Package: ipfr (via r-universe)

September 7, 2024

Title List Balancing for Reweighting and Population Synthesis

Version 1.0.2

Description Performs iterative proportional updating given a seed table and an arbitrary number of marginal distributions. This is commonly used in population synthesis, survey raking, matrix rebalancing, and other applications. For example, a household survey may be weighted to match the known distribution of households by size from the census. An origin/ destination trip matrix might be balanced to match traffic counts. The approach used by this package is based on a paper from Arizona State University (Ye, Xin, et. al. (2009) <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.537.723&rep=rep1&type=pdf>).
Some enhancements have been made to their work including primary and secondary target balance/importance, general marginal agreement, and weight restriction.

License Apache License (== 2.0)

URL https://github.com/dkyleward/ipfr

BugReports https://github.com/dkyleward/ipfr/issues

Depends R (>= 3.2.0)

Imports dplyr (>= 0.7.3), ggplot2 (>= 2.2.1), magrittr (>= 1.5), tidyr (>= 0.5.1), mlr (>= 2.11)

LazyData true

Suggests knitr, rmarkdown, testthat (>= 2.1.0), covr

VignetteBuilder knitr

RoxygenNote 7.0.2

Repository https://dkyleward.r-universe.dev

RemoteUrl https://github.com/dkyleward/ipfr

RemoteRef HEAD

RemoteSha 0e4aad0b624a3816132d7ddc03cca468cdfc6996

Contents

`ipfr` *ipfr: A package to perform iterative proportional fitting*

Description

The main function is `ipu`. For a 2D/matrix problem, the `ipu_matrix` function is easier to use. The resulting `weight_tbl` from `ipu()` can be fed into `synthesize` to generate a synthetic population

Author(s)

Maintainer: Kyle Ward <kyleward084@gmail.com> [copyright holder]

Other contributors:

- Greg Macfarlane <gregmacfarlane@byu.edu> [contributor]

See Also

Useful links:

- https://github.com/dkyleward/ipfr
- Report bugs at https://github.com/dkyleward/ipfr/issues

`ipu` *Iterative Proportional Updating*

Description

A general case of iterative proportional fitting. It can satisfy two, disparate sets of marginals that do not agree on a single total. A common example is balancing population data using household- and person-level marginal controls. This could be for survey expansion or synthetic population creation. The second set of marginal/seed data is optional, meaning it can also be used for more basic IPF tasks.

Usage

```
ipu(
  primary_seed,
  primary_targets,
  secondary_seed = NULL,
  secondary_targets = NULL,
  primary_id = "id",
  secondary_importance = 1,
  relative_gap = 0.01,
  max_iterations = 100,
  absolute_diff = 10,
  weight_floor = 1e-05,
  verbose = FALSE,
  max_ratio = 10000,
  min_ratio = 1e-04
)
```

Arguments

`primary_seed`	In population synthesis or household survey expansion, this would be the household seed table (each record would represent a household). It could also be a trip table, where each row represents an origin-destination pair.
`primary_targets`	A `named list` of data frames. Each name in the list defines a marginal dimension and must match a column from the `primary_seed` table. The data frame associated with each named list element can contain a geography field (starting with "geo_"). If so, each row in the target table defines a new geography (these could be TAZs, tracts, clusters, etc.). The other column names define the marginal categories that targets are provided for. The vignette provides more detail.
`secondary_seed`	Most commonly, if the primary_seed describes households, the secondary seed table would describe the persons in each household. Must contain the same `primary_id` column that links each person to their respective household in `primary_seed`.
`secondary_targets`	Same format as `primary_targets`, but they constrain the `secondary_seed` table.
`primary_id`	The field used to join the primary and secondary seed tables. Only necessary if `secondary_seed` is provided.
`secondary_importance`	A `real` between 0 and 1 signifying the importance of the secondary targets. At an importance of 1, the function will try to match the secondary targets exactly. At 0, only the percentage distributions are used (see the vignette section "Target Agreement".)
`relative_gap`	After each iteration, the weights are compared to the previous weights and the the `relative_gap` threshold, then the process terminates.
`max_iterations`	maximum number of iterations to perform, even if `relative_gap` is not reached.

absolute_diff
: Upon completion, the ipu() function will report the worst-performing marginal category and geography based on the percent difference from the target. absolute_diff is a threshold below which percent differences don't matter.

  For example, if if a target value was 2, and the expanded weights equaled 1, that's a 100 is only 1.

  Defaults to 10.

weight_floor
: Minimum weight to allow in any cell to prevent zero weights. Set to .0001 by default. Should be arbitrarily small compared to your seed table weights.

verbose
: Print iteration details and worst marginal stats upon completion? Default FALSE.

max_ratio
: real number. The average weight per seed record is calculated by dividing the total of the targets by the number of records. The max_scale caps the maximum weight at a multiple of that average. Defaults to 10000 (basically turned off).

min_ratio
: real number. The average weight per seed record is calculated by dividing the total of the targets by the number of records. The min_scale caps the minimum weight at a multiple of that average. Defaults to 0.0001 (basically turned off).

Value

a named list with the primary_seed with weight, a histogram of the weight distribution, and two comparison tables to aid in reporting.

References

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.537.723&rep=rep1&type=pdf

Examples

```
hh_seed <- dplyr::tibble(
  id = c(1, 2, 3, 4),
  siz = c(1, 2, 2, 1),
  weight = c(1, 1, 1, 1),
  geo_cluster = c(1, 1, 2, 2)
)

hh_targets <- list()
hh_targets$siz <- dplyr::tibble(
  geo_cluster = c(1, 2),
  `1` = c(75, 100),
  `2` = c(25, 150)
)

result <- ipu(hh_seed, hh_targets, max_iterations = 5)
```

ipu_matrix *Balance a matrix given row and column targets*

Description

This function simplifies the call to 'ipu()' for the simple case of a matrix and row/column targets.

Usage

```
ipu_matrix(mtx, row_targets, column_targets, ...)
```

Arguments

mtx	a matrix
row_targets	a vector of targets that the row sums must match
column_targets	a vector of targets that the column sums must match
...	additional arguments that are passed to 'ipu()'. See ipu for details.

Value

A matrix that matches row and column targets

Examples

```
mtx <- matrix(data = runif(9), nrow = 3, ncol = 3)
row_targets <- c(3, 4, 5)
column_targets <- c(5, 4, 3)
ipu_matrix(mtx, row_targets, column_targets)
```

setup_arizona *Create the ASU example*

Description

Sets up the Arizona example IPU problem and is used in multiple places throughout the package (vignettes/tests).

Usage

```
setup_arizona()
```

Value

A list of four variables: hh_seed, hh_targets, per_seed, and per_targets. These can be used directly by ipu.

Examples

```
setup_arizona()
```

`synthesize` *Creates a synthetic population based on ipu results*

Description

A simple function that takes the `weight_tbl` output from `ipu` and randomly samples based on the weight.

Usage

```
synthesize(weight_tbl, group_by = NULL, primary_id = "id")
```

Arguments

`weight_tbl`	the `data.frame` of the same name output by `ipu`.
`group_by`	if provided, the `data.frame` will be grouped by this variable before sampling. If not provided, tidyverse/dplyr groupings will be respected. If no grouping info is present, samples are drawn from the entire table.
`primary_id`	The field used to join the primary and secondary seed tables. Only necessary if `secondary_seed` is provided.

Value

A `data.frame` with one record for each synthesized member of the population (e.g. household). A `new_id` column is created, but the previous `primary_id` column is maintained to facilitate joining back to other data sources (e.g. a person attribute table).

Examples

```
hh_seed <- dplyr::tibble(
id = c(1, 2, 3, 4),
siz = c(1, 2, 2, 1),
weight = c(1, 1, 1, 1),
geo_cluster = c(1, 1, 2, 2)
)
hh_targets <- list()
hh_targets$siz <- dplyr::tibble(
  geo_cluster = c(1, 2),
  `1` = c(75, 100),
  `2` = c(25, 150)
)
result <- ipu(hh_seed, hh_targets, max_iterations = 5)
synthesize(result$weight_tbl, "geo_cluster")
```

Index