0: F1 = F2 versus H1: F1 != F2.

Usage

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

Arguments

object	An object of class 'admix_test' (see ?admix_test).
...	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.estim_BVdk *Summary method for objects 'estim_BVdk'*

Description

Summarizes the results stored in an object of class 'estim_BVdk'.

Usage

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

Arguments

object	An object of class 'estim_BVdk'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.estim_IBM *Summary method for objects 'estim_IBM'*

Description

Summarizes the results stored in an object of class 'estim_IBM'.

Usage

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

Arguments

object	An object of class 'estim_IBM'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.estim_PS *Summary method for objects 'estim_PS'*

Description

Summarizes the results stored in an object of class 'estim_PS'.

Usage

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

Arguments

object	An object of class 'estim_PS'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.gaussianity_test *Summary method for objects 'gaussianity_test'*

Description

Summary method for objects 'gaussianity_test'

Usage

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

Arguments

object	An object of class 'gaussianity_test'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.IBM_test *Summary method for objects 'IBM_test'*

Description

Summary method for objects 'IBM_test'

Usage

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

Arguments

object	An object of class 'IBM_test'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.orthobasis_test *Summary method for objects of class 'orthobasis_test'*

Description

Summary method for objects of class 'orthobasis_test'

Usage

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

Arguments

object	An object of class 'orthobasis_test'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

twoComp_mixt *Simulation of a two-component mixture model*

Description

Simulate a two-component mixture model following the probability density function (pdf) l such that l = p*f + (1-p)*g, with f and g the mixture component distributions, and p the mixing weight.

Usage

```
twoComp_mixt(
  n = 1000,
  weight = 0.5,
  comp.dist = list("norm", "norm"),
  comp.param = list(c(mean = 0, sd = 1), c(mean = 2, sd = 1))
)
```

Arguments

n	Number of observations to be simulated.
weight	Weight of the first component distribution (distribution f) in the mixture.
comp.dist	A list of two elements corresponding to the component distributions (specified with R native names) involved in the mixture model. These elements respectively refer to the two component distributions f and g.
comp.param	A list of two elements corresponding to the parameters of the component distributions, each element being a list itself. The names used in each list must correspond to the native R argument names for these distributions. These elements respectively refer to the parameters of f and g distributions of the mixture model.

Value

An object of class 'twoComp_mixt', containing eight attributes: 1) the number of simulated observations, 2) the simulated mixture data, 3) the support of the distributions, 4) the name of the component distributions, 5) the name of the parameters of the component distributions and their values, 6) the mixing proportion, 7) the observations coming from the first component, 8) the observations coming from the second component.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
## Mixture of continuous random variables:
sim.X <- twoComp_mixt(n = 2000, weight = 0.5, comp.dist = list("norm", "norm"),
                    comp.param = list(list("mean"=3, "sd"=0.5), list("mean"=0, "sd"=1)))
sim.Y <- twoComp_mixt(n = 1200, weight = 0.7, comp.dist = list("norm", "exp"),
                      comp.param = list(list("mean"=-3, "sd"=0.5), list("rate"=1)))
print(sim.X)
print(sim.Y)
plot(sim.X, xlim=c(-5,5), ylim=c(0,0.5))
plot(sim.Y, add.plot = TRUE, xlim=c(-5,5), ylim=c(0,0.5), col = "red")

## Mixture of discrete random variables:
sim.X <- twoComp_mixt(n = 2000, weight = 0.5, comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size"=1, "prob"=c(0.3,0.4,0.3)),
                      list("size"=1, "prob"=c(0.1,0.2,0.7))))
print(sim.X)
plot(sim.X)
```

Index