Package: watson (via r-universe)

August 24, 2024

Type Package

Title Fitting and Simulating Mixtures of Watson Distributions

Version 0.4

Maintainer Lukas Sablica <lsablica@wu.ac.at>

Description Tools for fitting and simulating mixtures of Watson distributions. The random sampling scheme of the package offers two sampling algorithms that are based of the results of Sablica, Hornik and Leydold (2022) <https://research.wu.ac.at/en/publications/random-sampling-from-the-watson-distribution>. What is more, the package offers a smart tool to combine these two methods, and based on the selected parameters, it approximates the relative sampling speed for both methods and picks the faster one. In addition, the package offers a fitting function for the mixtures of Watson distribution, that uses the expectation-maximization (EM) algorithm. Special features are the possibility to use multiple variants of the E-step and M-step, sparse matrices for the data representation and state of the art methods for numerical evaluation of needed special functions using the results of Sablica and Hornik (2022) <https://www.ams.org/journals/mcom/2022-91-334/S0025-5718-2021-03690-X/>.

License GPL-3

Encoding UTF-8

Imports Rcpp (>= 0.12.18), Tinflex (>= 1.8)

LinkingTo Rcpp, RcppArmadillo, Tinflex

RoxygenNote 7.2.3

Depends R (>= 3.1.0)

NeedsCompilation yes

Author Lukas Sablica [aut, cre] (<https://orcid.org/0000-0001-9166-4563>), Kurt Hornik [aut] (<https://orcid.org/0000-0003-4198-9911>), Josef Leydold [aut] (<https://orcid.org/0000-0002-9076-4893>), Gerard Jungman [ctb,

cph] (Author and copyright holder of included GNU GSL code),
Brian Gough [ctb, cph] (Author and copyright holder of included
GNU GSL code)

Repository CRAN

Date/Publication 2023-06-01 09:20:02 UTC

Contents

`diam_clus` *Diametrical clustering*

Description

`diam_clus` clusters axial data on sphere using the algorithm proposed in Dhillon et al. (2003).

Usage

```
diam_clus(x, k, niter = 100)
```

Arguments

`x`	a numeric data matrix, with rows corresponding to observations. Can be a dense matrix, or any of the supported sparse matrices from `Matrix` package by `Rcpp`.
`k`	an integer giving the number of mixture components.
`niter`	integer indicating the number of iterations of the diametrical clustering algorithm, default: 100.

Value

a matrix with the concentration directions with an attribute "id" defining the classified categories.

References

Inderjit S Dhillon, Edward M Marcotte, and Usman Roshan. Diametrical clustering for identifying anti-correlated gene clusters. Bioinformatics, 19(13):1612-1619, 2003.

Examples

```
## Generate a sample
a <- rmwat(n = 200, weights = c(0.5,0.5), kappa = c(20, 20),
                        mu = matrix(c(1,1,-1,1),nrow = 2))
## Fit basic model
q <- diam_clus(a, 2)
```

---

rmwat *Random Sampling from a Mixture of Watson Distributions*

Description

rmwat generates a random sample from a mixture of multivariate Watson distributions.

Usage

```
rmwat(n, weights, kappa, mu, method = "acg", b = -10, rho = 1.1)
```

Arguments

n	an integer giving the number of samples to draw.
weights	a numeric vector with non-negative elements giving the mixture probabilities.
kappa	a numeric vector giving the kappa parameters of the mixture components.
mu	a numeric matrix with columns giving the mu parameters of the mixture components.
method	a string indicating whether ACG sampler (method = "acg"), Tinflex sampler (method = "tinflex") or automatic selection (method = "auto") of the sampler should be used, default: "acg".
b	a positive numeric hyper-parameter used in the sampling. If not a positive value is given, optimal choice of b is used, default: -10.
rho	performance parameter: requested upper bound for ratio of area below hat to area below squeeze (numeric). See Tinflex.setup, default: 1.1.

Details

The function generates samples from finite mixtures of Watson distributions, using methods from Sablica, Hornik and Leydold (2022) https://research.wu.ac.at/en/publications/random-sampling-from-the-wat

Value

A matrix with rows equal to the generated values.

References

Sablica, Hornik and Leydold (2022). Random Sampling from the Watson Distribution https://research.wu.ac.at/en/publications/random-sampling-from-the-watson-distribution.

Examples

```
## simulate from Watson distribution
sample1 <- rmwat(n = 20, weights = 1, kappa = 20, mu = matrix(c(1,1,1),nrow = 3))

## simulate from a mixture of Watson distributions
sample2 <- rmwat(n = 20, weights = c(0.5,0.5), kappa = c(-200,-200),
                           mu = matrix(c(1,1,1,-1,1,1),nrow = 3))
```

`watson` *Fit Mixtures of Watson Distributions*

Description

`watson` fits a finite mixture of multivariate Watson distributions.

Usage

```
watson(x, k, control = list(), ...)
```

Arguments

`x`	a numeric data matrix, with rows corresponding to observations. Can be a dense matrix, or any of the supported sparse matrices from `Matrix` package by `Rcpp`.
`k`	an integer giving the number of mixture components.
`control`	a list of control parameters. See Details.
`...`	a list of control parameters (overriding those specified in control).

Details

watson returns an object of class "watfit" representing the fitted mixture of Watson distributions model. Available methods for such objects include `coef`, `logLik`, `print` and `predict`. `predict` has an extra type argument with possible values `"class_ids"` (default) and `"memberships"` for indicating hard or soft prediction, respectively.

The mixture of Watson distributions is fitted using EM variants as specified by control option E (specifying the E-step employed), with possible values "softmax" (default), "hardmax" or "stochmax". For "stochmax", class assignments are drawn from the posteriors for each observation in the E-step as outlined as SEM in Celeux and Govaert (1992). The stopping criterion for this algorithm is by default changed to not check for convergence (logical control option converge), but to return the parameters with the maximum likelihood encountered.

In the M-step, the parameters of the respective component distributions are estimated via maximum likelihood, which is accomplished by solving the equation

$$g(\alpha, \beta, \kappa) = r,$$

where

$$0 < \alpha < \beta,\ 0 \leq r \leq 1$$

and

$$g(\alpha, \beta, \kappa) = (\alpha/\beta)M(\alpha + 1, \beta + 1, \kappa)/M(\alpha, \beta, \kappa),$$

with M being the Kummer's function. Via control argument M, one can specify how to (approximately) solve these equations. The possible methods are:

"Sra_2007" uses the approximation of Sra (2007).

"BBG" uses the approximation of Bijral et al. (2007), without the correction term.

"BBG_c" uses the approximation of Bijral et al. (2007), with the correction term.

"Sra_Karp_2013" uses the bounds derived in Sra and Karp (2013), with the decision rule .

"bisection" uses a bisection to solve the problem using evaluation proposed in Writeup1 (2018).

"newton" uses a bracketet type of Neton algorithm to solve the problem using evaluation proposed in Writeup1 (2018). (Default.)

"lognewton" uses a bracketet type of Neton algorithm to solve the problem $log(g((\alpha, \beta, \kappa)) = log(r)$ using evaluation proposed in Writeup1 (2018).

Additional control parameters are as follows.

maxiter an integer giving the maximal number of EM iterations to be performed, default: 100.

reltol the minimum relative improvement of the objective function per iteration. If improvement is less, the EM algorithm will stop under the assumption that no further significant improvement can be made, defaults to sqrt(.Machine$double.eps).

ids either a vector of class memberships or TRUE which implies that the class memberships are obtained from the attribute named "id" of the input data; these class memberships are used for initializing the EM algorithm and the algorithm is stopped after the first iteration.

init_iter a numeric vector setting the number of diametrical clustering iterations to do, before the EM starts, default: 0.

start a specification of the starting values to be employed. Can be a list of matrices giving the memberships of objects to components. This information is combined with the `init_iter` parameter and together form the initialization procedure. If nothing is given, the starting values are drwan randomly.

If several starting values are specified, the EM algorithm is performed individually to each starting value, and the best solution found is returned.

nruns an integer giving the number of EM runs to be performed. Default: 1. Only used if start is not given.

minweight a numeric indicating the minimum prior probability. Components falling below this threshold are removed during the iteration. If is greater than 1, the value is taken as the minimal number of observations in a component, default: 0 if E = "softmax" and 2 if other type of E-method is used .

converge a logical, if TRUE the EM algorithm is stopped if the reltol criterion is met and the current parameter estimate is returned. If FALSE the EM algorithm is run for maxiter iterations and the parametrizations with the maximum likelihood encountered during the EM algorithm is returned. Default: TRUE, changed to FALSE if E="stochmax".

N an integer indicating number of iteration used when the Kummer function is approximate, default: 30.

verbose a logical indicating whether to provide some output on algorithmic progress, default: FALSE.

Value

An object of class "watfit" representing the fitted mixture, which is a list containing the fitted weights, concentrations parameters (kappa), concentrations directions (mu) and further metadata.

Examples

```
## Generate a sample with two orthogonal circles (negative kappas)
a <- rmwat(n = 200, weights = c(0.5,0.5), kappa = c(-200,-200),
                        mu = matrix(c(1,1,1,-1,1,1),nrow = 3))
## Fit basic model
q <- watson(a, 2)
## Fit the models, giving the true categories
q <- watson(a, 2, ids=TRUE)
## Fit a model with hard-assignment, and 50 runs
q <- watson(a, 2, E = "hard", nruns = 50)
## Print details
q
## Extract coefficients
coef(q)
## Calculate likelihood for new data
a2 <- rmwat(n = 20, weights = c(0.5,0.5), kappa = c(-200,-200),
                        mu = matrix(c(1,1,1,-1,1,1),nrow = 3))
logLik(q, a2)
## Compare the fitted classes to the true ones:
table(True = attr(a, "id"), Fitted = predict(q))
```

Index