Package 'openEBGM'

September 15, 2023

Title EBGM Disproportionality Scores for Adverse Event Data Mining

Version 0.9.1

Maintainer John Ihrie <John.Ihrie@fda.hhs.gov>

Description An implementation of DuMouchel's (1999) <doi:10.1080/00031305.1999.10474456> Bayesian data mining method for the market basket problem.
Calculates Empirical Bayes Geometric Mean (EBGM) and posterior quantile scores using the Gamma-Poisson Shrinker (GPS) model to find unusually large cell counts in large, sparse contingency tables. Can be used to find unusually high reporting rates of adverse events associated with products. In general, can be used to mine any database where the co-occurrence of two variables or items is of interest. Also calculates relative and proportional reporting ratios.
Builds on the work of the 'PhViD' package, from which much of the code is derived. Some of the added features include stratification to adjust for confounding variables and data squashing to improve computational efficiency.
Includes an implementation of the EM algorithm for hyperparameter estimation loosely derived from the 'mederrRank' package.

Depends R (>= 3.2.3)

License GPL-2 | GPL-3

URL https://journal.r-project.org/archive/2017/RJ-2017-063/index.html

LazyData TRUE

RoxygenNote 7.2.3

Imports data.table (>= 1.10.0), ggplot2 (>= 2.2.1), stats (>= 3.2.3)

Suggests DEoptim (>= 2.2), dplyr (>= 0.5.0), knitr (>= 1.15.1), rmarkdown (>= 1.2), testthat (>= 1.0.2), tidyr (>= 0.6.0)

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Author John Ihrie [cre, aut],
Travis Canida [aut],
Ismaïl Ahmed [ctb] (author of 'PhViD' package (derived code)),
Antoine Poncet [ctb] (author of 'PhViD'),
Sergio Venturini [ctb] (author of 'mederrRank' package (derived code)),
Jessica Myers [ctb] (author of 'mederrRank')

Repository CRAN

Date/Publication 2023-09-14 22:40:02 UTC

R topics documented:

`autoHyper` *Semi-automated hyperparameter estimation*

Description

`autoHyper` finds a single hyperparameter estimate using an algorithm that evaluates results from multiple starting points (see `exploreHypers`). The algorithm verifies that the optimization converges within the bounds of the parameter space and that the chosen estimate (smallest negative log-likelihood) is similar to at least one (see `min_conv` argument) of the other convergent solutions.

Usage

```
autoHyper(
  data,
  theta_init,
  squashed = TRUE,
  zeroes = FALSE,
  N_star = 1,
  tol = c(0.05, 0.05, 0.2, 0.2, 0.025),
  min_conv = 1,
```

```
  param_limit = 100,
  max_pts = 20000,
  conf_ints = FALSE,
  conf_level = c("95", "80", "90", "99")
)
```

Arguments

`data`	A data frame from `processRaw` containing columns named *N*, *E*, and (if squashed) *weight*.
`theta_init`	A data frame of initial hyperparameter guesses with columns ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
`squashed`	A scalar logical (`TRUE` or `FALSE`) indicating whether or not data squashing was used.
`zeroes`	A scalar logical specifying if zero counts are included.
`N_star`	A positive scalar whole number value for the minimum count size to be used for hyperparameter estimation. If zeroes are used, set `N_star` to `NULL`.
`tol`	A numeric vector of tolerances for determining how close the chosen estimate must be to at least `min_conv` convergent solutions. Order is α_1, β_1, α_2, β_2, P.
`min_conv`	A scalar positive whole number for defining the minimum number of convergent solutions that must be close to the convergent solution with the smallest negative log-likelihood. Must be at least one and at most one less than the number of rows in `theta_init`.
`param_limit`	A scalar numeric value for the largest acceptable value for the α and β estimates. Used to help protect against unreasonable/erroneous estimates.
`max_pts`	A scalar whole number for the largest number of data points allowed. Used to help prevent extremely long run times.
`conf_ints`	A scalar logical indicating if confidence intervals and standard errors should be returned.
`conf_level`	A scalar string for the confidence level used if confidence intervals are requested.

Details

The algorithm first attempts to find a consistently convergent solution using `nlminb`. If it fails, it will next try `nlm`. If it still fails, it will try `optim` (*method = "BFGS"*). If all three approaches fail, the function returns an error message.

Since this function runs multiple optimization procedures, it is best to start with 5 or less initial starting points (rows in `theta_init`). If the function runs in a reasonable amount of time, this number can be increased.

This function should not be used with very large data sets since each optimization call will take a long time. `squashData` can be used first to reduce the size of the data.

It is recommended to use `N_star = 1` when practical. Data squashing (see `squashData`) can be used to further reduce the number of data points.

Asymptotic normal confidence intervals, if requested, use standard errors calculated from the observed Fisher information matrix as discussed in DuMouchel (1999).

Value

A list containing the following elements:

- *method*: A scalar character string for the method used to find the hyperparameter estimate (possibilities are "nlminb", "nlm", and "bfgs").
- *estimates*: A named numeric vector of length 5 for the hyperparameter estimate corresponding to the smallest log-likelihood.
- *conf_int*: A data frame including the standard errors and confidence limits. Only included if conf_ints = TRUE.
- *num_close*: A scalar integer for the number of other convergent solutions that were close (within tolerance) to the chosen estimate.
- *theta_hats*: A data frame for the estimates corresponding to the initial starting points defined by theta_init. See exploreHypers.

References

DuMouchel W (1999). "Bayesian Data Mining in Large Frequency Tables, With an Application to the FDA Spontaneous Reporting System." *The American Statistician*, 53(3), 177-190.

See Also

nlminb, nlm, and optim for optimization details

squashData for data preparation

Other hyperparameter estimation functions: exploreHypers(), hyperEM()

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
#Start with 2 or more guesses
theta_init <- data.frame(
  alpha1 = c(0.5, 1),
  beta1  = c(0.5, 1),
  alpha2 = c(2,   3),
  beta2  = c(2,   3),
  p      = c(0.1, 0.2)
)
data(caers)
proc <- processRaw(caers)
squashed <- squashData(proc, bin_size = 300, keep_pts = 10)
squashed <- squashData(squashed, count = 2, bin_size = 13, keep_pts = 10)
suppressWarnings(
  hypers <- autoHyper(squashed, theta_init = theta_init)
)
print(hypers)
```

autoSquash *Automated data squashing*

Description

autoSquash squashes data by calling squashData once for each count (*N*), removing the need to repeatedly squash the same data set.

Usage

```
autoSquash(
  data,
  keep_pts = c(100, 75, 50, 25),
  cut_offs = c(500, 1000, 10000, 1e+05, 5e+05, 1e+06, 5e+06),
  num_super_pts = c(50, 75, 150, 500, 750, 1000, 2000, 5000)
)
```

Arguments

data	A data frame (typically from processRaw) containing columns named *N*, *E*, and (possibly) *weight*. Can contain additional columns, which will be ignored.
keep_pts	A vector of whole numbers for the number of points to leave unsquashed for each count (*N*). See the 'Details' section.
cut_offs	A vector of whole numbers for the cutoff values of unsquashed data used to determine how many "super points" to end up with after squashing each count (*N*). See the 'Details' section.
num_super_pts	A vector of whole numbers for the number of "super points" to end up with after squashing each count (*N*). Length must be 1 more than length of cut_offs. See the 'Details' section.

Details

See squashData for details on squashing a given count (*N*).

The elements in keep_pts determine how many points are left unsquashed for each count (*N*). The first element in keep_pts is used for the smallest *N* (usually 1). Each successive element is used for each successive *N*. Once the last element is reached, it is used for all other *N*.

For counts that are squashed, cut_offs and num_super_pts determine how the points are squashed. For instance, by default, if a given *N* contains less than 500 points to be squashed, then those points are squashed to 50 "super points".

Value

A data frame with column names *N*, *E*, and *weight* containing the reduced data set.

References

DuMouchel W, Pregibon D (2001). "Empirical Bayes Screening for Multi-item Associations." In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pp. 67-76. ACM, New York, NY, USA. ISBN 1-58113-391-X.

See Also

`processRaw` for data preparation and `squashData` for squashing individual counts

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
data(caers)
proc <- processRaw(caers)
table(proc$N)

squash1 <- autoSquash(proc)
ftable(squash1[, c("N", "weight")])

## Not run: squash2 <- autoSquash(proc, keep_pts = c(50, 5))
## Not run: ftable(squash2[, c("N", "weight")])

## Not run:
  squash3 <- autoSquash(proc, keep_pts = 100,
                        cut_offs = c(250, 500),
                        num_super_pts = c(20, 60, 125))

## End(Not run)
## Not run: ftable(squash3[, c("N", "weight")])
```

---

caers *Dietary supplement reports and products*

Description

A dataset for dietary supplement adverse event reports from 2012 containing CAERS product and adverse event reports as reported to the FDA. This particular dataset contains only products which were reported to be dietary supplements (industry code 54) reported in the year 2012. and includes 2874 unique product names and 1328 unique adverse events. There are a total of 3356 unique reports. In addition, there is also one stratification variable, indicating whether the patient is male or female

Usage

```
caers
```

Format

A data frame with 20156 rows and 4 variables:

`id` Identification number

`var1` Name of the product

`var2` Name of the symptom/event category

`strat1` Gender of the patient associated with report

Details

Further details about the data can be found using the links below.

Source

CFSAN Adverse Event Reporting System (FDA Center for Food Safety and Nutrition)

https://www.fda.gov/food/compliance-enforcement-food

https://www.fda.gov/media/97035/download

`caers_raw` *Raw CAERS data*

Description

A small subset of raw, publicly available CAERS data used to demonstrate how to prepare data for use by **openEBGM**'s functions.

Usage

```
caers_raw
```

Format

A data frame with 117 rows and 6 variables:

`RA_Report..` CAERS report identification number.

`PRI_Reported.Brand.Product.Name` The verbatim brands and/or product names indicated to have been used by the consumer reported to have experienced the adverse event.

`CI_Age.at.Adverse.Event` The age of the consumer reported to have experienced the adverse event, in units specified by `CI_Age.Unit`.

`CI_Age.Unit` The time unit (day, week, month, year) of the age provided in the `CI_Age.at.Adverse.Event` data field for the consumer reported to have experienced the adverse event.

`CI_Gender` The sex of the individual reported to have experienced the adverse event.

`SYM_One.Row.Coded.Symptoms` The symptom(s) experienced by the injured consumer as specified by the reporter and coded by FDA according to the Medical Data Dictionary for Regulatory Activities (MedDRA).

Details

The column names appear exactly as they would if you had used `read.csv()` to pull the data directly from the website below.

Further details about the data can be found using the links below.

Source

CFSAN Adverse Event Reporting System (FDA Center for Food Safety and Nutrition)

https://www.fda.gov/food/compliance-enforcement-food

https://www.fda.gov/media/97035/download

`ebgm` *Calculate EBGM scores*

Description

`ebgm` calculates the Empirical Bayes Geometric Mean (*EBGM*), which is 'the geometric mean of the empirical Bayes posterior distribution of the "true" *RR*' (DuMouchel 1999, see Eq.11). The *EBGM* is essentially a version of the relative reporting ratio (*RR*) that uses Bayesian shrinkage.

Usage

```
ebgm(theta_hat, N, E, qn, digits = 2)
```

Arguments

`theta_hat`	A numeric vector of hyperparameter estimates (likely from `autoHyper` or from directly minimizing `negLLsquash`) ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
`N`	A whole number vector of actual counts from `processRaw`.
`E`	A numeric vector of expected counts from `processRaw`.
`qn`	A numeric vector of posterior probabilities that λ came from the first component of the mixture, given $N = n$ (i.e., the mixture fraction). See function `Qn`.
`digits`	A scalar whole number that determines the number of decimal places used when rounding the results.

Details

The hyperparameter estimates (`theta_hat`) are:

- α_1, β_1: Parameter estimates of the first component of the prior distribution
- α_2, β_2: Parameter estimates of the second component
- P: Mixture fraction estimate of the prior distribution

Value

A numeric vector of EBGM scores.

References

DuMouchel W (1999). "Bayesian Data Mining in Large Frequency Tables, With an Application to the FDA Spontaneous Reporting System." *The American Statistician*, 53(3), 177-190.

See Also

autoHyper, exploreHypers, negLLsquash, negLL, negLLzero, and negLLzeroSquash for hyperparameter estimation.

processRaw for finding counts.

Qn for finding mixture fractions.

Other posterior distribution functions: Qn(), quantBisect()

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
theta_init <- data.frame(
  alpha1 = c(0.5, 1),
  beta1  = c(0.5, 1),
  alpha2 = c(2,   3),
  beta2  = c(2,   3),
  p      = c(0.1, 0.2)
)
data(caers)
proc <- processRaw(caers)
squashed <- squashData(proc, bin_size = 300, keep_pts = 10)
squashed <- squashData(squashed, count = 2, bin_size = 13, keep_pts = 10)
theta_hat <- autoHyper(data = squashed, theta_init = theta_init)$estimates
qn <- Qn(theta_hat, N = proc$N, E = proc$E)
proc$EBGM <- ebgm(theta_hat, N = proc$N, E = proc$E, qn = qn)
head(proc)
```

---

ebScores *Construct an openEBGM object*

Description

ebScores calculates EBGM scores as well as the quantiles from the posterior distribution and returns an object of class openEBGM.

Usage

```
ebScores(processed, hyper_estimate, quantiles = c(5, 95), digits = 2)
```

Arguments

processed	A data frame resulting from running processRaw.
hyper_estimate	A list resulting from running autoHyper.
quantiles	Either a numeric vector of desired quantiles to be calculated from the posterior distribution or NULL for no calculation of quantiles.
digits	A whole number scalar specifying how many decimal places to use for rounding *EBGM* and the quantiles scores.

Details

This function takes the processed data as well as the hyperparameter estimates and instantiates an object of class openEBGM. This object then contains additional calculations, such as the EBGM score, and the quantiles that are supplied by the quantiles parameter at the time of calling the function.

The function allows for the choice of an arbitrary amount of quantiles or no quantiles at all to be calculated. This may be helpful for large datasets, or when the EBGM score is the only metric of interest.

Value

An openEBGM object (class S3) containing:

- *data*: A data frame containing the results (scores, etc.).
- *hyper_parameters*: A list containing the hyperparameter estimation results.
- *quantiles*: The chosen percentiles.

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
theta_init <- data.frame(
  alpha1 = c(0.5, 1),
  beta1  = c(0.5, 1),
  alpha2 = c(2,   3),
  beta2  = c(2,   3),
  p      = c(0.1, 0.2)
)
data(caers)
proc <- processRaw(caers)
squashed <- squashData(proc, bin_size = 300, keep_pts = 10)
squashed <- squashData(squashed, count = 2, bin_size = 13, keep_pts = 10)
suppressWarnings(
  hypers <- autoHyper(data = squashed, theta_init = theta_init)
)
obj <- ebScores(processed = proc, hyper_estimate = hypers, quantiles = 5)
print(obj)
```

`exploreHypers` *Explore various hyperparameter estimates*

Description

`exploreHypers` finds hyperparameter estimates using a variety of starting points to examine the consistency of the optimization procedure.

Usage

```
exploreHypers(
  data,
  theta_init,
  squashed = TRUE,
  zeroes = FALSE,
  N_star = 1,
  method = c("nlminb", "nlm", "bfgs"),
  param_limit = 100,
  max_pts = 20000,
  std_errors = FALSE
)
```

Arguments

`data`	A data frame from `processRaw` containing columns named *N*, *E*, and (if squashed) *weight*.
`theta_init`	A data frame of initial hyperparameter guesses with columns ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
`squashed`	A scalar logical (`TRUE` or `FALSE`) indicating whether or not data squashing was used.
`zeroes`	A scalar logical specifying if zero counts are included.
`N_star`	A positive scalar whole number value for the minimum count size to be used for hyperparameter estimation. If zeroes are used, set `N_star` to `NULL`.
`method`	A scalar string indicating which optimization procedure is to be used. Choices are `"nlminb"`, `"nlm"`, or `"bfgs"`.
`param_limit`	A scalar numeric value for the largest acceptable value for the α and β estimates. Used to help protect against unreasonable/erroneous estimates.
`max_pts`	A scalar whole number for the largest number of data points allowed. Used to help prevent extremely long run times.
`std_errors`	A scalar logical indicating if standard errors should be returned for the hyperparameter estimates.

Details

The `method` argument determines which optimization procedure is used. All the options use functions from the `stats` package:

- `"nlminb"`: `nlminb`
- `"nlm"`: `nlm`
- `"bfgs"`: `optim` (*method = "BFGS"*)

Since this function runs multiple optimization procedures, it is best to start with 5 or less initial starting points (rows in `theta_init`). If the function runs in a reasonable amount of time, this number can be increased.

This function should not be used with very large data sets unless data squashing is used first since each optimization call will take a long time.

It is recommended to use `N_star = 1` when practical. Data squashing (see `squashData`) can be used to reduce the number of data points.

The *converge* column in the resulting data frame was determined by examining the convergence *code* of the chosen optimization method. In some instances, the code is somewhat ambiguous. The determination of *converge* was intended to be conservative (leaning towards FALSE when questionable). See the documentation for the chosen method for details about *code*.

Standard errors, if requested, are calculated using the observed Fisher information matrix as discussed in DuMouchel (1999).

Value

A list including the data frame `estimates` of hyperparameter estimates corresponding to the initial guesses from `theta_init` (plus convergence results):

- *code*: The convergence code returned by the chosen optimization function (see `nlminb`, `nlm`, and `optim` for details).
- *converge*: A logical indicating whether or not convergence was reached. See "Details" section for more information.
- *in_bounds*: A logical indicating whether or not the estimates were within the bounds of the parameter space (upper bound for $\alpha_1, \beta_1, \alpha_2, and \beta_2$ was determined by the `param_limit` argument).
- *minimum*: The negative log-likelihood value corresponding to the estimated optimal value of the hyperparameter.

Also returns the data frame `std_errs` if standard errors are requested.

Warning

Make sure to properly specify the `squashed`, `zeroes`, and `N_star` arguments for your data set, since these will determine the appropriate likelihood function. Also, this function will not filter out data points. For instance, if you use `N_star = 2` you must filter out the ones and zeroes (if present) from `data` prior to using this function.

References

DuMouchel W (1999). "Bayesian Data Mining in Large Frequency Tables, With an Application to the FDA Spontaneous Reporting System." *The American Statistician*, 53(3), 177-190.

See Also

`nlminb`, `nlm`, and `optim` for optimization details

`squashData` for data preparation

Other hyperparameter estimation functions: `autoHyper()`, `hyperEM()`

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
#Start with 2 or more guesses
theta_init <- data.frame(
  alpha1 = c(0.5, 1),
  beta1  = c(0.5, 1),
  alpha2 = c(2,   3),
  beta2  = c(2,   3),
  p      = c(0.1, 0.2)
)
data(caers)
proc <- processRaw(caers)
squashed <- squashData(proc, bin_size = 300, keep_pts = 10)
squashed <- squashData(squashed, count = 2, bin_size = 13, keep_pts = 10)
suppressWarnings(
  exploreHypers(squashed, theta_init = theta_init)
)
```

---

hyperEM *Estimate hyperparameters using an EM algorithm*

Description

`hyperEM` finds hyperparameter estimates using a variation on the Expectation-Maximization (EM) algorithm known as the Expectation/Conditional Maximization (ECM) algorithm (Meng et al, 1993). The algorithm estimates each element of the hyperparameter vector, θ, while holding fixed (conditioning on) the other parameters in the vector. Alternatively, it can estimate both parameters for each distribution in the mixture while holding the parameters from the other distribution and the mixing fraction fixed.

Usage

```
hyperEM(
  data,
  theta_init_vec,
  squashed = TRUE,
```

```
  zeroes = FALSE,
  N_star = 1,
  method = c("score", "nlminb"),
  profile = c("parameter", "distribution"),
  LL_tol = 1e-04,
  consecutive = 100,
  param_lower = 1e-05,
  param_upper = 20,
  print_level = 2,
  max_iter = 5000,
  conf_ints = FALSE,
  conf_level = c("95", "80", "90", "99"),
  track = FALSE
)
```

Arguments

`data`	A data frame from `processRaw` or `squashData` containing columns named *N*, *E*, and (if squashed) *weight*.
`theta_init_vec`	A numeric vector of initial hyperparameter guesses ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
`squashed`	A scalar logical (`TRUE` or `FALSE`) indicating whether or not data squashing was used.
`zeroes`	A scalar logical specifying if zero counts are included.
`N_star`	A positive scalar whole number value for the minimum count size to be used for hyperparameter estimation. If zeroes are used, set `N_star` to `NULL`.
`method`	A scalar string indicating which method (i.e. score functions or log-likelihood function) to use for the maximization steps. Possible values are `"score"` and `"nlminb"`.
`profile`	A scalar string indicating which method to use to optimize the log-likelihood function if `method = "nlminb"` (ignored if `method = "score"`). `profile = "parameter"` optimizes one parameter (α or β) from the log-likelihood function at a time. `profile = "distribution"` optimizes one distribution from the mixture at a time (α and β simultaneously).
`LL_tol`	A scalar numeric value for the tolerance used for determining when the change in log-likelihood at each iteration is sufficiently small. Used for convergence assessment.
`consecutive`	A positive scalar whole number value for the number of consecutive iterations the change in log-likelihood must be below `LL_tol` in order to reach convergence. Larger values reduce the chance of getting stuck in a flat region of the curve.
`param_lower`	A scalar numeric value for the smallest acceptable value for each α and β estimate.
`param_upper`	A scalar numeric value for the largest acceptable value for each α and β estimate.

print_level	A value that determines how much information is printed during execution. Possible value are 0 for no printing, 1 for minimal information, and 2 for maximal information.
max_iter	A positive scalar whole number value for the maximum number of iterations to use.
conf_ints	A scalar logical indicating if confidence intervals and standard errors should be returned.
conf_level	A scalar string for the confidence level used if confidence intervals are requested.
track	A scalar logical indicating whether or not to retain the hyperparameter estimates and log-likelihood value at each iteration. Can be used for plotting to better understand the algorithm's behavior.

Details

If method = "score", the maximization step finds a root of the score function. If method = "nlminb", nlminb is used to find a minimum of the negative log-likelihood function.

If method = "score" and zeroes = FALSE, then 'N_star' must equal 1.

If method = "score", the model is reparameterized. The parameters are transformed to force the parameter space to include all real numbers. This approach addresses numerical issues at the edge of the parameter space. The reparameterization follows: $\alpha_{prime} = log(\alpha)$, $\beta_{prime} = log(\beta)$, and $P_{prime} = tan(pi * P - pi/2)$. However, the values returned in estimates are on the original scale (back-transformed).

On every 100th iteration, the procedure described in Millar (2011) is used to accelerate the estimate of θ.

The score vector and its Euclidean norm should be close to zero at a local maximum and can therefore be used to help assess the reliability of the results. A local maximum might not be the global MLE, however.

Asymptotic normal confidence intervals, if requested, use standard errors calculated from the observed Fisher information matrix as discussed in DuMouchel (1999).

Value

A list including the following:

- *estimates*: The maximum likelihood estimate (MLE) of the hyperparameter, θ.
- *conf_int*: A data frame including the standard errors and confidence limits for estimates. Only included if conf_ints = TRUE.
- *maximum*: The log-likelihood function evaluated at estimates.
- *method*: The method used in the maximization step.
- *elapsed*: The elapsed function execution time in seconds.
- *iters*: The number of iterations used.
- *score*: The score functions (i.e. score vector) evaluated at estimates. All elements should be close to zero.
- *score_norm*: The Euclidean norm of the score vector evaluated at estimates. Should be close to zero.

- *tracking*: The estimates of θ at each iteration and the log-likelihood function evaluated at those estimates. Unless `track = TRUE`, only shows the starting point of the algorithm.

References

DuMouchel W (1999). "Bayesian Data Mining in Large Frequency Tables, With an Application to the FDA Spontaneous Reporting System." *The American Statistician*, 53(3), 177-190.

Meng X-L, Rubin D (1993). "Maximum likelihood estimation via the ECM algorithm: A general framework", *Biometrika*, 80(2), 267-278.

Millar, Russell B (2011). "Maximum Likelihood Estimation and Inference", *John Wiley & Sons, Ltd*, 111-112.

See Also

`uniroot` for finding a zero of the score function and `nlminb` for minimizing the negative log-likelihood function

Other hyperparameter estimation functions: `autoHyper()`, `exploreHypers()`

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
data(caers)
proc <- processRaw(caers)
squashed <- squashData(proc, bin_size = 300, keep_pts = 10)
squashed <- squashData(squashed, count = 2, bin_size = 13, keep_pts = 10)
hypers <- hyperEM(squashed, theta_init_vec = c(1, 1, 2, 2, .3),
                  consecutive = 10, LL_tol = 1e-03)
```

`negLL` *Likelihood without zero counts*

Description

`negLL` computes the negative log-likelihood based on the conditional marginal distribution of the counts, N, given that $N >= N^*$, where N^* is the smallest count used for estimating the hyperparameters (DuMouchel et al. 2001). This function is minimized to estimate the hyperparameters of the prior distribution. Use this function when neither zero counts nor data squashing are being used. Generally this function is not recommended unless using a small data set since data squashing (see `squashData` and `negLLsquash`) can increase efficiency (DuMouchel et al. 2001).

Usage

```
negLL(theta, N, E, N_star = 1)
```

Arguments

theta	A numeric vector of hyperparameters ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
N	A whole number vector of actual counts from processRaw.
E	A numeric vector of expected counts from processRaw.
N_star	A scalar whole number for the minimum count size used.

Details

The conditional marginal distribution for the counts, *N*, given that *N* >= *N**, is based on a mixture of two negative binomial distributions. The hyperparameters for the prior distribution (mixture of gammas) are estimated by optimizing the likelihood equation from this conditional marginal distribution. It is recommended to use `N_star = 1` when practical.

The hyperparameters are:

- α_1, β_1: Parameters of the first component of the marginal distribution of the counts (also the prior distribution)
- α_2, β_2: Parameters of the second component
- P: Mixture fraction

This function will not need to be called directly if using exploreHypers or autoHyper.

Value

A scalar negative log-likelihood value

Warnings

Make sure *N_star* matches the smallest actual count in *N* before using this function. Filter *N* and *E* if needed.

Make sure the data were not squashed before using this function.

References

DuMouchel W, Pregibon D (2001). "Empirical Bayes Screening for Multi-item Associations." In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pp. 67-76. ACM, New York, NY, USA. ISBN 1-58113-391-X.

See Also

nlm, nlminb, and optim for optimization

Other negative log-likelihood functions: negLLsquash(), negLLzeroSquash(), negLLzero()

`negLLsquash` *Likelihood with data squashing and no zero counts*

Description

`negLLsquash` computes the negative log-likelihood based on the conditional marginal distribution of the counts, *N*, given that *N* >= *N**, where *N** is the smallest count used for estimating the hyperparameters. This function is minimized to estimate the hyperparameters of the prior distribution. Use this function when zero counts are not used and data squashing is used as described by DuMouchel et al. (2001). This function is the likelihood function that should usually be chosen.

Usage

```
negLLsquash(theta, ni, ei, wi, N_star = 1)
```

Arguments

`theta`	A numeric vector of hyperparameters ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
`ni`	A whole number vector of squashed actual counts from `squashData`.
`ei`	A numeric vector of squashed expected counts from `squashData`.
`wi`	A whole number vector of bin weights from `squashData`.
`N_star`	A scalar whole number for the minimum count size used.

Details

The conditional marginal distribution for the counts, *N*, given that *N* >= *N**, is based on a mixture of two negative binomial distributions. The hyperparameters for the prior distribution (mixture of gammas) are estimated by optimizing the likelihood equation from this conditional marginal distribution. It is recommended to use `N_star = 1` when practical.

The hyperparameters are:

- α_1, β_1: Parameters of the first component of the marginal distribution of the counts (also the prior distribution)
- α_2, β_2: Parameters of the second component
- P: Mixture fraction

This function will not need to be called directly if using `exploreHypers` or `autoHyper`.

Value

A scalar negative log-likelihood value

Warnings

Make sure *N_star* matches the smallest actual count in *ni* before using this function. Filter *ni*, *ei*, and *wi* if needed.

Make sure the data were actually squashed (see `squashData`) before using this function.

References

DuMouchel W, Pregibon D (2001). "Empirical Bayes Screening for Multi-item Associations." In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pp. 67-76. ACM, New York, NY, USA. ISBN 1-58113-391-X.

See Also

`nlm`, `nlminb`, and `optim` for optimization and `squashData` for data squashing

Other negative log-likelihood functions: `negLLzeroSquash()`, `negLLzero()`, `negLL()`

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
theta_init <- c(1, 1, 3, 3, .2)  #initial guess
data(caers)
proc <- processRaw(caers)
squashed <- squashData(proc, bin_size = 300, keep_pts = 10)
squashed <- squashData(squashed, count = 2, bin_size = 13, keep_pts = 10)
negLLsquash(theta = theta_init, ni = squashed$N, ei = squashed$E,
            wi = squashed$weight)
#For hyperparameter estimation...
stats::nlminb(start = theta_init, objective = negLLsquash, ni = squashed$N,
              ei = squashed$E, wi = squashed$weight)
```

`negLLzero` *Likelihood with zero counts*

Description

`negLLzero` computes the negative log-likelihood based on the unconditional marginal distribution of N (equation 12 in DuMouchel 1999, except taking negative natural log). This function is minimized to estimate the hyperparameters of the prior distribution. Use this function if including zero counts but not squashing data. Generally this function is not recommended (`negLLsquash` is typically more efficient).

Usage

```
negLLzero(theta, N, E)
```

Arguments

`theta`	A numeric vector of hyperparameters ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
`N`	A whole number vector of actual counts from `processRaw`.
`E`	A numeric vector of expected counts from `processRaw`.

Details

The marginal distribution of the counts, *N*, is a mixture of two negative binomial distributions. The hyperparameters for the prior distribution (mixture of gammas) are estimated by optimizing the likelihood equation from this marginal distribution.

The hyperparameters are:

- α_1, β_1: Parameters of the first component of the marginal distribution of the counts (also the prior distribution)
- α_2, β_2: Parameters of the second component
- P: Mixture fraction

This function will not need to be called directly if using `exploreHypers` or `autoHyper`.

Value

A scalar negative log-likelihood value.

Warnings

Make sure *N* actually contains zeroes before using this function. You should have used the `zeroes = TRUE` option when calling the `processRaw` function.

Make sure the data were not squashed before using this function.

References

DuMouchel W (1999). "Bayesian Data Mining in Large Frequency Tables, With an Application to the FDA Spontaneous Reporting System." *The American Statistician*, 53(3), 177-190.

See Also

`nlm`, `nlminb`, and `optim` for optimization

Other negative log-likelihood functions: `negLLsquash()`, `negLLzeroSquash()`, `negLL()`

`negLLzeroSquash` *Likelihood with data squashing & zero counts*

Description

`negLLzeroSquash` computes the negative log-likelihood based on the unconditional marginal distribution of *N* (DuMouchel et al. 2001). This function is minimized to estimate the hyperparameters of the prior distribution. Use this function if including zero counts and using data squashing. Generally this function is not recommended unless convergence issues occur without zero counts (`negLLsquash` is typically more efficient).

Usage

```
negLLzeroSquash(theta, ni, ei, wi)
```

Arguments

theta	A numeric vector of hyperparameters ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
ni	A whole number vector of squashed actual counts from squashData.
ei	A numeric vector of squashed expected counts from squashData.
wi	A whole number vector of bin weights from squashData.

Details

The marginal distribution of the counts, *N*, is a mixture of two negative binomial distributions. The hyperparameters for the prior distribution (mixture of gammas) are estimated by optimizing the likelihood equation from this marginal distribution.

The hyperparameters are:

- α_1, β_1: Parameters of the first component of the marginal distribution of the counts (also the prior distribution)
- α_2, β_2: Parameters of the second component
- P: Mixture fraction

This function will not need to be called directly if using exploreHypers or autoHyper.

Value

A scalar negative log-likelihood value.

Warnings

Make sure *ni* actually contains zeroes before using this function. You should have used the zeroes = TRUE option when calling the processRaw function.

Make sure the data were actually squashed (see squashData) before using this function.

References

DuMouchel W, Pregibon D (2001). "Empirical Bayes Screening for Multi-item Associations." In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pp. 67-76. ACM, New York, NY, USA. ISBN 1-58113-391-X.

See Also

nlm, nlminb, and optim for optimization and squashData for data squashing

Other negative log-likelihood functions: negLLsquash(), negLLzero(), negLL()

`plot.openEBGM` *Plot an openEBGM object*

Description

Plot an openEBGM object

Usage

```
## S3 method for class 'openEBGM'
plot(x, y = NULL, event = NULL, plot.type = "bar", ...)
```

Arguments

`x`	An openEBGM object constructed by `ebScores()`
`y`	Unused parameter to satisfy generic function requirement
`event`	An (optional) specification of an event to subset the data by.
`plot.type`	A character vector specifying which type of plot should be output. See details.
`...`	Arguments to be passed to methods

Details

There are three different types of plots that the plot function may produce when called on an openEBGM object. These are

- bar
- shrinkage
- histogram

A bar chart displays the top ten product-symptom EBGM scores, as well as error bars which display the highest and lowest of the quantiles chosen at the time of instantiating the openEBGM object. A shrinkage plot plots EBGM score on the y axis, and the natural log of the RR on the x axis. This plot is also called a squid plot and was conceived by Stuart Chirtel. Finally, a histogram simply displays a histogram of the EBGM scores.

print.openEBGM *Print an openEBGM object*

Description

Print an openEBGM object

Usage

```
## S3 method for class 'openEBGM'
print(x, threshold = 2, ...)
```

Arguments

x	An openEBGM object constructed by ebScores()
threshold	A numeric value indicating the minimum threshold for QUANT or EBGM values to display.
...	Arguments to be passed to other methods

processRaw *Process raw data*

Description

processRaw finds the actual and expected counts using the methodology described by DuMouchel (1999); however, an adjustment is applied to expected counts to prevent double-counting (i.e., using unique marginal ID counts instead of contingency table marginal totals). Also calculates the relative reporting ratio (*RR*) and the proportional reporting ratio (*PRR*).

Usage

```
processRaw(
  data,
  stratify = FALSE,
  zeroes = FALSE,
  digits = 2,
  max_cats = 10,
  list_ids = FALSE
)
```

Arguments

`data`	A data frame containing columns named: *id*, *var1*, and *var2*. Possibly includes columns for stratification variables identified by the substring *'strat'* (e.g. *strat_age*). Other columns will be ignored.
`stratify`	A logical scalar (TRUE or FALSE) specifying if stratification is to be used.
`zeroes`	A logical scalar specifying if zero counts should be included. Using zero counts is only recommended for small data sets because it will dramatically increase run time.
`digits`	A whole number scalar specifying how many decimal places to use for rounding *RR* and *PRR*.
`max_cats`	A whole number scalar specifying the maximum number of categories to allow in any given stratification variable. Used to help prevent a situation where the user forgets to categorize a continuous variable, such as age.
`list_ids`	A logical scalar specifying if a column for pipe-concatenated IDs should be returned.

Details

An *id* column must be included in `data`. If your data set does not include IDs, make a column of unique IDs using `df$id <- 1:nrow(df)`. However, unique IDs should only be constructed if the cells in the contingency table are mutually exclusive. For instance, unique IDs for each row in `data` are not appropriate with CAERS data since a given report can include multiple products and/or adverse events.

Stratification variables are identified by searching for the substring *'strat'*. Only variables containing *'strat'* (case sensitive) will be used as stratification variables. *PRR* calculations ignore stratification, but *E* and *RR* calculations are affected. A warning will be displayed if any stratum contains less than 50 unique IDs.

If a *PRR* calculation results in division by zero, `Inf` is returned.

Value

A data frame with actual counts (*N*), expected counts (*E*), relative reporting ratio (*RR*), and proportional reporting ratio (*PRR*) for *var1*-*var2* pairs. Also includes a column for IDs (*ids*) if `list_ids = TRUE`.

Warnings

Use of the `zeroes = TRUE` option will result in a considerable increase in runtime. Using zero counts is not recommended if the contingency table is moderate or large in size (~500K cells or larger). However, using zeroes could be useful if the optimization algorithm fails to converge when estimating hyperparameters.

Any columns in `data` containing the substring *'strat'* in the column name will be considered stratification variables, so verify that you do not have any extraneous columns with that substring.

References

DuMouchel W (1999). "Bayesian Data Mining in Large Frequency Tables, With an Application to the FDA Spontaneous Reporting System." *The American Statistician*, 53(3), 177-190.

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
var1 <- c("product_A", rep("product_B", 3), "product_C",
          rep("product_A", 2), rep("product_B", 2), "product_C")
var2 <- c("event_1", rep("event_2", 2), rep("event_3", 2),
          "event_2", rep("event_3", 3), "event_1")
strat1 <- c(rep("Male", 5), rep("Female", 3), rep("Male", 2))
strat2 <- c(rep("age_cat1", 5), rep("age_cat1", 3), rep("age_cat2", 2))
dat <- data.frame(
  var1 = var1, var2 = var2, strat1 = strat1, strat2 = strat2,
  stringsAsFactors = FALSE
)
dat$id <- 1:nrow(dat)
processRaw(dat)
suppressWarnings(
  processRaw(dat, stratify = TRUE)
)
processRaw(dat, zeroes = TRUE)
suppressWarnings(
  processRaw(dat, stratify = TRUE, zeroes = TRUE)
)
processRaw(dat, list_ids = TRUE)
```

Qn *Calculate Qn*

Description

Qn calculates Q_n, the posterior probability that λ came from the first component of the mixture, given $N = n$ (Eq. 6, DuMouchel 1999). Q_n is the mixture fraction for the posterior distribution.

Usage

```
Qn(theta_hat, N, E)
```

Arguments

theta_hat	A numeric vector of hyperparameter estimates (likely from autoHyper or from directly minimizing negLLsquash) ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
N	A whole number vector of actual counts from processRaw.
E	A numeric vector of expected counts from processRaw.

Details

The hyperparameter estimates (`theta_hat`) are:

- α_1, β_1: Parameter estimates of the first component of the prior distribution
- α_2, β_2: Parameter estimates of the second component
- P: Mixture fraction estimate of the prior distribution

Value

A numeric vector of probabilities.

References

DuMouchel W (1999). "Bayesian Data Mining in Large Frequency Tables, With an Application to the FDA Spontaneous Reporting System." *The American Statistician*, 53(3), 177-190.

See Also

`autoHyper`, `exploreHypers`, `negLLsquash`, `negLL`, `negLLzero`, and `negLLzeroSquash` for hyperparameter estimation.

`processRaw` for finding counts.

Other posterior distribution functions: `ebgm()`, `quantBisect()`

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
theta_init <- data.frame(
  alpha1 = c(0.5, 1),
  beta1  = c(0.5, 1),
  alpha2 = c(2,   3),
  beta2  = c(2,   3),
  p      = c(0.1, 0.2)
)
data(caers)
proc <- processRaw(caers)
squashed <- squashData(proc, bin_size = 300, keep_pts = 10)
squashed <- squashData(squashed, count = 2, bin_size = 13, keep_pts = 10)
theta_hat <- autoHyper(data = squashed, theta_init = theta_init)$estimates
qn <- Qn(theta_hat, N = proc$N, E = proc$E)
head(qn)
```

`quantBisect` *Find quantiles of the posterior distribution*

Description

`quantBisect` finds the desired quantile of the posterior distribution using the bisection method. Used to create credibility limits.

Usage

```
quantBisect(
  percent,
  theta_hat,
  N,
  E,
  qn,
  digits = 2,
  limits = c(-1e+05, 1e+05),
  max_iter = 2000
)
```

Arguments

`percent`	A numeric scalar between 1 and 99 for the desired percentile (e.g., 5 for 5th percentile).
`theta_hat`	A numeric vector of hyperparameter estimates (likely from `autoHyper` or from directly minimizing `negLLsquash`) ordered as: $\alpha_1, \beta_1, \alpha_2, \beta_2, P$.
`N`	A whole number vector of actual counts from `processRaw`.
`E`	A numeric vector of expected counts from `processRaw`.
`qn`	A numeric vector of posterior probabilities that λ came from the first component of the mixture, given $N = n$ (i.e., the mixture fraction). See function `Qn`.
`digits`	A scalar whole number that determines the number of decimal places used when rounding the results.
`limits`	A whole number vector of length 2 for the upper and lower bounds of the search space.
`max_iter`	A whole number scalar for the maximum number of iterations. Used to prevent infinite loops.

Details

The hyperparameter estimates (`theta_hat`) are:

- α_1, β_1: Parameter estimates of the first component of the prior distribution
- α_2, β_2: Parameter estimates of the second component
- P: Mixture fraction estimate of the prior distribution

Although this function can find any quantile of the posterior distribution, it will often be used to calculate the 5th and 95th percentiles to create a 90% credibility interval.

The quantile is calculated by solving for x in the general equation $F(x) = cutoff$, or equivalently, $F(x) - cutoff = 0$, where $F(x)$ is the cumulative distribution function of the posterior distribution and $cutoff$ is the appropriate cutoff level (e.g., 0.05 for the 5th percentile).

Value

A numeric vector of quantile estimates.

Warning

The `digits` argument determines the tolerance for the bisection algorithm. The more decimal places you want returned, the longer the run time.

See Also

https://en.wikipedia.org/wiki/Bisection_method

`autoHyper`, `exploreHypers`, `negLLsquash`, `negLL`, `negLLzero`, and `negLLzeroSquash` for hyperparameter estimation.

`processRaw` for finding counts.

`Qn` for finding mixture fractions.

Other posterior distribution functions: `Qn()`, `ebgm()`

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
theta_init <- data.frame(
  alpha1 = c(0.5, 1),
  beta1  = c(0.5, 1),
  alpha2 = c(2,   3),
  beta2  = c(2,   3),
  p      = c(0.1, 0.2)
)
data(caers)
proc <- processRaw(caers)
squashed <- squashData(proc, bin_size = 300, keep_pts = 10)
squashed <- squashData(squashed, count = 2, bin_size = 13, keep_pts = 10)
theta_hat <- autoHyper(data = squashed, theta_init = theta_init)$estimates
qn <- Qn(theta_hat, N = proc$N, E = proc$E)
proc$QUANT_05 <- quantBisect(percent = 5, theta = theta_hat, N = proc$N,
                             E = proc$E, qn = qn)
## Not run: proc$QUANT_95 <- quantBisect(percent = 95, theta = theta_hat,
                                       N = proc$N, E = proc$E, qn = qn)
## End(Not run)
head(proc)
```

squashData *Squash data for hyperparameter estimation*

Description

squashData squashes data by binning expected counts, *E*, for a given actual count, *N*, using bin means as the expected counts for the reduced data set. The squashed points are weighted by bin size. Data can be squashed to reduce computational burden (see DuMouchel et al., 2001) when estimating the hyperparameters.

Usage

```
squashData(
  data,
  count = 1,
  bin_size = 50,
  keep_pts = 100,
  min_bin = 50,
  min_pts = 500
)
```

Arguments

data	A data frame (typically from processRaw or a previous call to squashData) containing columns named *N*, *E*, and (possibly) *weight*. Can contain additional columns, which will be ignored.
count	A non-negative scalar whole number for the count size, *N*, used for binning
bin_size	A scalar whole number (>= 2)
keep_pts	A nonnegative scalar whole number for number of points with the largest expected counts to leave unsquashed. Used to help prevent "oversquashing".
min_bin	A positive scalar whole number for the minimum number of bins needed. Used to help prevent "oversquashing".
min_pts	A positive scalar whole number for the minimum number of original (unsquashed) points needed for squashing. Used to help prevent "oversquashing".

Details

Can be used iteratively (count = 1, then 2, etc.).

The *N* column in data will be coerced using as.integer, and *E* will be coerced using as.numeric. Missing data are not allowed.

Since the distribution of expected counts, *E*, tends to be skewed to the right, the largest *E*s are not squashed by default. This behavior can be changed by setting the keep_pts argument to zero (0); however, this is not recommended. Squashing the largest *E*s could result in a large loss of information, so it is recommended to use a value of 100 or more for keep_pts.

Values for keep_pts, min_bin, and min_pts should typically be at least as large as the default values.

Value

A data frame with column names *N*, *E*, and *weight* containing the reduced data set.

References

DuMouchel W, Pregibon D (2001). "Empirical Bayes Screening for Multi-item Associations." In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pp. 67-76. ACM, New York, NY, USA. ISBN 1-58113-391-X.

See Also

`processRaw` for data preparation and `autoSquash` for automatically squashing an entire data set in one function call

Examples

```
set.seed(483726)
dat <- data.frame(
  var1 = letters[1:26], var2 = LETTERS[1:26],
  N = c(rep(0, 11), rep(1, 10), rep(2, 4), rep(3, 1)),
  E = round(abs(c(rnorm(11, 0), rnorm(10, 1), rnorm(4, 2), rnorm(1, 3))), 3),
  stringsAsFactors = FALSE
)
(zeroes <- squashData(dat, count = 0, bin_size = 3, keep_pts = 1,
                      min_bin = 2, min_pts = 2))
(ones <- squashData(zeroes, bin_size = 2, keep_pts = 1,
                    min_bin = 2, min_pts = 2))
(twos <- squashData(ones, count = 2, bin_size = 2, keep_pts = 1,
                    min_bin = 2, min_pts = 2))

squashData(zeroes, bin_size = 2, keep_pts = 0,
           min_bin = 2, min_pts = 2)
squashData(zeroes, bin_size = 2, keep_pts = 1,
           min_bin = 2, min_pts = 2)
squashData(zeroes, bin_size = 2, keep_pts = 2,
           min_bin = 2, min_pts = 2)
squashData(zeroes, bin_size = 2, keep_pts = 3,
           min_bin = 2, min_pts = 2)
```

`summary.openEBGM` *Summarize an openEBGM object*

Description

Summarize an openEBGM object

Usage

```
## S3 method for class 'openEBGM'
summary(object, plot.out = TRUE, log.trans = FALSE, ...)
```

Arguments

object	An openEBGM object constructed by ebScores
plot.out	A logical value indicating whether or not a histogram of the EBGM scores should be displayed
log.trans	A logical value indicating whether or not the data should be log-transformed.
...	Additional arguments affecting the summary produced

Details

This function provides a brief summary of the results of the calculations performed in the ebScores function. In particular, it provides the numerical summary of the EBGM and QUANT_* vectors.

Additionally, calling summary on an openEBGM object will produce a histogram of the EBGM scores. By setting the log.trans parameter to TRUE, one can convert the EBGM score to EBlog2, which is a Bayesian version of the information criterion (DuMouchel).

References

DuMouchel W (1999). "Bayesian Data Mining in Large Frequency Tables, With an Application to the FDA Spontaneous Reporting System." *The American Statistician*, 53(3), 177-190.

Examples

```
data.table::setDTthreads(2)  #only needed for CRAN checks
theta_init <- data.frame(
  alpha1 = c(0.5, 1),
  beta1  = c(0.5, 1),
  alpha2 = c(2,   3),
  beta2  = c(2,   3),
  p      = c(0.1, 0.2)
)
data(caers)
proc <- processRaw(caers)
squashed <- squashData(proc, bin_size = 300, keep_pts = 10)
squashed <- squashData(squashed, count = 2, bin_size = 13, keep_pts = 10)
suppressWarnings(
  hypers <- autoHyper(data = squashed, theta_init = theta_init)
)
ebout <- ebScores(processed = proc, hyper_estimate = hypers, quantiles = 5)
summary(ebout)
## Not run: summary(ebout, plot.out = FALSE)
## Not run: summary(ebout, log.trans = TRUE)
```

Index