Package ‘mlfit’

October 13, 2022

Type Package

Title Iterative Proportional Fitting Algorithms for Nested Structures

Version 0.5.3

Date 2021-10-08

Description The Iterative Proportional Fitting (IPF) algorithm operates on count data.
This package offers implementations for several algorithms that extend this to
nested structures: 'parent' and 'child' items for both of which constraints can be provided.
The fitting algorithms include Iterative Proportional Updating <https://trid.trb.org/view/881554>,
Hierarchical IPF <doi:10.3929/ethz-a-006620748>, Entropy Optimization <https://trid.trb.org/view/881144>,
and Generalized Raking <doi:10.2307/2290793>. Additionally, a number of replication methods
is also provided such as 'Truncate, replicate, sample' <doi:10.1016/j.compenvurbsys.2013.03.004>.

License GPL (>= 3)

URL https://mlfit.github.io/mlfit/, https://github.com/mlfit/mlfit

BugReports https://github.com/mlfit/mlfit/issues

Depends methods

Imports BB, dplyr, hms, kimisc, Matrix, plyr, tibble, forcats, rlang,
utils, wrswoR, lifecycle

Suggests covr, testthat (>= 3.0.0), MASS, sampling, XML, waldo

Encoding UTF-8

RoxygenNote 7.1.2

Config/testthat/edition 3

NeedsCompilation no

Author Kirill Müller [aut, cph] (Creator of the package),
Kay W. Axhausen [ths] (Advisor of Kirill Müller),
Amarin Siripanich [aut, cre] (Contributed `ml_replicate()`),
Taha H. Rashidi [ths] (Advisor of Amarin Siripanich)

Maintainer Amarin Siripanich <amarin.siri@gmail.com>

Repository CRAN

Date/Publication 2021-10-08 06:50:02 UTC

R topics documented:

`mlfit-package` *mlfit: Iterative Proportional Fitting Algorithms for Nested Structures*

Description

The Iterative Proportional Fitting (IPF) algorithm operates on count data. This package offers implementations for several algorithms that extend this to nested structures: 'parent' and 'child' items for both of which constraints can be provided. The fitting algorithms include Iterative Proportional Updating <https://trid.trb.org/view/881554>, Hierarchical IPF <doi:10.3929/ethz-a-006620748>, Entropy Optimization <https://trid.trb.org/view/881144>, and Generalized Raking <doi:10.2307/2290793>. Additionally, a number of replication methods is also provided such as 'Truncate, replicate, sample' <doi:10.1016/j.compenvurbsys.2013.03.004>.

Details

To use this package, you need to:

1. Specify your fitting problem with `ml_problem()`
2. Optionally, convert the fitting problem to a structure that can be processed by the algorithms with `flatten_ml_fit_problem()`; this is helpful if you want to run the same fitting problem with multiple algorithms and compare results.
3. Compute weights with one of the algorithms provided in this package with `ml_fit()` or one of the specialized functions
4. Analyze weights or residuals, e.g. with `compute_margins()`

Author(s)

Maintainer: Amarin Siripanich <`amarin.siri@gmail.com`> (Contributed 'ml_replicate()')

Authors:

- Kirill Müller (Creator of the package) [copyright holder]

Other contributors:

- Kay W. Axhausen (Advisor of Kirill Müller) [thesis advisor]
- Taha H. Rashidi (Advisor of Amarin Siripanich) [thesis advisor]

See Also

Useful links:

- https://mlfit.github.io/mlfit/
- https://github.com/mlfit/mlfit
- Report bugs at https://github.com/mlfit/mlfit/issues

`compute_margins` *Compute margins for a weighting of a multi-level fitting problem*

Description

These functions allows checking a fit in terms of the original input data.

Usage

```
compute_margins(ml_problem, weights, verbose = FALSE)

margin_to_df(controls, count = NULL, verbose = FALSE)
```

Arguments

`ml_problem`	A fitting problem created by `ml_problem()` or returned by `flatten_ml_fit_problem()`.
`weights`	A vector with one entry per row of the original reference sample
`verbose`	If `TRUE`, print diagnostic output.
`controls`	Margins as returned by `compute_margins` or as passed to the `controls` parameter of `ml_problem()`.
`count`	Name of control total column, autodetected by default.

Details

`compute_margins()` computes margins in the format used for the input controls (i.e., as expected by the `controls` parameter of the `ml_problem()` function), based on a reference sample and a weight vector.

`margins_to_df()` converts margins to a data frame for easier comparison.

Value

`compute_margins()` returns a named list with two components, `individual` and `group`. Each contains a list of margins as `data.frames`.

`margins_to_df()` returns a data frame with the following columns:

`..control.type..` Type of the control total: either `individual` or `group`.

`..control.name..` Name of the control total, if exists.

`..id..` Name of the control category.

`..count..` Count of the control category.

See Also

`ml_fit()`

Examples

```
path <- toy_example("Tiny")
problem <- readRDS(path)
fit <- ml_fit(ml_problem = problem, algorithm = "entropy_o")
margins <- compute_margins(problem, fit$weights)
margins

margin_to_df(problem$controls)
margin_to_df(margins)
```

`dss` *Calibrate sample weights*

Description

Calibrate sample weights according to known marginal population totals. Based on initial sample weights, the so-called *g*-weights are computed by generalized raking procedures. The final sample weights need to be computed by multiplying the resulting *g*-weights with the initial sample weights.

Usage

```
dss(
  X,
  d,
  totals,
  q = NULL,
  method = c("raking", "linear", "logit"),
  bounds = NULL,
  maxit = 500,
  ginv = gginv(),
  tol = 1e-06,
  attributes = FALSE
)
```

Arguments

`X`	a matrix of calibration variables.
`d`	a numeric vector giving the initial sample (or design) weights.
`totals`	a numeric vector of population totals corresponding to the calibration variables in `X`.
`q`	a numeric vector of positive values accounting for heteroscedasticity. Small values reduce the variation of the *g*-weights.

`method`	a character string specifying the calibration method to be used. Possible values are `"linear"` for the linear method, `"raking"` for the multiplicative method known as raking and `"logit"` for the logit method.
`bounds`	a numeric vector of length two giving bounds for the g-weights to be used in the logit method. The first value gives the lower bound (which must be smaller than or equal to 1) and the second value gives the upper bound (which must be larger than or equal to 1). If `NULL`, the bounds are set to `c(0, 10)`.
`maxit`	a numeric value giving the maximum number of iterations.
`ginv`	a function that computes the Moore-Penrose generalized inverse (default: an optimized version of `MASS::ginv()`). In some cases it is possible to speed up the process by using a function that computes a "regular" matrix inverse such as `{solve.default}`.
`tol`	relative tolerance; convergence is achieved if the difference of all residuals (relative to the corresponding total) is smaller than this tolerance.
`attributes`	should additional attributes (currently `success`, `iterations`, `method` and `bounds`) be added to the result? If `FALSE` (default), a warning is given if convergence within the given relative tolerance could not be achieved.

Value

A numeric vector containing the *g*-weights.

Note

This is a faster implementation of parts of `sampling::calib()` from package `sampling`. Note that the default calibration method is raking and that the truncated linear method is not yet implemented.

Author(s)

Andreas Alfons, with improvements by Kirill Müller

References

Deville, J.-C. and Särndal, C.-E. (1992) Calibration estimators in survey sampling. *Journal of the American Statistical Association*, **87**(418), 376–382.

Deville, J.-C., Särndal, C.-E. and Sautory, O. (1993) Generalized raking procedures in survey sampling. *Journal of the American Statistical Association*, **88**(423), 1013–1020.

Examples

```
obs <- 1000
vars <- 100
Xs <- matrix(runif(obs * vars), nrow = obs)
d <- runif(obs) / obs
totals <- rep(1, vars)
g <- dss(Xs, d, totals, method = "linear", ginv = solve)
g2 <- dss(Xs, d, totals, method = "raking")
```

flatten_ml_fit_problem

Return a flattened representation of a multi-level fitting problem instance

Description

This function transforms a multi-level fitting problem to a representation more suitable for applying the algorithms: A matrix with one row per controlled attribute and one column per household, a weight vector with one weight per household, and a control vector.

Usage

```
flatten_ml_fit_problem(
  ml_problem,
  model_matrix_type = c("combined", "separate"),
  verbose = FALSE
)

as_flat_ml_fit_problem(x, model_matrix_type = c("combined", "separate"), ...)
```

Arguments

`ml_problem`	A fitting problem created by `ml_problem()` or returned by `flatten_ml_fit_problem()`.
`model_matrix_type`	Which model matrix building strategy to use? See details.
`verbose`	If `TRUE`, print diagnostic output.
`x`	An object
`...`	Further parameters passed to the algorithm

Details

The standard way to build a model matrix (`model_matrix = "combined"`) is to include intercepts and avoid repeating redundant attributes. A simpler model matrix specification, available via `model_matrix = "separate"`, is suggested by Ye et al. (2009) and required for the `ml_fit_ipu()` implementation: Here, simply one column per target value is used, which results in a larger model matrix if more than one control is given.

Value

An object of classes `flat_ml_fit_problem`, essentially a named list.

See Also

`ml_fit()`

Examples

```
path <- toy_example("Tiny")
flat_problem <- flatten_ml_fit_problem(ml_problem = readRDS(path))
flat_problem

fit <- ml_fit_dss(flat_problem)
fit$flat_weights
fit$weights
```

`gginv` *Generalized Inverse of a Matrix using a custom tolerance or SVD implementation*

Description

The `gginv` function creates a function that calculates the Moore-Penrose generalized inverse of a matrix X using a fixed tolerance value and a custom implementation for computing the singular value decomposition.

Usage

```
gginv(tol = sqrt(.Machine$double.eps), svd = base::svd)
```

Arguments

`tol`	A relative tolerance to detect zero singular values.
`svd`	A function that computes the singular value decomposition of a matrix

Details

The `svd` argument is expected to adhere to the interface of `base::svd()`. It will be called as `svd(x)` (with the `nu` and `nv` arguments unset) and is expected to return a named list with components `d`, `u` and `v`.

Value

A function that accepts one argument `X` that computes a MP generalized inverse matrix for it.

Author(s)

Adapted implementation from the `MASS` package.

See Also

`MASS::ginv()`, `base::svd()`

ml_fit *Estimate weights for a fitting problem*

Description

These functions reweight a reference sample to match constraints given by aggregate controls.

`ml_fit()` accepts an algorithm as argument and calls the corresponding function. This is useful if the result of multiple algorithms are compared to each other.

Usage

```
ml_fit(
  ml_problem,
  algorithm = c("entropy_o", "dss", "ipu", "hipf"),
  verbose = FALSE,
  ...,
  tol = 1e-06
)

is_ml_fit(x)

## S3 method for class 'ml_fit'
format(x, ...)

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

ml_fit_dss(
  ml_problem,
  method = c("raking", "linear", "logit"),
  ginv = gginv(),
  tol = 1e-06,
  verbose = FALSE
)

ml_fit_entropy_o(
  ml_problem,
  verbose = FALSE,
  tol = 1e-06,
  dfsane_args = list()
)

ml_fit_hipf(
  ml_problem,
  diff_tol = 16 * .Machine$double.eps,
  tol = 1e-06,
  maxiter = 2000,
  verbose = FALSE
)

ml_fit_ipu(
  ml_problem,
  diff_tol = 16 * .Machine$double.eps,
  tol = 1e-06,
  maxiter = 2000,
  verbose = FALSE
)
```

Arguments

`ml_problem`	A fitting problem created by `ml_problem()` or returned by `flatten_ml_fit_problem()`.
`algorithm`	Algorithm to use
`verbose`	If `TRUE`, print diagnostic output.
`...`	Further parameters passed to the algorithm
`tol`	Tolerance, the algorithm has succeeded when all target values are reached within this tolerance.
`x`	An object
`method`	Calibration method, one of `"raking"` (default), `"linear"`, or `"logit"`
`ginv`	Function that computes the Moore-Penrose pseudoinverse.
`dfsane_args`	Additional arguments (as a named list) passed to the `BB::dfsane()` function used internally for the optimization.
`diff_tol`	Tolerance, the algorithm stops when relative difference of control values between iterations drops below this value
`maxiter`	Maximum number of iterations.

Value

All functions return an object of class `ml_fit`, which is a named list under the hood. The class matches the function called, e.g., the return value of the `ml_fit_ipu` function also is of class `ml_fit_ipu`.

All returned objects contain at least the following components, which can be accessed with `$` or `[[`:

- `weights`: Resulting weights, compatible to the original reference sample
- `tol`: The input tolerance
- `iterations`: The actual number of iterations required to obtain the result
- `flat`: The flattened fitting problem, see `flatten_ml_fit_problem()`
- `flat_weights`: Weights in terms of the flattened fitting problem
- `residuals`: Absolute residuals
- `rel_residuals`: Relative residuals
- `success`: Are the residuals within the tolerance?

`is_ml_fit()` returns a logical.

References

Deville, J.-C. and Särndal, C.-E. (1992) Calibration estimators in survey sampling. *Journal of the American Statistical Association*, **87** (418), 376–382.

Deville, J.-C., Särndal, C.-E. and Sautory, O. (1993) Generalized raking procedures in survey sampling. *Journal of the American Statistical Association*, **88** (423), 1013–1020.

Bar-Gera, H., Konduri, K. C., Sana, B., Ye, X., & Pendyala, R. M. (2009, January). Estimating survey weights with multiple constraints using entropy optimization methods. In 88th Annual Meeting of the Transportation Research Board, Washington, DC.

Müller, K. and Axhausen, K. W. (2011), Hierarchical IPF: Generating a synthetic population for Switzerland, paper presented at the 51st Congress of the European Regional Science Association, University of Barcelona, Barcelona.

Ye, X., K. Konduri, R. M. Pendyala, B. Sana and P. A. Waddell (2009) A methodology to match distributions of both household and person attributes in the generation of synthetic populations, paper presented at the *88th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2009.

See Also

`dss()`, `gginv()`

`BB::dfsane()`

Examples

```
path <- toy_example("Tiny")
fit <- ml_fit(ml_problem = readRDS(path), algorithm = "entropy_o")
fit
fit$weights
fit$tol
fit$iterations
fit$flat
fit$flat_weights
fit$residuals
fit$rel_residuals
fit$success
ml_fit_dss(ml_problem = readRDS(path))
ml_fit_dss(ml_problem = readRDS(path), ginv = solve)
ml_fit_entropy_o(ml_problem = readRDS(path))
ml_fit_hipf(ml_problem = readRDS(path))
ml_fit_ipu(ml_problem = readRDS(path))
```

`ml_problem` *Create an instance of a fitting problem*

Description

The `ml_problem()` function is the first step for fitting a reference sample to known control totals with mlfit. All algorithms (see `ml_fit()`) expect an object created by this function (or optionally processed with `flatten_ml_fit_problem()`).

The `special_field_names()` function is useful for the `field_names` argument to `ml_problem`.

Usage

```
ml_problem(
  ref_sample,
  controls = list(individual = individual_controls, group = group_controls),
  field_names,
  individual_controls,
  group_controls,
  prior_weights = NULL,
  geo_hierarchy = NULL
)

is_ml_problem(x)

## S3 method for class 'ml_problem'
format(x, ...)

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

special_field_names(
  groupId,
  individualId,
  individualsPerGroup = NULL,
  count = NULL,
  zone = NULL,
  region = NULL
)
```

Arguments

`ref_sample`	The reference sample
`controls`	Control totals, by default initialized from the `individual_controls` and `group_controls` arguments
`field_names`	Names of special fields, construct using `special_field_names()`
`individual_controls, group_controls`	Control totals at individual and group level, given as a list of data frames where each data frame defines a control
`prior_weights`	Prior (or design) weights at group level; by default a vector of ones will be used, which corresponds to random sampling of groups

`geo_hierarchy`	A table shows mapping between a larger zoning level to many zones of a smaller zoning level. The column name of the larger level should be specified in `field_names` as 'region' and the smaller one as 'zone'.
`x`	An object
`...`	Ignored.
`groupId, individualId`	Name of the column that defines the ID of the group or the individual
`individualsPerGroup`	Obsolete.
`count`	Name of control total column in control tables (use first numeric column in each control by default).
`region, zone`	Name of the column that defines the region of the reference sample or the zone of the controls. Note that region is a larger area that contains more than one zone.

Value

An object of class `ml_problem` or a list of them if `geo_hierarchy` was given, essentially a named list with the following components:

`refSample` The reference sample, a `data.frame`.

`controls` A named list with two components, `individual` and `group`. Each contains a list of controls as `data.frames`.

`fieldNames` A named list with the names of special fields.

`is_ml_problem()` returns a logical.

Examples

```
# Create example from Ye et al., 2009

# Provide reference sample
ye <- tibble::tribble(
  ~HHNR, ~PNR, ~APER, ~HH_VAR, ~P_VAR,
  1, 1, 3, 1, 1,
  1, 2, 3, 1, 2,
  1, 3, 3, 1, 3,
  2, 4, 2, 1, 1,
  2, 5, 2, 1, 3,
  3, 6, 3, 1, 1,
  3, 7, 3, 1, 1,
  3, 8, 3, 1, 2,
  4, 9, 3, 2, 1,
  4, 10, 3, 2, 3,
  4, 11, 3, 2, 3,
  5, 12, 3, 2, 2,
  5, 13, 3, 2, 2,
  5, 14, 3, 2, 3,
  6, 15, 2, 2, 1,
  6, 16, 2, 2, 2,
  7, 17, 5, 2, 1,
  7, 18, 5, 2, 1,
  7, 19, 5, 2, 2,
  7, 20, 5, 2, 3,
  7, 21, 5, 2, 3,
  8, 22, 2, 2, 1,
  8, 23, 2, 2, 2
)
ye

# Specify control at household level
ye_hh <- tibble::tribble(
  ~HH_VAR, ~N,
  1,       35,
  2,       65
)
ye_hh

# Specify control at person level
ye_ind <- tibble::tribble(
  ~P_VAR, ~N,
  1, 91,
  2, 65,
  3, 104
)
ye_ind

ye_problem <- ml_problem(
  ref_sample = ye,
  field_names = special_field_names(
    groupId = "HHNR", individualId = "PNR", count = "N"
  ),
  group_controls = list(ye_hh),
  individual_controls = list(ye_ind)
)
ye_problem

fit <- ml_fit_dss(ye_problem)
fit$weights
```

`ml_replicate` *Replicate records in a reference sample based on its fitted weights*

Description

This function replicates each entry in a reference sample based on its fitted weights. This is useful if the result of multiple replication algorithms are compared to each other, or to generate a full synthetic population based on the result of a `ml_fit` object. Note that, all individual and group ids of the synthetic population are not the same as those in the original reference sample, and the total number of groups replicated is always very close to or equal the sum of the fitted group weights.

Usage

```
ml_replicate(
  ml_fit,
  algorithm = c("pp", "trs", "round"),
  verbose = FALSE,
  .keep_original_ids = FALSE
)
```

Arguments

`ml_fit`	A `ml_fit` object created by the `ml_fit()` family.
`algorithm`	Replication algorithm to use. "trs" is the 'Truncate, replicate, sample' integerisation algorithm proposed by Lovelace et al. (2013), "pp" is weighted sampling with replacement, and "round" is just simple rounding.
`verbose`	If `TRUE`, print diagnostic output.
`.keep_original_ids`	If `TRUE`, the original individual and group ids of the reference sample will be kept with suffix '_old'.

Value

The function returns a replicated sample in data.frame in the format used in the reference sample of the input `ml_fit` object.

References

```
 Lovelace, R., & Ballas, D. (2013). ‘Truncate, replicate, sample’:
     A method for creating integer weights for spatial microsimulation.
     Computers, Environment and Urban Systems, 41, 1-11.
```

Examples

```
path <- toy_example("Tiny")
fit <- ml_fit(ml_problem = readRDS(path), algorithm = "entropy_o")
syn_pop <- ml_replicate(fit, algorithm = "trs")
syn_pop
```

`toy_example` *Access to toy examples bundled in this package*

Description

Returns the paths to all available toy examples, or to a specific toy example. Load via `readRDS()`.

Usage

```
toy_example(name = NULL)
```

Arguments

`name`	Name of the example, default: return all

Value

A named vector of file system paths.

Examples

```
toy_example()

# Load example with results from Ye et al. (2009)
readRDS(toy_example("Tiny"))
```

Index