Package ‘singleRcapture’

February 3, 2025

Type Package

Title Single-Source Capture-Recapture Models

Version 0.2.2

Description Implementation of single-source capture-recapture methods for population size estimation using zero-truncated, zero-one truncated and zero-truncated one-inflated Poisson, Geometric and Negative Binomial regression as well as Zelterman's and Chao's regression. Package includes point and interval estimators for the population size with variances estimated using analytical or bootstrap method. Details can be found in: van der Heijden et all. (2003) <doi:10.1191/1471082X03st057oa>, Böhning and van der Heijden (2019) <doi:10.1214/18-AOAS1232>, Böhning et al. (2020) Capture-Recapture Methods for the Social and Medical Sciences or Böhning and Friedl (2021) <doi:10.1007/s10260-021-00556-8>.

License MIT + file LICENSE

Encoding UTF-8

LazyData yes

RdMacros mathjaxr

BuildManual TRUE

VignetteBuilder knitr

Depends R (>= 3.5.0)

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL `https://github.com/ncn-foreigners/singleRcapture`

BugReports `https://github.com/ncn-foreigners/singleRcapture/issues`

Imports stats,
lamW,
mathjaxr,
sandwich,
doParallel,
foreach,
parallel

Suggests rmarkdown,
knitr,
tinytest,
covr,
VGAM,
lmtest

Contents

`carcassubmission` *British farm carcass submissions data*

Description

Data on British animal farms submissions to AHVLA. British farms are able to submit samples to AHVLA if cause of death for an animal cannot be determined and private veterinary surgeon decides to submit them, unless there is notifiable disease suspected then such a submission is not required.

This data set contains information about such farms. Only submissions that are included in this data frame are submissions of carcasses i.e. submissions of blood samples etc. are excluded.

Usage

```
data("carcassubmission")
```

Format

Data frame with 1,858 rows and 4 columns.

`TOTAL_SUB` Number of submissions of animal carcasses.

`log_size` Numerical value equal to logarithm of size of farm.

`log_distance` Numerical value equal to logarithm of distance to nearest AHVLA center.

`C_TYPE` Factor describing type of activity on farm that animals are used for. Either `Dairy` or `Beef`

References

This data set and its description was provided in publication: Böhning, D., Vidal Diez, A., Lerdsuwansri, R., Viwatwongkasem, C., and Arnold, M. (2013). "A generalization of Chao's estimator for covariate information". *Biometrics*, 69(4), 1033-1042. doi:10.1111/biom.12082

`chao` *Family functions in singleRcapture package*

Description

Package `singleRcapture` utilizes various family type functions that specify variable parts of population size estimation, regression, diagnostics and other necessary information that depends on the model. These functions are used as `model` argument in `estimatePopsize` function.

Usage

```
chao(lambdaLink = "loghalf", ...)

Hurdleztgeom(
  lambdaLink = c("log", "neglog"),
  piLink = c("logit", "cloglog", "probit"),
  ...
)

Hurdleztnegbin(
  nSim = 1000,
  epsSim = 1e-08,
  eimStep = 6,
  lambdaLink = c("log", "neglog"),
  alphaLink = c("log", "neglog"),
  piLink = c("logit", "cloglog", "probit"),
  ...
)

Hurdleztpoisson(
  lambdaLink = c("log", "neglog"),
  piLink = c("logit", "cloglog", "probit"),
  ...
)

oiztgeom(
  lambdaLink = c("log", "neglog"),
  omegaLink = c("logit", "cloglog", "probit"),
  ...
)

oiztnegbin(
  nSim = 1000,
  epsSim = 1e-08,
  eimStep = 6,
  lambdaLink = c("log", "neglog"),
  alphaLink = c("log", "neglog"),
  omegaLink = c("logit", "cloglog", "probit"),
  ...
)

oiztpoisson(
  lambdaLink = c("log", "neglog"),
  omegaLink = c("logit", "cloglog", "probit"),
  ...
)

zelterman(lambdaLink = "loghalf", ...)

zotgeom(lambdaLink = c("log", "neglog"), ...)

zotnegbin(
  nSim = 1000,
  epsSim = 1e-08,
  eimStep = 6,
  lambdaLink = c("log", "neglog"),
  alphaLink = c("log", "neglog"),
  ...
)

zotpoisson(lambdaLink = c("log", "neglog"), ...)

ztHurdlegeom(
  lambdaLink = c("log", "neglog"),
  piLink = c("logit", "cloglog", "probit"),
  ...
)

ztHurdlenegbin(
  nSim = 1000,
  epsSim = 1e-08,
  eimStep = 6,
  lambdaLink = c("log", "neglog"),
  alphaLink = c("log", "neglog"),
  piLink = c("logit", "cloglog", "probit"),
  ...
)

ztHurdlepoisson(
  lambdaLink = c("log", "neglog"),
  piLink = c("logit", "cloglog", "probit"),
  ...
)

ztgeom(lambdaLink = c("log", "neglog"), ...)

ztnegbin(
  nSim = 1000,
  epsSim = 1e-08,
  eimStep = 6,
  lambdaLink = c("log", "neglog"),
  alphaLink = c("log", "neglog"),
  ...
)

ztoigeom(
  lambdaLink = c("log", "neglog"),
  omegaLink = c("logit", "cloglog", "probit"),
  ...
)

ztoinegbin(
  nSim = 1000,
  epsSim = 1e-08,
  eimStep = 6,
  lambdaLink = c("log", "neglog"),
  alphaLink = c("log", "neglog"),
  omegaLink = c("logit", "cloglog", "probit"),
  ...
)

ztoipoisson(
  lambdaLink = c("log", "neglog"),
  omegaLink = c("logit", "cloglog", "probit"),
  ...
)

ztpoisson(lambdaLink = c("log", "neglog"), ...)
```

Arguments

`lambdaLink`	a link for Poisson parameter, `"log"` by default except for zelterman's and chao's models where only $\ln\left(\frac{x}{2}\right)$ is possible.
`...`	Additional arguments, not used for now.
`piLink`	a link for probability parameter, `"logit"` by default
`nSim`, `epsSim`	if working weights cannot be computed analytically these arguments specify maximum number of simulations allowed and precision level for finding them numerically respectively.
`eimStep`	a non negative integer describing how many values should be used at each step of approximation of information matrixes when no analytic solution is available (e.g. `"ztnegbin"`), default varies depending on a function. Higher value usually means faster convergence but may potentially cause issues with convergence.
`alphaLink`	a link for dispersion parameter, `"log"` by default
`omegaLink`	a link for inflation parameter, `"logit"` by default

Details

Most of these functions are based on some "base" distribution with support $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ that describe distribution of Y before truncation. Currently they include:

$$\mathbb{P}(Y=y|\lambda,\alpha) = \begin{cases} \frac{\lambda^y e^{-\lambda}}{y!} & \text{Poisson distribution} \\ \frac{\Gamma(y+\alpha^{-1})}{\Gamma(\alpha^{-1})y!}\left(\frac{\alpha^{-1}}{\alpha^{-1}+\lambda}\right)^{\alpha^{-1}}\left(\frac{\lambda}{\alpha^{-1}+\lambda}\right)^{y} & \text{negative binomial distribution} \\ \frac{\lambda^y}{(1+\lambda)^{y+1}} & \text{geometric distribution} \end{cases}$$

where λ is the Poisson parameter and α is the dispersion parameter. Geometric distribution is a special case of negative binomial distribution when $\alpha = 1$ it is included because negative binomial distribution is quite troublesome numerical regression in fitting. It is important to know that PMF of negative binomial distribution approaches the PMF of Poisson distribution when $\alpha \rightarrow 0^+$.

Note in literature on single source capture recapture models the dispersion parameter which introduces greater variability in negative binomial distribution compared to Poisson distribution is generally interpreted as explaining the *unobserved* heterogeneity i.e. presence of important unobserved independent variables. All these methods for estimating population size are tied to Poisson processes hence we use λ as parameter symbol instead of μ to emphasize this connection. Also will not be hard to see that **all** estimators derived from modifying the "base" distribution are unbiased if assumptions made by respective models are not violated.

The **zero truncated** models corresponding to "base" distributions are characterized by relation:

$$\mathbb{P}(Y=y|Y>0) = \begin{cases} \frac{\mathbb{P}(Y=y)}{1-\mathbb{P}(Y=0)} & \text{when } y \neq 0 \\ 0 & \text{when } y = 0 \end{cases}$$

which allows us to estimate parameter values using only observed part of population. These models lead to the following estimates, respectively:

$$\hat{N} = \sum_{k=1}^{N_{obs}} \frac{1}{1-\exp(-\lambda_k)} \quad \text{For Poisson distribution}$$

$$\hat{N} = \sum_{k=1}^{N_{obs}} \frac{1}{1-(1+\alpha_k\lambda_k)^{-\alpha_k^{-1}}} \quad \text{For negative binomial distribution}$$

$$\hat{N} = \sum_{k=1}^{N_{obs}} \frac{1+\lambda_k}{\lambda_k} \quad \text{For geometric distribution}$$

One common way in which assumptions of zero truncated models are violated is presence of **one inflation** the presence of which is somewhat similar in single source capture-recapture models to zero inflation in usual count data analysis. There are two ways in which one inflation may be understood, they relate to whether $\mathbb{P}(Y=0)$ is modified by inflation. The first approach is inflate (ω parameter) zero truncated distribution as:

$$\mathbb{P}_{new}(Y=y|Y>0) = \begin{cases} \omega + (1-\omega)\mathbb{P}_{old}(Y=1|Y>0) & \text{when: } y=1 \\ (1-\omega)\mathbb{P}_{old}(Y=y|Y>0) & \text{when: } y \neq 1 \end{cases}$$

which corresponds to:

$$\mathbb{P}_{new}(Y=y) = \begin{cases} \mathbb{P}_{old}(Y=0) & \text{when: } y=0 \\ \omega(1-\mathbb{P}(Y=0)) + (1-\omega)\mathbb{P}_{old}(Y=1) & \text{when: } y=1 \\ (1-\omega)\mathbb{P}_{old}(Y=y) & \text{when: } y>1 \end{cases}$$

before zero truncation. Models that utilize this approach are commonly referred to as *zero truncated one inflated models*. Another way of accommodating one inflation in SSCR is by putting inflation parameter on base distribution as:

$$\mathbb{P}_{new}(Y=y)=\begin{cases}\omega+(1-\omega)\mathbb{P}_{old}(Y=1) & \text{when: } y=1\\ (1-\omega)\mathbb{P}_{old}(Y=y) & \text{when: } y\neq 1\end{cases}$$

which then becomes:

$$\mathbb{P}_{new}(Y=y|Y>0)=\begin{cases}\frac{\omega}{1-(1-\omega)\mathbb{P}_{old}(Y=0)}+\frac{(1-\omega)}{1-(1-\omega)\mathbb{P}_{old}(Y=0)}\mathbb{P}_{old}(Y=1) & \text{when: } y=1\\ \frac{(1-\omega)}{1-(1-\omega)\mathbb{P}_{old}(Y=0)}\mathbb{P}_{old}(Y=y) & \text{when: } y>1\end{cases}$$

after truncation. It was shown by Böhning in 2022 paper that these approaches are equivalent in terms of maximizing likelihoods if we do not put formula on ω. They can however lead to different population size estimates.

For *zero truncated one inflated models* the formula for population size estimate $\hat{N}$ does not change since $\mathbb{P}(y=0)$ remains the same but estimation of parameters changes all calculations.

For *one inflated zero truncated models* population size estimates are expressed, respectively by:

$$\hat{N}=\sum_{k=1}^{N_{obs}}\frac{1}{1-(1-\omega_k)\exp(-\lambda_k)} \qquad \text{For base Poisson distribution}$$

$$\hat{N}=\sum_{k=1}^{N_{obs}}\frac{1}{1-(1-\omega_k)(1+\alpha_k\lambda_k)^{-\alpha_k^{-1}}} \qquad \text{For base negative binomial distribution}$$

$$\hat{N}=\sum_{k=1}^{N_{obs}}\frac{1+\lambda_k}{\lambda_k+\omega_k} \qquad \text{For base geometric distribution}$$

Zero one truncated models ignore one counts instead of accommodating one inflation by utilizing the identity

$$\ell_{\text{ztoi}}=\boldsymbol{f}_1\ln\frac{\boldsymbol{f}_1}{N_{obs}}+(N_{obs}-\boldsymbol{f}_1)\ln\left(1-\frac{\boldsymbol{f}_1}{N_{obs}}\right)+\ell_{\text{zot}}$$

where ℓ_{zot} is the log likelihood of zero one truncated distribution characterized by probability mass function:

$$\mathbb{P}(Y=y|Y>1)=\begin{cases}\frac{\mathbb{P}(Y=y)}{1-\mathbb{P}(Y=0)-\mathbb{P}(Y=1)} & \text{when } y>1\\ 0 & \text{when } y\in\{0,1\}\end{cases}$$

where $\mathbb{P}(Y)$ is the probability mass function of the "base" distribution. The identity above justifies use of zero one truncated, unfortunately it was only proven for intercept only models, however numerical simulations seem to indicate that even if the theorem cannot be extended for (non trivial) regression population size estimation is still possible.

For *zero one truncated models* population size estimates are expressed by:

$$\hat{N}=\boldsymbol{f}_1+\sum_{k=1}^{N_{obs}}\frac{1-\lambda_k\exp(-\lambda_k)}{1-\exp(-\lambda_k)-\lambda_k\exp(-\lambda_k)} \qquad \text{For base Poisson distribution}$$

$$\hat{N}=\boldsymbol{f}_1+\sum_{k=1}^{N_{obs}}\frac{1-\lambda_k(1+\alpha_k\lambda_k)^{-1-\alpha_k^{-1}}}{1-(1+\alpha_k\lambda_k)^{-\alpha_k^{-1}}-\lambda_k(1+\alpha_k\lambda_k)^{-1-\alpha_k^{-1}}} \qquad \text{For base negative binomial distribution}$$

$$\hat{N}=\boldsymbol{f}_1+\sum_{k=1}^{N_{obs}}\frac{\lambda_k^2+\lambda_k+1}{\lambda_k^2} \qquad \text{For base geometric distribution}$$

Pseudo hurdle models are experimental and not yet described in literature.

Lastly there are **chao** and **zelterman** models which are based on logistic regression on the dummy variable

$$Z = \begin{cases} 0 & \text{if } Y = 1 \\ 1 & \text{if } Y = 2 \end{cases}$$

based on the equation:

$$\text{logit}(p_k) = \ln\left(\frac{\lambda_k}{2}\right) = \boldsymbol{\beta}\mathbf{x}_k = \eta_k$$

where λ_k is the Poisson parameter.

The *zelterman* estimator of population size is expressed as:

$$\hat{N} = \sum_{k=1}^{N_{obs}} 1 - \exp\left(-\lambda_k\right)$$

and *chao* estimator has the form:

$$\hat{N} = N_{obs} + \sum_{k=1}^{\boldsymbol{f}_1 + \boldsymbol{f}_2} \frac{1}{\lambda_k + \frac{\lambda_k^2}{2}}$$

Value

A object of class `family` containing objects:

- `makeMinusLogLike` – A factory function for creating the following functions: $\ell(\boldsymbol{\beta}), \frac{\partial\ell}{\partial\boldsymbol{\beta}}, \frac{\partial^2\ell}{\partial\boldsymbol{\beta}^T\partial\boldsymbol{\beta}}$ functions from the $\boldsymbol{y}$ vector and the $\boldsymbol{X}_{vlm}$ matrix (or just $\boldsymbol{X}$ if applied to model with single linear predictor)which has the `deriv` argument with possible values in `c(0, 1, 2)` that determine which derivative to return; the default value is `0`, which represents the minus log-likelihood.
- `links` – A list with link functions.
- `mu.eta, variance` – Functions of linear predictors that return expected value and variance. The `type` argument with 2 possible values (`"trunc"` and `"nontrunc"`) that specifies whether to return $\mathbb{E}(Y|Y>0)$, $\text{var}(Y|Y>0)$ or $\mathbb{E}(Y)$, $\text{var}(Y)$ respectively; the `deriv` argument with values in `c(0, 1, 2)` is used for indicating the derivative with respect to the linear predictors, which is used for providing standard errors in the `predict` method.
- `family` – A string that specifies name of the model.
- `valideta, validmu` – For now it only returns `TRUE`. In the near future, it will be used to check whether applied linear predictors are valid (i.e. are transformed into some elements of the parameter space subjected to the inverse link function).
- `funcZ, Wfun` – Functions that create pseudo residuals and working weights used in IRLS algorithm.
- `devResids` – A function that given the linear predictors prior weights vector and response vector returns deviance residuals. Not all family functions have these functions implemented yet.
- `pointEst, popVar` – Functions that given prior weights linear predictors and in the latter case also estimate of $\text{cov}(\hat{\boldsymbol{\beta}})$ and $\boldsymbol{X_{vlm}}$ matrix return point estimate for population size and analytic estimation of its variance.There is a additional boolean parameter `contr` in the former function that if set to true returns contribution of each unit.
- `etaNames` – Names of linear predictors.
- `densityFunction` – A function that given linear predictors returns value of PMF at values `x`. Additional argument `type` specifies whether to return $\mathbb{P}(Y|Y>0)$ or $\mathbb{P}(Y)$.
- `simulate` – A function that generates values of a dependent vector given linear predictors.
- `getStart` – An expression for generating starting points.

Author(s)

Piotr Chlebicki, Maciej Beręsewicz

See Also

`estimatePopsize()`

`confint.singleRStaticCountData` *Confidence Intervals for Model Parameters*

Description

A function that computes studentized confidence intervals for model coefficients.

Usage

```
## S3 method for class 'singleRStaticCountData'
confint(object, parm, level = 0.95, ...)
```

Arguments

`object`	object of singleRStaticCountData class.
`parm`	names of parameters for which confidence intervals are to be computed, if missing all parameters will be considered.
`level`	confidence level for intervals.
`...`	currently does nothing.

Value

An object with named columns that include upper and lower limit of confidence intervals.

`controlMethod` *Control parameters for regression*

Description

`controlMethod` constructs a list with all necessary control parameters for regression fitting in `estimatePopsizeFit` and `estimatePopsize`.

Usage

```
controlMethod(
  epsilon = 1e-08,
  maxiter = 1000,
  verbose = 0,
  printEveryN = 1L,
  coefStart = NULL,
  etaStart = NULL,
  optimMethod = "Nelder-Mead",
  silent = FALSE,
  optimPass = FALSE,
  stepsize = 1,
  checkDiagWeights = TRUE,
  weightsEpsilon = 1e-08,
  momentumFactor = 0,
  saveIRLSlogs = FALSE,
  momentumActivation = 5,
  criterion = c("coef", "abstol", "reltol")
)
```

Arguments

`epsilon` a tolerance level for fitting algorithms by default `1e-8`.

`maxiter` a maximum number of iterations.

`verbose` a numeric value indicating whether to trace steps of fitting algorithm for `IRLS` fitting method different values of verbose give the following information:

- 1 – Returns information on the number of current iteration and current log-likelihood.
- 2 – Returns information on vector of regression parameters at current iteration (and all of the above).
- 3 – Returns information on reduction of log-likelihood at current iteration (and all of the above).
- 4 – Returns information on value of log-likelihood function gradient at current iteration (and all of the above).
- 5 – Returns information on convergence criterion and values that are taken into account when considering convergence (and all of the above).

if `optim` method was chosen verbose will be passed to `stats::optim()` as trace.

`printEveryN` an integer value indicating how often to print information specified in `verbose`, by default set to `1`.

`coefStart, etaStart`
initial parameters for regression coefficients or linear predictors if `NULL`. For `IRLS` fitting only `etaStart` is needed so if `coefStart` is provided it will be converted to `etaStart`, for `optim` fitting `coefStart` is necessary and argument `etaStart` will be ignored.

`optimMethod` a method of `stats::optim()` used `"Nelder-Mead"` is the default .

`silent` a logical value, indicating whether warnings in `IRLS` method should be suppressed.

`optimPass` an optional list of parameters passed to `stats::optim(..., control = optimPass)` if FALSE then list of control parameters will be inferred from other parameters.

`stepsize`	only for `IRLS`, scaling of updates to `beta` vector lower value means slower convergence but more accuracy by default 1. In general if fitting algorithm fails lowering this value tends to be most effective at correcting it.
`checkDiagWeights`	a logical value indicating whether to check if diagonal elements of working weights matrixes in `IRLS` are sufficiently positive so that these matrixes are positive defined. By default `TRUE`.
`weightsEpsilon`	a small number to ensure positive definedness of weights matrixes. Only matters if `checkDiagWeights` is set to `TRUE`. By default `1e-8`.
`momentumFactor`	an experimental parameter in `IRLS` only allowing for taking previous step into account at current step, i.e instead of updating regression parameters as: $$\boldsymbol{\beta}_{(a)} = \boldsymbol{\beta}_{(a-1)} + \text{stepsize} \cdot \text{step}_{(a)}$$ the update will be made as: $$\boldsymbol{\beta}_{(a)} = \boldsymbol{\beta}_{(a-1)} + \text{stepsize} \cdot (\text{step}_{(a)} + \text{momentum} \cdot \text{step}_{(a-1)})$$
`saveIRLSlogs`	a logical value indicating if information specified in `verbose` should be saved to output object, by default `FALSE`.
`momentumActivation`	the value of log-likelihood reduction bellow which momentum will apply.
`criterion`	a criterion used to determine convergence in `IRLS`, multiple values may be provided. By default `c("coef", "abstol")`.

Value

List with selected parameters, it is also possible to call list directly.

Author(s)

Piotr Chlebicki, Maciej Beręsewicz

See Also

`estimatePopsize()` `estimatePopsizeFit()` `controlModel()` `controlPopVar()`

`controlModel` *Control parameters specific to some models*

Description

`controlModel` constructs a list with all necessary control parameters in `estimatePopsize` that are either specific to selected model or do not fit anywhere else.

Specifying additional formulas should be done by using only right hand side of the formula also for now all variables from additional formulas should also be included in the "main" formula.

Usage

```
controlModel(
  weightsAsCounts = FALSE,
  omegaFormula = ~1,
  alphaFormula = ~1,
  piFormula = ~1
)
```

Arguments

`weightsAsCounts`	a boolean value indicating whether to treat `weights` argument as number of occurrences for each row in the `data` and adjust necessary methods and functionalities, like adjustments in bootstrap or decreasing weights in `dfbeta` instead or deleting rows from data, to accommodate this form of model specification.
`omegaFormula`	a formula for inflation parameter in one inflated zero truncated and zero truncated one inflated models.
`alphaFormula`	a formula for dispersion parameter in negative binomial based models.
`piFormula`	a formula for probability parameter in pseudo hurdle zero truncated and zero truncated pseudo hurdle models.

Value

A list with selected parameters, it is also possible to call list directly.

Author(s)

Piotr Chlebicki, Maciej Beręsewicz

See Also

`estimatePopsize()` `controlMethod()` `controlPopVar()` `singleRmodels()`

`controlPopVar` *Control parameters for population size estimation*

Description

Creating control parameters for population size estimation and respective standard error and variance estimation.

Usage

```
controlPopVar(
  alpha = 0.05,
  bootType = c("parametric", "semiparametric", "nonparametric"),
  B = 500,
  confType = c("percentilic", "normal", "basic"),
  keepbootStat = TRUE,
  traceBootstrapSize = FALSE,
  bootstrapVisualTrace = FALSE,
  fittingMethod = c("optim", "IRLS"),
  bootstrapFitcontrol = NULL,
  sd = c("sqrtVar", "normalMVUE"),
  covType = c("observedInform", "Fisher"),
  cores = 1L
)
```

Arguments

alpha	a significance level, 0.05 used by default.
bootType	the bootstrap type to be used. Default is "parametric", other possible values are: "semiparametric" and "nonparametric".
B	a number of bootstrap samples to be performed (default 500).
confType	a type of confidence interval for bootstrap confidence interval, "percentile" by default. Other possibilities: "studentized" and "basic".
keepbootStat	a boolean value indicating whether to keep a vector of statistics produced by bootstrap.
traceBootstrapSize	a boolean value indicating whether to print size of bootstrapped sample after truncation for semi- and fully parametric bootstraps.
bootstrapVisualTrace	a boolean value indicating whether to plot bootstrap statistics in real time if cores = 1 if cores > 1 it instead indicates whether to make progress bar.
fittingMethod	a method used for fitting models from bootstrap samples.
bootstrapFitcontrol	control parameters for each regression works exactly like controlMethod but for fitting models from bootstrap samples.
sd	a character indicating how to compute standard deviation of population size estimator either as: $$\hat{\sigma}=\sqrt{\hat{\text{var}}(\hat{N})}$$ for sqrt (which is slightly biased if $\hat{N}$ has a normal distribution) or for normalMVUE as the unbiased minimal variance estimator for normal distribution: $$\hat{\sigma}=\sqrt{\hat{\text{var}}(\hat{N})}\frac{\Gamma\left(\frac{N_{obs}-1}{2}\right)}{\Gamma\left(\frac{N_{obs}}{2}\right)}\sqrt{\frac{N_{obs}}{2}}$$ where the ration involving gamma functions is computed by log gamma function.
covType	a type of covariance matrix for regression parameters by default observed information matrix.
cores	for bootstrap only, a number of processor cores to be used, any number greater than 1 activates code designed with doParallel, foreach and parallel packages. Note that for now using parallel computing makes tracing impossible so traceBootstrapSize and bootstrapVisualTrace parameters are ignored in this case.

Value

A list with selected parameters, it is also possible to call list directly.

Author(s)

Piotr Chlebicki, Maciej Beręsewicz

See Also

estimatePopsize() controlModel() controlMethod()

`estfun.singleRStaticCountData`
Heteroscedasticity-Consistent Covariance Matrix Estimation for singleRStaticCountData class

---

Description

S3 method for `vcovHC` to handle `singleRStaticCountData` class objects. Works exactly like `vcovHC.default` the only difference being that this method handles vector generalised linear models. Updating the covariance matrix in variance/standard error estimation for population size estimator can be done via `redoPopEstimation()`

Usage

```
## S3 method for class 'singleRStaticCountData'
estfun(x, ...)

## S3 method for class 'singleRStaticCountData'
bread(x, ...)

## S3 method for class 'singleRStaticCountData'
vcovHC(
  x,
  type = c("HC3", "const", "HC", "HC0", "HC1", "HC2", "HC4", "HC4m", "HC5"),
  omega = NULL,
  sandwich = TRUE,
  ...
)
```

Arguments

`x`	a fitted `singleRStaticCountData` class object.
`...`	for `vcovHC` additional optional arguments passed to the following functions: • `estfun` – for empirical estimating functions. • `hatvalues` – for diagonal elements of projection matrix. • `sandwich` – only if `sandwich` argument in function call was set to `TRUE`. • `vcov` – when calling `bread` internally.
`type`	a character string specifying the estimation type, same as in `sandwich::vcovHC.default`. HC3 is the default value.
`omega`	a vector or a function depending on the arguments residuals (i.e. the derivative of log-likelihood with respect to each linear predictor), diaghat (the diagonal of the corresponding hat matrix) and df (the residual degrees of freedom), same as in `sandwich::vcovHC.default`.

`sandwich`	logical. Should the sandwich estimator be computed? If set to FALSE only the meat matrix is returned. Same as in `sandwich::vcovHC()`

Value

Variance-covariance matrix estimation corrected for heteroscedasticity of regression errors

Author(s)

Piotr Chlebicki, Maciej Beręsewicz

See Also

`sandwich::vcovHC()` `redoPopEstimation()`

Examples

```
set.seed(1)
N <- 10000
gender <- rbinom(N, 1, 0.2)
eta <- -1 + 0.5*gender
counts <- rpois(N, lambda = exp(eta))
df <- data.frame(gender, eta, counts)
df2 <- subset(df, counts > 0)
mod1 <-  estimatePopsize(
  formula = counts ~ 1 + gender,
  data = df2,
  model = "ztpoisson",
  method = "optim",
  popVar = "analytic"
)
require(sandwich)
HC <- sandwich::vcovHC(mod1, type = "HC4")
Fisher <- vcov(mod1, "Fisher") # variance covariance matrix obtained from
#Fisher (expected) information matrix
HC
Fisher
# usual results
summary(mod1)
# updated results
summary(mod1, cov = HC,
popSizeEst = redoPopEstimation(mod1, cov = HC))
# estimating equations
mod1_sims <- sandwich::estfun(mod1)
head(mod1_sims)
# bread method
all(vcov(mod1, "Fisher") * nrow(df2) == sandwich::bread(mod1, type = "Fisher"))
```

`estimatePopsize` *Single-source capture-recapture models*

Description

`estimatePopsize` first fits appropriate (v)glm model and then estimates full (observed and unobserved) population size. In this types of models it is assumed that the response vector (i.e. the dependent variable) corresponds to the number of times a given unit was observed in the source. Population size is then usually estimated by Horvitz-Thompson type estimator:

$$\hat{N} = \sum_{k=1}^{N} \frac{I_k}{\mathbb{P}(Y_k > 0)} = \sum_{k=1}^{N_{obs}} \frac{1}{1 - \mathbb{P}(Y_k = 0)}$$

where $I_k = I_{Y_k > 0}$ are indicator variables, with value 1 if kth unit was observed at least once and 0 otherwise.

Usage

```
estimatePopsize(formula, ...)

## Default S3 method:
estimatePopsize(
  formula,
  data,
  model = c("ztpoisson", "ztnegbin", "ztgeom", "zotpoisson", "ztoipoisson",
   "oiztpoisson", "ztHurdlepoisson", "Hurdleztpoisson", "zotnegbin", "ztoinegbin",
   "oiztnegbin", "ztHurdlenegbin", "Hurdleztnegbin", "zotgeom", "ztoigeom", "oiztgeom",
    "ztHurdlegeom", "ztHurdlegeom", "zelterman", "chao"),
  weights = NULL,
  subset = NULL,
  naAction = NULL,
  method = c("optim", "IRLS"),
  popVar = c("analytic", "bootstrap", "noEst"),
  controlMethod = NULL,
  controlModel = NULL,
  controlPopVar = NULL,
  modelFrame = TRUE,
  x = FALSE,
  y = TRUE,
  contrasts = NULL,
  ratioReg = FALSE,
  offset,
  ...
)
```

Arguments

`formula`	a formula for the model to be fitted, only applied to the "main" linear predictor. Only single response models are available.
`...`	additional optional arguments passed to other methods eg. `estimatePopsizeFit`.
`data`	a data frame or object coercible to data.frame class containing data for the regression and population size estimation.
`model`	a model for regression and population estimate full description in `singleRmodels()`.
`weights`	an optional object of prior weights used in fitting the model. Can be used to specify number of occurrences of rows in data see `controlModel()`

`subset`	a logical vector indicating which observations should be used in regression and population size estimation. It will be evaluated on `data` argument provided on call.
`naAction`	Not yet implemented.
`method`	a method for fitting values currently supported: iteratively reweighted least squares (`IRLS`) and maximum likelihood (`optim`).
`popVar`	a method of constructing confidence interval and estimating the standard error either analytic or bootstrap. Bootstrap confidence interval type may be specified in `controlPopVar.` There is also the third possible value of `noEst` which skips the population size estimate all together.
`controlMethod`	a list indicating parameters to use in fitting the model may be constructed with `singleRcapture::controlMethod` function. More information included in `controlMethod()`.
`controlModel`	a list indicating additional formulas for regression (like formula for inflation parameter/dispersion parameter) may be constructed with `singleRcapture::controlModel` function. More information will eventually be included in `controlModel()`.
`controlPopVar`	a list indicating parameters to use in estimating variance of population size estimation may be constructed with `singleRcapture::controlPopVar` function. More information included in `controlPopVar()`.
`modelFrame, x, y`	logical values indicating whether to return model matrix, dependent vector and model matrix as a part of output.
`contrasts`	not yet implemented.
`ratioReg`	Not yet implemented
`offset`	a matrix of offset values with the number of columns matching the number of distribution parameters providing offset values to each of linear predictors.

Details

The generalized linear model is characterized by equation

$$\boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta}$$

where $\boldsymbol{X}$ is the (lm) model matrix. The vector generalized linear model is similarly characterized by equations

$$\boldsymbol{\eta}_k = \boldsymbol{X}_k\boldsymbol{\beta}_k$$

where $\boldsymbol{X}_k$ is a (lm) model matrix constructed from appropriate formula (specified in `controlModel` parameter).

The $\boldsymbol{\eta}$ is then a vector constructed as:

$$\boldsymbol{\eta} = \begin{pmatrix} \boldsymbol{\eta}_1 \\ \boldsymbol{\eta}_2 \\ \dots \\ \boldsymbol{\eta}_p \end{pmatrix}^T$$

and in cases of models in our package the (vlm) model matrix is constructed as a block matrix:

$$\boldsymbol{X}_{vlm} = \begin{pmatrix} \boldsymbol{X}_1 & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X}_2 & \dots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{X}_p \end{pmatrix}$$

this differs from convention in VGAM package (if we only consider our special cases of vglm models) but this is just a convention and does not affect the model, this convention is taken because it makes fitting with IRLS (explanation of algorithm in `estimatePopsizeFit()`) algorithm easier. (If `constraints` matrixes in `vglm` match the ones we implicitly use the `vglm` model matrix differs with respect to order of `kronecker` multiplication of `X` and `constraints`.) In this package we use observed likelihood to fit regression models.

As mentioned above usually the population size estimation is done via:

$$\hat{N} = \sum_{k=1}^{N} \frac{I_k}{\mathbb{P}(Y_k > 0)} = \sum_{k=1}^{N_{obs}} \frac{1}{1 - \mathbb{P}(Y_k = 0)}$$

where $I_k = I_{Y_k > 0}$ are indicator variables, with value 1 if kth unit was observed at least once and 0 otherwise. The $\mathbb{P}(Y_k > 0)$ are estimated by maximum likelihood.

The following assumptions are usually present when using the method of estimation described above:

1. The specified regression model is correct. This entails linear relationship between independent variables and dependent ones and dependent variable being generated by appropriate distribution.
2. No unobserved heterogeneity. If this assumption is broken there are some possible (admittedly imperfect) workarounds see details in `singleRmodels()`.
3. The population size is constant in relevant time frame.
4. Depending on confidence interval construction (asymptotic) normality of $\hat{N}$ statistic is assumed.

There are two ways of estimating variance of estimate $\hat{N}$, the first being `"analytic"` usually done by application of law of total variance to $\hat{N}$:

$$\text{var}(\hat{N}) = \mathbb{E}\left(\text{var}\left(\hat{N}|I_1, \dots, I_n\right)\right) + \text{var}\left(\mathbb{E}(\hat{N}|I_1, \dots, I_n)\right)$$

and then by δ method to $\hat{N}|I_1, \dots I_N$:

$$\mathbb{E}\left(\text{var}\left(\hat{N}|I_1, \dots, I_n\right)\right) = \left.\left(\frac{\partial(N|I_1, \dots, I_N)}{\partial\boldsymbol{\beta}}\right)^T \text{cov}\left(\boldsymbol{\beta}\right)\left(\frac{\partial(N|I_1, \dots, I_N)}{\partial\boldsymbol{\beta}}\right)\right|_{\boldsymbol{\beta}=\hat{\boldsymbol{\beta}}}$$

and the var $\left(\mathbb{E}(\hat{N}|I_1, \dots, I_n)\right)$ term may be derived analytically (if we assume independence of observations) since $\hat{N}|I_1, \dots, I_n$ is just a constant.

In general this gives us:

$$\begin{aligned}
\text{var}\left(\mathbb{E}(\hat{N}|I_1,\ldots,I_n)\right) &= \text{var}\left(\sum_{k=1}^{N}\frac{I_k}{\mathbb{P}(Y_k>0)}\right)\\
&= \sum_{k=1}^{N}\text{var}\left(\frac{I_k}{\mathbb{P}(Y_k>0)}\right)\\
&= \sum_{k=1}^{N}\frac{1}{\mathbb{P}(Y_k>0)^2}\text{var}(I_k)\\
&= \sum_{k=1}^{N}\frac{1}{\mathbb{P}(Y_k>0)^2}\mathbb{P}(Y_k>0)(1-\mathbb{P}(Y_k>0))\\
&= \sum_{k=1}^{N}\frac{1}{\mathbb{P}(Y_k>0)}(1-\mathbb{P}(Y_k>0))\\
&\approx \sum_{k=1}^{N}\frac{I_k}{\mathbb{P}(Y_k>0)^2}(1-\mathbb{P}(Y_k>0))\\
&= \sum_{k=1}^{N_{obs}}\frac{1-\mathbb{P}(Y_k>0)}{\mathbb{P}(Y_k>0)^2}
\end{aligned}$$

Where the approximation on 6th line appears because in 5th line we sum over all units, that includes unobserved units, since I_k are independent and $I_k \sim b(\mathbb{P}(Y_k > 0))$ the 6th line is an unbiased estimator of the 5th line.

The other method for estimating variance is `"bootstrap"`, but since $N_{obs} = \sum_{k=1}^{N} I_k$ is also a random variable bootstrap will not be as simple as just drawing N_{obs} units from data with replacement and just computing $\hat{N}$.

Method described above is referred to in literature as `"nonparametric"` bootstrap (see `controlPopVar()`), due to ignoring variability in observed sample size it is likely to underestimate variance.

A more sophisticated bootstrap procedure may be described as follows:

1. Compute the probability distribution as:

$$\frac{\hat{\boldsymbol{f}}_0}{\hat{N}}, \frac{\boldsymbol{f}_1}{\hat{N}}, \ldots, \frac{\boldsymbol{f}_{\max y}}{\hat{N}}$$

   where $\boldsymbol{f}_n$ denotes observed marginal frequency of units being observed exactly n times.
2. Draw $\hat{N}$ units from the distribution above (if $\hat{N}$ is not an integer than draw $\lfloor\hat{N}\rfloor + b(\hat{N} - \lfloor\hat{N}\rfloor)$), where $\lfloor\cdot\rfloor$ is the floor function.
3. Truncated units with $y = 0$.
4. If there are covariates draw them from original data with replacement from uniform distribution. For example if unit drawn to new data has $y = 2$ choose one of covariate vectors from original data that was associated with unit for which was observed 2 times.
5. Regress $\boldsymbol{y}_{new}$ on $\boldsymbol{X}_{vlmnew}$ and obtain $\hat{\boldsymbol{\beta}}_{new}$, with starting point $\hat{\boldsymbol{\beta}}$ to make it slightly faster, use them to compute $\hat{N}_{new}$.
6. Repeat 2-5 unit there are at least `B` statistics are obtained.
7. Compute confidence intervals based on `alpha` and `confType` specified in `controlPopVar()`.

To do step 1 in procedure above it is convenient to first draw binary vector of length $\lfloor\hat{N}\rfloor + b(\hat{N} - \lfloor\hat{N}\rfloor)$ with probability $1 - \frac{\hat{\boldsymbol{f}}_0}{\hat{N}}$, sum elements in that vector to determine the sample size and then draw sample of this size uniformly from the data.

This procedure is known in literature as `"semiparametric"` bootstrap it is necessary to assume that the have a correct estimate $\hat{N}$ in order to use this type of bootstrap.

Lastly there is `"paramteric"` bootstrap where we assume that the probabilistic model used to obtain $\hat{N}$ is correct the bootstrap procedure may then be described as:

1. Draw $\lfloor\hat{N}\rfloor + b(\hat{N} - \lfloor\hat{N}\rfloor)$ covariate information vectors with replacement from data according to probability distribution that is proportional to: N_k, where N_k is the contribution of kth unit i.e. $\dfrac{1}{\mathbb{P}(Y_k > 0)}$.
2. Determine $\boldsymbol{\eta}$ matrix using estimate $\hat{\boldsymbol{\beta}}$.
3. Generate $\boldsymbol{y}$ (dependent variable) vector using $\boldsymbol{\eta}$ and probability mass function associated with chosen model.
4. Truncated units with $y = 0$ and construct $\boldsymbol{y}_{new}$ and $\boldsymbol{X}_{vlmnew}$.
5. Regress $\boldsymbol{y}_{new}$ on $\boldsymbol{X}_{vlmnew}$ and obtain $\hat{\boldsymbol{\beta}}_{new}$ use them to compute $\hat{N}_{new}$.
6. Repeat 1-5 unit there are at least `B` statistics are obtained.
7. Compute confidence intervals based on `alpha` and `confType` specified in `controlPopVar()`

It is also worth noting that in the `"analytic"` method `estimatePopsize` only uses "standard" covariance matrix estimation. It is possible that improper covariance matrix estimate is the only part of estimation that has its assumptions violated. In such cases post-hoc procedures are implemented in this package to address this issue.

Lastly confidence intervals for $\hat{N}$ are computed (in analytic case) either by assuming that it follows a normal distribution or that variable $\ln(N - \hat{N})$ follows a normal distribution.

These estimates may be found using either `summary.singleRStaticCountData` method or `popSizeEst.singleRStati` function. They're labelled as `normal` and `logNormal` respectively.

Value

Returns an object of class `c("singleRStaticCountData", "singleR", "glm", "lm")` with type `list` containing:

- `y` – Vector of dependent variable if specified at function call.
- `X` – Model matrix if specified at function call.
- `formula` – A list with formula provided on call and additional formulas specified in `controlModel`.
- `call` – Call matching original input.
- `coefficients` – A vector of fitted coefficients of regression.
- `control` – A list of control parameters for `controlMethod` and `controlModel`, `controlPopVar` is included in populationSize.
- `model` – Model which estimation of population size and regression was built, object of class family.
- `deviance` – Deviance for the model.
- `priorWeights` – Prior weight provided on call.
- `weights` – If `IRLS` method of estimation was chosen weights returned by `IRLS`, otherwise same as `priorWeights`.

- `residuals` – Vector of raw residuals.
- `logL` – Logarithm likelihood obtained at final iteration.
- `iter` – Numbers of iterations performed in fitting or if `stats::optim` was used number of call to loglikelihood function.
- `dfResiduals` – Residual degrees of freedom.
- `dfNull` – Null degrees of freedom.
- `fittValues` – Data frame of fitted values for both mu (the expected value) and lambda (Poisson parameter).
- `populationSize` – A list containing information of population size estimate.
- `modelFrame` – Model frame if specified at call.
- `linearPredictors` – Vector of fitted linear predictors.
- `sizeObserved` – Number of observations in original model frame.
- `terms` – terms attribute of model frame used.
- `contrasts` – contrasts specified in function call.
- `naAction` – naAction used.
- `which` – list indicating which observations were used in regression/population size estimation.
- `fittingLog` – log of fitting information for `"IRLS"` fitting if specified in `controlMethod`.

Author(s)

Piotr Chlebicki, Maciej Beręsewicz

References

General single source capture recapture literature:

Zelterman, Daniel (1988). ‘Robust estimation in truncated discrete distributions with application to capture-recapture experiments’. In: Journal of statistical planning and inference 18.2, pp. 225–237.

Heijden, Peter GM van der et al. (2003). ‘Point and interval estimation of the population size using the truncated Poisson regression model’. In: Statistical Modelling 3.4, pp. 305–322. doi: 10.1191/1471082X03st057oa.

Cruyff, Maarten J. L. F. and Peter G. M. van der Heijden (2008). ‘Point and Interval Estimation of the Population Size Using a Zero-Truncated Negative Binomial Regression Model’. In: Biometrical Journal 50.6, pp. 1035–1050. doi: 10.1002/bimj.200810455

Böhning, Dankmar and Peter G. M. van der Heijden (2009). ‘A covariate adjustment for zero-truncated approaches to estimating the size of hidden and elusive populations’. In: The Annals of Applied Statistics 3.2, pp. 595–610. doi: 10.1214/08-AOAS214.

Böhning, Dankmar, Alberto Vidal-Diez et al. (2013). ‘A Generalization of Chao’s Estimator for Covariate Information’. In: Biometrics 69.4, pp. 1033– 1042. doi: 10.1111/biom.12082

Böhning, Dankmar and Peter G. M. van der Heijden (2019). ‘The identity of the zero-truncated, one-inflated likelihood and the zero-one-truncated likelihood for general count densities with an application to drink-driving in Britain’. In: The Annals of Applied Statistics 13.2, pp. 1198–1211. doi: 10.1214/18-AOAS1232.

Navaratna WC, Del Rio Vilas VJ, Böhning D. Extending Zelterman’s approach for robust estimation of population size to zero-truncated clustered Data. Biom J. 2008 Aug;50(4):584-96. doi: 10.1002/bimj.200710441.

Böhning D. On the equivalence of one-inflated zero-truncated and zero-truncated one-inflated count data likelihoods. Biom J. 2022 Aug 15. doi: 10.1002/bimj.202100343.

Böhning, D., Friedl, H. Population size estimation based upon zero-truncated, one-inflated and sparse count data. Stat Methods Appl 30, 1197–1217 (2021). doi: 10.1007/s10260-021-00556-8

Bootstrap:

Zwane, PGM EN and Van der Heijden, Implementing the parametric bootstrap in capture-recapture models with continuous covariates 2003 Statistics & probability letters 65.2 pp 121-125

Norris, James L and Pollock, Kenneth H Including model uncertainty in estimating variances in multiple capture studies 1996 in Environmental and Ecological Statistics 3.3 pp 235-244

Vector generalized linear models:

Yee, T. W. (2015). Vector Generalized Linear and Additive Models: With an Implementation in R. New York, USA: Springer. ISBN 978-1-4939-2817-0.

See Also

`stats::glm()` – For more information on generalized linear models.

`stats::optim()` – For more information on `optim` function used in `optim` method of fitting regression.

`controlMethod()` – For control parameters related to regression.

`controlPopVar()` – For control parameters related to population size estimation.

`controlModel()` – For control parameters related to model specification.

`estimatePopsizeFit()` – For more information on fitting procedure in `esitmate_popsize`.

`popSizeEst()` `redoPopEstimation()` – For extracting population size estimation results are applying post-hoc procedures.

`summary.singleRStaticCountData()` – For summarizing important information about the model and population size estimation results.

`marginalFreq()` – For information on marginal frequencies and comparison between observed and fitted quantities.

`VGAM::vglm()` – For more information on vector generalized linear models.

`singleRmodels()` – For description of various models.

Examples

```
# Model from 2003 publication
# Point and interval estimation of the
# population size using the truncated Poisson regression mode
# Heijden, Peter GM van der et al. (2003)
model <- estimatePopsize(
   formula = capture ~ gender + age + nation,
   data = netherlandsimmigrant,
   model = ztpoisson
)
summary(model)
# Graphical presentation of model fit
plot(model, "rootogram")
# Statistical test
# see documentation for summary.singleRmargin
summary(marginalFreq(model), df = 1, "group")
```

```
# We currently support 2 methods of numerical fitting
# (generalized) IRLS algorithm and via stats::optim
# the latter one is faster when fitting negative binomial models
# (and only then) due to IRLS having to numerically compute
# (expected) information matrixes, optim is also less reliable when
# using alphaFormula argument in controlModel
modelNegBin <- estimatePopsize(
    formula = TOTAL_SUB ~ .,
    data = farmsubmission,
    model = ztnegbin,
    method = "optim"
)
summary(modelNegBin)
summary(marginalFreq(modelNegBin))

# More advanced call that specifies additional formula and shows
# in depth information about fitting procedure
pseudoHurdleModel <- estimatePopsize(
    formula = capture ~ nation + age,
    data = netherlandsimmigrant,
    model = Hurdleztgeom,
    method = "IRLS",
    controlMethod = controlMethod(verbose = 5),
    controlModel = controlModel(piFormula = ~ gender)
)
summary(pseudoHurdleModel)
# Assessing model fit
plot(pseudoHurdleModel, "rootogram")
summary(marginalFreq(pseudoHurdleModel), "group", df = 1)


# A advanced input with additional information for fitting procedure and
# additional formula specification and different link for inflation parameter.
Model <- estimatePopsize(
 formula = TOTAL_SUB ~ .,
 data = farmsubmission,
 model = oiztgeom(omegaLink = "cloglog"),
 method = "IRLS",
 controlMethod = controlMethod(
   stepsize = .85,
   momentumFactor = 1.2,
   epsilon = 1e-10,
   silent = TRUE
 ),
 controlModel = controlModel(omegaFormula = ~ C_TYPE + log_size)
)
summary(marginalFreq(Model), df = 18 - length(Model$coefficients))
summary(Model)
```

`estimatePopsizeFit` *Regression fitting in single-source capture-recapture models*

Description

`estimatePopsizeFit` does for `estimatePopsize` what `glm.fit` does for `glm`. It is internally called in `estimatePopsize`. Since `estimatePopsize` does much more than just regression fitting `estimatePopsizeFit` is much faster.

Usage

```
estimatePopsizeFit(
  y,
  X,
  family,
  control,
  method,
  priorWeights,
  coefStart,
  etaStart,
  offset,
  ...
)
```

Arguments

`y`	vector of dependent variables.
`X`	model matrix, the vglm one.
`family`	same as model in `estimatePopsize`.
`control`	control parameters created in `controlModel`.
`method`	method of estimation same as in `estimatePopsize`.
`priorWeights`	vector of prior weights its the same argument as weights in `estimatePopsize`.
`etaStart, coefStart`	initial value of regression parameters or linear predictors.
`offset`	offset passed from by default passed from `estimatePopsize()`.
`...`	arguments to pass to other methods.

Details

If `method` argument was set to `"optim"` the `stats::optim` function will be used to fit regression with analytically computed gradient and (minus) log likelihood functions as `gr` and `fn` arguments. Unfortunately `optim` does not allow for hessian to be specified. More information about how to modify `optim` fitting is included in `controlMethod()`.

If `method` argument was set to `"IRLS"` the iteratively reweighted least squares. The algorithm is well know in generalised linear models. Thomas W. Yee later extended this algorithm to vector generalised linear models and in more general terms it can roughly be described as (this is Yee's description after changing some conventions):

1. Initialize with:
   - `converged <- FALSE`
   - `iter <- 1`
   - $\boldsymbol{\beta}$ `<- start`
   - $\boldsymbol{W}$ `<- prior`

   - ℓ <- $\ell(\boldsymbol{\beta})$

2. If `converged` or `iter` > `Maxiter` move to step 7.
3. Store values from previous algorithm step:
   - $\boldsymbol{W}_-$ <- $\boldsymbol{W}$
   - ℓ_- <- ℓ
   - $\boldsymbol{\beta}_-$ <- $\boldsymbol{\beta}$

   and assign values at current step:
   - $\boldsymbol{\eta}$ <- $\boldsymbol{X}_{vlm}\boldsymbol{\beta}$
   - Z_i <- $\eta_i + \frac{\partial \ell_i}{\partial \eta_i}\mathbb{E}\left(\frac{\partial^2 \ell_i}{\partial \eta_i^T \partial \eta_i}\right)^{-1}$
   - $\boldsymbol{W}_{ij}$ <- $\mathbb{E}\left(\frac{\partial^2 \ell}{\partial \boldsymbol{\eta}_j^T \partial \boldsymbol{\eta}_i}\right)$

   where ℓ_i is the ith component of log likelihood function, η_i is the vector of linear predictors associated with ith row and $\mathbb{E}\left(\frac{\partial^2 \ell_i}{\partial \eta_i^T \partial \eta_i}\right)$ corresponds to weights associated with ith row and $\boldsymbol{W}$ is a block matrix, made of diagonal matrixes $\mathbb{E}\left(\frac{\partial^2 \ell}{\partial \boldsymbol{\eta}_j^T \partial \boldsymbol{\eta}_i}\right)$
4. Regress $\boldsymbol{Z}$ on $\boldsymbol{X}_{vlm}$ to obtain $\boldsymbol{\beta}$ as:

$$\boldsymbol{\beta} = \left(\boldsymbol{X}_{vlm}^T \boldsymbol{W} \boldsymbol{X}_{vlm}\right)^{-1} \boldsymbol{X}_{vlm}^T \boldsymbol{W} \boldsymbol{Z}$$

5. Assign:
   - `converged` <- $\ell(\boldsymbol{\beta}) - \ell_- < \varepsilon \cdot \ell_-$ or $||\boldsymbol{\beta} - \boldsymbol{\beta}_-||_\infty < \varepsilon$
   - `iter` <- `iter + 1`

   where ε is the relative tolerance level, by default `1e-8`.
6. Return to step 2.
7. Return $\boldsymbol{\beta}, \boldsymbol{W}$, `iter`.

In this package we use different conventions for $\boldsymbol{X}_{vlm}$ matrix hence slight differences are present in algorithm description but results are identical.

Value

List with regression parameters, working weights (if IRLS fitting method) was chosen and number of iterations taken.

Author(s)

Piotr Chlebicki, Maciej Beresewicz

References

Yee, T. W. (2015). Vector Generalized Linear and Additive Models: With an Implementation in R. New York, USA: Springer. ISBN 978-1-4939-2817-0.

See Also

`stats::glm()` `estimatePopsize()` `controlMethod()` `stats::optim()`

Examples

```
summary(farmsubmission)

# construct vglm model matrix
X <- matrix(data = 0, nrow = 2 * NROW(farmsubmission), ncol = 7)
X[1:NROW(farmsubmission), 1:4] <- model.matrix(
~ 1 + log_size + log_distance + C_TYPE,
farmsubmission
)


X[-(1:NROW(farmsubmission)), 5:7] <- X[1:NROW(farmsubmission), c(1, 3, 4)]

# this attribute tells the function which elements of the design matrix
# correspond to which linear predictor
attr(X, "hwm") <- c(4, 3)

# get starting points
start <- glm.fit(
y = farmsubmission$TOTAL_SUB,
x = X[1:NROW(farmsubmission), 1:4],
family = poisson()
)$coefficients

res <- estimatePopsizeFit(
y = farmsubmission$TOTAL_SUB,
X = X,
method = "IRLS",
priorWeights = 1,
family = ztoigeom(),
control = controlMethod(verbose = 5),
coefStart = c(start, 0, 0, 0),
etaStart = matrix(X %*% c(start, 0, 0, 0), ncol = 2),
offset = cbind(rep(0, NROW(farmsubmission)), rep(0, NROW(farmsubmission)))
)

# extract results

# regression coefficient vector
res$beta

# check likelihood
ll <- ztoigeom()$makeMinusLogLike(y = farmsubmission$TOTAL_SUB, X = X)

-ll(res$beta)

# number of iterations
res$iter

# working weights
head(res$weights)

# Compare with optim call

res2 <- estimatePopsizeFit(
  y = farmsubmission$TOTAL_SUB,
  X = X,
  method = "optim",
  priorWeights = 1,
  family = ztoigeom(),
  coefStart = c(start, 0, 0, 0),
  control = controlMethod(verbose = 1, silent = TRUE),
  offset = cbind(rep(0, NROW(farmsubmission)), rep(0, NROW(farmsubmission)))
)
# extract results

# regression coefficient vector
res2$beta


# check likelihood
-ll(res2$beta)

# number of calls to log lik function
# since optim does not return the number of
# iterations
res2$iter

# optim does not calculated working weights
head(res2$weights)
```

`farmsubmission` *British farm submissions data*

Description

Data on British animal farms submissions to AHVLA. British farms are able to submit samples to AHVLA if cause of death for an animal cannot be determined and private veterinary surgeon decides to submit them, unless there is notifiable disease suspected then such a submission is not required.

This data set contains information about such farms. All submissions from farms are included in this data frame not only carcasses but also blood samples etc.

Usage

```
data("farmsubmission")
```

Format

Data frame with 12,036 rows and 4 columns.

`TOTAL_SUB` Number of submissions of animal material.

`log_size` Numerical value equal to logarithm of size of farm.

`log_distance` Numerical value equal to logarithm of distance to nearest AHVLA center.

`C_TYPE` Factor describing type of activity on farm that animals are used for. Either `Dairy` or `Beef`

References

This data set and its description was provided in publication: Böhning, D., Vidal Diez, A., Lerdsuwansri, R., Viwatwongkasem, C., and Arnold, M. (2013). "A generalization of Chao's estimator for covariate information". *Biometrics*, 69(4), 1033-1042. doi:10.1111/biom.12082

`marginalFreq` *Observed and fitted marginal frequencies*

Description

A function that given a fitted `singleR` class object computed marginal frequencies by as sum of probability density functions for each unit in data at each point i.e. kth element of marginal frequency table is given by $\sum_{j=1}^{N_{obs}}\mathbb{P}(Y_{j}=k|\eta_{j})$. For k=0 only (if specified at call) they are computed as $\hat{N}-N_{obs}$ because $\boldsymbol{f}_{0}$ is assumed to the unobserved part of the studied population.

These frequencies are useful in diagnostics for count data regression, such as assessment of fit.

Usage

```
marginalFreq(
  object,
  includeones = TRUE,
  includezeros = TRUE,
  onecount = NULL,
  range,
  ...
)
```

Arguments

`object`	object of `singleR` class.
`includeones`	logical value indicating whether to include the estimated number of zero counts.
`includezeros`	logical value indicating whether to include one counts in the zero-one truncated models.
`onecount`	a numeric value indicating number of one counts if null `trcount` from object will be assumed to be a number one counts.
`range`	optional argument specifying range of selected Y values.
`...`	currently does nothing.

Value

A list with observed name of the fitted model family degrees of freedom and observed and fitted marginal frequencies.

Author(s)

Piotr Chlebicki

See Also

`estimatePopsize()` – where example of usage is provided

`netherlandsimmigrant` *Data on immigration in the Netherlands*

Description

This data set contains information about immigrants in four cities (Amsterdam, Rotterdam, The Hague and Utrecht) in Netherlands that have been staying in the country illegally in 1995 and have appeared in police records that year.

Usage

```
data("netherlandsimmigrant")
```

Format

Data frame with 1,880 rows and 5 columns.

`capture` Number of times a person has been captured by police.

`gender` Factor describing gender of the apprehended person.

`age` Factor describing age of apprehended person. Either bellow or above 40 years old.

`reason` Factor describing reason for being apprehended by police either illegal stay in Netherlands or other reasons.

`nation` Factor with nation of origin of the captured person. There are 6 levels of this variable: `"American and Australia"`, `"Asia"`, `"North Africa"`, `"Rest of Africa"`, `"Surinam"`, `"Turkey"`.

References

This data set and its description was provided in publication: van Der Heijden, P. G., Bustami, R., Cruyff, M. J., Engbersen, G., and Van Houwelingen, H. C. (2003). Point and interval estimation of the population size using the truncated Poisson regression model. *Statistical Modelling*, 3(4), 305-322. doi:10.1191/1471082X03st057oa

`plot.singleRStaticCountData` *Diagnostic plots for regression and population size estimation*

Description

Simple diagnostic plots for `singleRStaticCountData` class objects.

Usage

```
## S3 method for class 'singleRStaticCountData'
plot(
  x,
  plotType = c("qq", "marginal", "fitresid", "bootHist", "rootogram", "dfpopContr",
    "dfpopBox", "scaleLoc", "cooks", "hatplot", "strata"),
  confIntStrata = c("normal", "logNormal"),
  histKernels = TRUE,
  dfpop,
  ...
)
```

Arguments

`x`	object of `singleRStaticCountData` class.
`plotType`	character parameter (default `"qq"`) specifying type of plot to be made. The following list presents and briefly explains possible type of plots: • `qq` – The quantile-quantile plot for pearson residuals (or standardized pearson residuals if these are available for the model) i.e. empirical quantiles from residuals are plotted against theoretical quantiles from standard distribution. • `marginal` – A plot made by `matplot` with fitted and observed marginal frequencies with labels. • `fitresid` – Plot of fitted linear predictors against (standardized) pearson residuals. • `bootHist` – Simple histogram for statistics obtained from bootstrapping (if one was performed and the statistics were saved). • `rootogram` – Rootogram, for full explanation see: Kleiber and Zeileis Visualizing Count Data Regressions Using Rootograms (2016), in short it is a `barplot` where height is the square root of observed marginal frequencies adjusted by difference between square root of observed and fitted marginal frequencies connected by line representing fitted marginal frequencies. The less of a difference there is between the 0 line and beginning of a bar the more accurate fitt was produced by the model. • `dfpopContr` – Plot of `dfpopsize` against unit contribution. On the plot is y = x line i.e. what deletion effect would be if removing the unit from the model didn't effect regression coefficients. The further away the observation is from this line the more influential it is. • `dfpopBox` – Boxplot of `dfpopsize` for getting the general idea about the distribution of the "influence" of each unit on population size estimate. • `scaleLoc` – The scale - location plot i.e. square root of absolute values of (standardized) pearson residuals against linear predictors for each column of linear predictors. • `cooks` – Plot of cooks distance for detecting influential observations. • `hatplot` – Plot of hat values for each linear predictor for detecting influential observations. • `strata` – Plot of confidence intervals and point estimates for strata provided in `...` argument
`confIntStrata`	confidence interval type to use for strata plot. Currently supported values are `"normal"` and `"logNormal"`.
`histKernels`	logical value indicating whether to add density lines to histogram.
`dfpop`	TODO
`...`	additional optional arguments passed to the following functions: • For `plotType = "bootHist"` – `graphics::hist` – with `x, main, xlab, ylab` parameters fixed. • For `plotType = "rootogram"`

  - graphics::barplot – with height, offset, ylab, xlab, ylim parameters fixed.
  - graphics::lines – with x, y, pch, type, lwd, col parameters fixed.
- For plotType = "dfpopContr"
  - dfpopsize – with model, observedPop parameters fixed.
  - plot.default – with x, y, xlab, main parameters fixed.
- For plotType = "dfpopBox"
  - dfpopsize – with model, observedPop parameters fixed.
  - graphics::boxplot – with x, ylab, main parameters fixed.
- For plotType = "scaleLoc"
  - plot.default – with x, y, xlab, ylab, main, sub parameters fixed.
- For plotType = "fitresid"
  - plot.default – with x, y, xlab, ylab, main, sub parameters fixed.
- For plotType = "cooks"
  - plot.default – with x, xlab, ylab, main parameters fixed.
- For plotType = "hatplot"
  - hatvalues.singleRStaticCountData
  - plot.default – with x, xlab, ylab, main parameters fixed.
- For plotType = "strata"
  - stratifyPopsize.singleRStaticCountData

Value

No return value only the plot being made.

Author(s)

Piotr Chlebicki

See Also

estimatePopsize() dfpopsize() marginalFreq() stats::plot.lm() stats::cooks.distance() hatvalues.singleRStaticCountData()

---

popSizeEst *Extract population size estimation results*

Description

An extractor function with singleRStaticCountData method for extracting important information regarding pop size estimate.

Usage

```
popSizeEst(object, ...)

## S3 method for class 'singleRStaticCountData'
popSizeEst(object, ...)
```

Arguments

`object`	object with population size estimates.
`...`	additional optional arguments, currently not used in `singleRStaticCountData` class method.

Value

An object of class `popSizeEstResults` containing population size estimation results.

`predict.singleRStaticCountData`
Predict method for singleRStaticCountData class

Description

A method for `predict` function, works analogous to `predict.glm` but gives the possibility to get standard errors of mean/distribution parameters and directly get pop size estimates for new data.

Usage

```
## S3 method for class 'singleRStaticCountData'
predict(
  object,
  newdata,
  type = c("response", "link", "mean", "popSize", "contr"),
  se.fit = FALSE,
  na.action = NULL,
  weights,
  cov,
  ...
)
```

Arguments

`object` an object of `singleRStaticCountData` class.

`newdata` an optional `data.frame` containing new data.

`type` the type of prediction required, possible values are:

- `"response"`– For matrix containing estimated distributions parameters.
- `"link"` – For matrix of linear predictors.
- `"mean"` – For fitted values of both Y and $Y|Y > 0$.
- `"contr"` – For inverse probability weights (here named for observation contribution to population size estimate).
- `"popSize"` – For population size estimation. Note this results in a call to `redoPopEstimation` and it is usually better to call this function directly.

by default set to `"response"`.

`se.fit` a logical value indicating whether standard errors should be computed. Only matters for `type` in `"response"`, `"mean"`, `"link"`.

`na.action` does nothing yet.

weights	optional vector of weights for type in "contr", "popSize".
cov	optional matrix or function or character specifying either a covariance matrix or a function to compute that covariance matrix. By default vcov.singleRStaticCountData can be set to e.g. vcovHC.
...	arguments passed to other functions, for now this only affects vcov.singleRStaticCountData method and cov function.

Details

Standard errors are computed with assumption of regression coefficients being asymptotically normally distributed, if this assumption holds then each of linear predictors i.e. each row of $\boldsymbol{\eta} = \boldsymbol{X}_{vlm}\boldsymbol{\beta}$ is asymptotically normally distributed and their variances are expressed by well known formula. The mean μ and distribution parameters are then differentiable functions of asymptotically normally distributed variables and therefore their variances can be computed using (multivariate) delta method.

Value

Depending on type argument if one of "response", "link", "mean" a matrix with fitted values and possibly standard errors if se.fit argument was set to TRUE, if type was set to "contr" a vector with inverses of probabilities, finally for "popSize" an object of class popSizeEstResults with its own methods containing population size estimation results.

See Also

redoPopEstimation() stats::summary.glm() estimatePopsize()

---

redoPopEstimation *Updating population size estimation results*

Description

A function that applies all post-hoc procedures that were taken (such as heteroscedastic consistent covariance matrix estimation or bias reduction) to population size estimation and standard error estimation.

Usage

```
redoPopEstimation(object, newdata, ...)

## S3 method for class 'singleRStaticCountData'
redoPopEstimation(
  object,
  newdata,
  cov,
  weights,
  coef,
  control,
  popVar,
  offset,
  weightsAsCounts,
  ...
)
```

Arguments

object	object for which update of population size estimation results will be done.
newdata	optional `data.frame` with new data for pop size estimation.
...	additional optional arguments, currently not used in `singleRStaticCountData` class method.
cov	an updated covariance matrix estimate.
weights	optional vector of weights to use in population size estimation.
coef	optional vector of coefficients of regression on which to base population size estimation. If missing it is set to `coef(object)`.
control	similar to `controlPopVar` in `estimatePopsize()`. If missing set to controls provided on call to `object`.
popVar	similar to `popVar` in `estimatePopsize()`. If missing set to `"analytic"`.
offset	offset argument for new data
weightsAsCounts	for `singleRStaticCountData` method used to specify whether weights should be treated as number of occurrences for rows in data

Details

Any non specified arguments will be inferred from the `object`

Value

An object of class `popSizeEstResults` containing updated population size estimation results.

Examples

```
# Create simple model
Model <- estimatePopsize(
  formula = capture ~ nation + gender,
  data = netherlandsimmigrant,
  model = ztpoisson,
  method = "IRLS"
)
# Apply heteroscedasticity consistent covariance matrix estimation
require(sandwich)
cov <- vcovHC(Model, type = "HC3")
summary(Model, cov = cov,
popSizeEst = redoPopEstimation(Model, cov = cov))
# Compare to results with usual covariance matrix estimation
summary(Model)

## get confidence interval with larger significance level
redoPopEstimation(Model, control = controlPopVar(alpha = .000001))
```

regDiagSingleR *Regression diagnostics in singleRcapture*

Description

List of some regression diagnostics implemented for `singleRStaticCountData` class. Functions that either require no changes from `glm` class or are not relevant to context of `singleRcapture` are omitted.

Usage

```
dfpopsize(model, ...)

## S3 method for class 'singleRStaticCountData'
dfpopsize(model, dfbeta = NULL, ...)

## S3 method for class 'singleRStaticCountData'
dfbeta(model, maxitNew = 1, trace = FALSE, cores = 1, ...)

## S3 method for class 'singleRStaticCountData'
hatvalues(model, ...)

## S3 method for class 'singleRStaticCountData'
residuals(
  object,
  type = c("pearson", "pearsonSTD", "response", "working", "deviance", "all"),
  ...
)

## S3 method for class 'singleRStaticCountData'
cooks.distance(model, ...)

## S3 method for class 'singleRStaticCountData'
sigma(object, ...)

## S3 method for class 'singleRStaticCountData'
influence(model, do.coef = FALSE, ...)

## S3 method for class 'singleRStaticCountData'
rstudent(model, ...)

## S3 method for class 'singleRStaticCountData'
rstandard(model, type = c("deviance", "pearson"), ...)
```

Arguments

`model, object`	an object of `singleRStaticCountData` class.
`...`	arguments passed to other methods. Notably `dfpopsize.singleRStaticCountData` calls `dfbeta.singleRStaticCountData` if no `dfbeta` argument was provided and `controlMethod` is called in `dfbeta` method.

`dfbeta`	if `dfbeta` was already obtained it is possible to pass them into function so that they need not be computed for the second time.
`maxitNew`	the maximal number of iterations for regressions with starting points $\hat{\boldsymbol{\beta}}$ on data specified at call for `model` after the removal of k'th row. By default 1.
`trace`	a logical value specifying whether to tracking results when `cores` > 1 it will result in a progress bar being created.
`cores`	a number of processor cores to be used, any number greater than 1 activates code designed with `doParallel`, `foreach` and `parallel` packages. Note that for now using parallel computing makes tracing impossible so `trace` parameter is ignored in this case.
`type`	a type of residual to return.
`do.coef`	logical indicating if `dfbeta` computation for influence should be done. `FALSE` by default.

Details

`dfpopsize` and `dfbeta` are closely related. `dfbeta` fits a regression after removing a specific row from the data and returns the difference between regression coefficients estimated on full data set and data set obtained after deletion of that row, and repeats procedure once for every unit present in the data.`dfpopsize` does the same for population size estimation utilizing coefficients computed by `dfbeta`.

`cooks.distance` is implemented (for now) only for models with a single linear predictor and works exactly like the method for `glm` class.

`sigma` computes the standard errors of predicted means. Returns a matrix with two columns first for truncated mean and the other for the non-truncated mean.

`residuals.singleRStaticCountData` (can be abbreviated to `resid`) works like `residuals.glm` with the exception that:

- `"pearson"` – returns non standardized residuals.
- `"pearsonSTD"` – is currently defined only for single predictors models but will be extended to all models in a near future, but for families with more than one distribution parameter it will be a multivariate residual.
- `"response"` – returns both residuals computed with truncated and non truncated fitted value.
- `"working"` – is possibly multivariate if more than one linear predictor is present.
- `"deviance"` – is not yet defined for all families in `singleRmodels()` e.g. negative binomial based methods.
- `"all"` – returns all available residual types.

`hatvalues.singleRStaticCountData` is method for `singleRStaticCountData` class for extracting diagonal elements of projection matrix.

Since `singleRcapture` supports not only regular glm's but also vglm's the `hatvalues` returns a matrix with number of columns corresponding to number of linear predictors in a model, where kth column corresponds to elements of the diagonal of projection matrix associated with kth linear predictor. For glm's

$$\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{X}\left(\boldsymbol{X}^{T}\boldsymbol{W}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{T}\boldsymbol{W}^{\frac{1}{2}}$$

where: $\boldsymbol{W} = \mathbb{E}\left(\text{Diag}\left(\frac{\partial^{2}\ell}{\partial\boldsymbol{\eta}^{T}\partial\boldsymbol{\eta}}\right)\right)$ and $\boldsymbol{X}$ is a model (lm) matrix. For vglm's present in the package it is instead :

$$\boldsymbol{X}_{vlm}\left(\boldsymbol{X}_{vlm}^{T}\boldsymbol{W}\boldsymbol{X}_{vlm}\right)^{-1}\boldsymbol{X}_{vlm}^{T}\boldsymbol{W}$$

where:

$$\boldsymbol{W} = \mathbb{E}\left(\begin{bmatrix} \text{Diag}\left(\frac{\partial^2\ell}{\partial\eta_1^T\partial\eta_1}\right) & \text{Diag}\left(\frac{\partial^2\ell}{\partial\eta_1^T\partial\eta_2}\right) & \dots & \text{Diag}\left(\frac{\partial^2\ell}{\partial\eta_1^T\partial\eta_p}\right) \\ \text{Diag}\left(\frac{\partial^2\ell}{\partial\eta_2^T\partial\eta_1}\right) & \text{Diag}\left(\frac{\partial^2\ell}{\partial\eta_2^T\partial\eta_2}\right) & \dots & \text{Diag}\left(\frac{\partial^2\ell}{\partial\eta_2^T\partial\eta_p}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Diag}\left(\frac{\partial^2\ell}{\partial\eta_p^T\partial\eta_1}\right) & \text{Diag}\left(\frac{\partial^2\ell}{\partial\eta_p^T\partial\eta_2}\right) & \dots & \text{Diag}\left(\frac{\partial^2\ell}{\partial\eta_p^T\partial\eta_p}\right) \end{bmatrix}\right)$$

is a block matrix constructed by taking the expected value from diagonal matrixes corresponding to second derivatives with respect to each linear predictor (and mixed derivatives) and $\boldsymbol{X}_{vlm}$ is a model (vlm) matrix constructed using specifications in `controlModel` and call to `estimatePopsize`.

`influence` works like `glm` counterpart computing the most important influence measures.

Value

- For `hatvalues` – A matrix with n rows and p columns where n is a number of observations in the data and p is number of regression parameters.
- For `dfpopsize` – A vector for which k'th element corresponds to the difference between point estimate of population size estimation on full data set and point estimate of population size estimation after the removal of k'th unit from the data set.
- For `dfbeta` – A matrix with n rows and p observations where p is a number of units in data and p is the number of regression parameters. K'th row of this matrix corresponds to $\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}_{-k}$ where $\hat{\boldsymbol{\beta}}_{-k}$ is a vector of estimates for regression parameters after the removal of k'th row from the data.
- `cooks.distance` – A matrix with a single columns with values of cooks distance for every unit in `model.matrix`
- `residuals.singleRStaticCountData` – A `data.frame` with chosen residuals.

Author(s)

Piotr Chlebicki, Maciej Beręsewicz

See Also

`estimatePopsize()` `stats::hatvalues()` `controlMethod()` `stats::dfbeta()` `stats::cooks.distance()`

Examples

```
# For singleRStaticCountData class
# Get simple model
Model <- estimatePopsize(
  formula = capture ~ nation + age + gender,
  data = netherlandsimmigrant,
  model = ztpoisson,
  method = "IRLS"
)
# Get dfbeta
dfb <- dfbeta(Model)
# The dfpopsize results are obtained via (It is also possible to not provide
# dfbeta then they will be computed manually):
res <- dfpopsize(Model, dfbeta = dfb)
summary(res)
plot(res)
# see vaious types of residuals:
```

```
head(resid(Model, "all"))
```

simulate *Generating data in singleRcapture*

Description

An S3 method for `stats::simulate` to handle `singleRStaticCountData` and `singleRfamily` classes.

Usage

```
## S3 method for class 'singleRStaticCountData'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'singleRfamily'
simulate(object, nsim, seed = NULL, eta, truncated = FALSE, ...)
```

Arguments

`object`	an object representing a fitted model.
`nsim`	a numeric scalar specifying: • number of response vectors to simulate in `simulate.singleRStaticCountData`, defaults to `1L`. • number of units to draw in `simulate.singleRfamily`, defaults to `NROW(eta)`.
`seed`	an object specifying if and how the random number generator should be initialized (‘seeded’).
`...`	additional optional arguments.
`eta`	a matrix of linear predictors
`truncated`	logical value indicating whether to sample from truncated or full distribution.

Value

a `data.frame` with `n` rows and `nsim` columns.

Author(s)

Maciej Beręsewicz, Piotr Chlebicki

See Also

`stats::simulate()` `estimatePopsize()`

Examples

```
N <- 10000
###gender <- rbinom(N, 1, 0.2)
gender <- rep(0:1, c(8042, 1958))
eta <- -1 + 0.5*gender
counts <- simulate(ztpoisson(), eta = cbind(eta), seed = 1)
df <- data.frame(gender, eta, counts)
df2 <- subset(df, counts > 0)
### check coverage with summary
mod1 <-  estimatePopsize(
  formula       = counts ~ 1 + gender,
  data          = df2,
  model         = ztpoisson,
  controlMethod = list(silent = TRUE)
)
mod1_sims <- simulate(mod1, nsim=10, seed = 1)
colMeans(mod1_sims)
mean(df2$counts)
```

`stratifyPopsize` *Estimate size of sub populations.*

Description

A function that estimates sizes of specific sub populations based on a capture-recapture model for the whole population.

Usage

```
stratifyPopsize(object, strata, alpha, ...)

## S3 method for class 'singleRStaticCountData'
stratifyPopsize(object, strata, alpha, cov = NULL, ...)
```

Arguments

`object` an object on which the population size estimates should be based in `singleRcapture` package this is a fitter `singleRStaticCountData` class object.

`strata` a specification of sub populations given by one of:

- formula – a formula to be applied to `model.frame` extracted from the object.
- Logical vector with number of entries equal to number of rows in the dataset.
- A (named) list where each element is a logical vector, names of the list will be used to specify names variable in returned object.
- Vector of names of explanatory variables. For `singleRStaticCountData` method for this function this specification of `strata` parameter will result in every level of explanatory variable having its own sub population for each variable specified.
- If no value was provided the `singleRStaticCountData` method for this function will itself create sub populations based on levels of factor variables in `model.frame`.

`alpha`	significance level for confidence intervals – Either a single numeric value or a vector of length equal to number of sub populations specified in `strata`. If missing it is set to `.05` in `singleRStaticCountData` method.
`...`	a vector of arguments to be passed to other functions. For `singleRStaticCountData` method for this functions arguments in `...` are passed to either `cov` if argument provided was a function or `vcov` if `cov` argument was missing at call.
`cov`	for `singleRStaticCountData` method an estimate of variance-covariance matrix for estimate of regression parameters. It is possible to pass a function such as for example `sandwich::vcovHC` which will be called as: `foo(object, ...)` and a user may specify additional arguments of a function in `...` argument. If not provided an estimate for covariance matrix will be set by calling appropriate `vcov` method.

Details

In single source capture-recapture models the most frequently used estimate for population size is Horvitz-Thompson type estimate:

$$\hat{N} = \sum_{k=1}^{N}\frac{I_{k}}{\mathbb{P}(Y_{k}>0)} = \sum_{k=1}^{N_{obs}}\frac{1}{1-\mathbb{P}(Y_{k}=0)}$$

where $I_{k}=I_{Y_{k} > 0}$ are indicator variables, with value 1 if kth unit was observed at least once and 0 otherwise and the inverse probabilistic weights weights for units observed in the data $\tfrac{1}{\mathbb{P}(Y_{k}>0)}$ are estimated using fitted linear predictors.

The estimates for different sub populations are made by changing the $I_{k}=I_{Y_{k} > 0}$ indicator variables to refer not to the population as a whole but to the sub populations that are being considered i.e. by changing values from 1 to 0 if kth unit is not a member of sub population that is being considered at the moment.

The estimation of variance for these estimates and estimation of variance for estimate of population size for the whole population follow the same relation as the one described above.

Value

A `data.frame` object with row names being the names of specified sub populations either provided or inferred.

See Also

`vcov.singleRStaticCountData()` `estimatePopsize()`

`summary.singleRmargin` *Statistical tests of goodness of fit*

Description

Performs two statistical test on observed and fitted marginal frequencies. For G test the test statistic is computed as:

$$G = 2\sum_{k} O_{k}\ln\left(\frac{O_{k}}{E_{k}}\right)$$

and for χ^2 the test statistic is computed as:

$$\chi^{2} = \sum_{k}\frac{\left(O_{k}-E_{k}\right)^{2}}{E_{k}}$$

where O_k, E_k denoted observed and fitted frequencies respectively. Both of these statistics converge to χ^2 distribution asymptotically with the same degrees of freedom.

The convergence of G, χ^2 statistics to χ^2 distribution may be violated if expected counts in cells are too low, say < 5, so it is customary to either censor or omit these cells.

Usage

```
## S3 method for class 'singleRmargin'
summary(object, df, dropl5 = c("drop", "group", "no"), ...)
```

Arguments

`object`	object of singleRmargin class.
`df`	degrees of freedom if not provided the function will try and manually but it is not always possible.
`dropl5`	a character indicating treatment of cells with frequencies < 5 either grouping them, dropping or leaving them as is. Defaults to `drop`.
`...`	currently does nothing.

Value

A chi squared test and G test for comparison between fitted and observed marginal frequencies.

Examples

```
# Create a simple model
Model <- estimatePopsize(
  formula = capture ~ .,
  data = netherlandsimmigrant,
  model = ztpoisson,
  method = "IRLS"
)
plot(Model, "rootogram")
# We see a considerable lack of fit
summary(marginalFreq(Model), df = 1, dropl5 = "group")
```

summary.singleRStaticCountData

Summary statistics for model of singleRStaticCountData class.

Description

A `summary` method for `singleRStaticCountData` class

Usage

```
## S3 method for class 'singleRStaticCountData'
summary(
  object,
  test = c("t", "z"),
  resType = "pearson",
  correlation = FALSE,
  confint = FALSE,
  cov,
  popSizeEst,
  ...
)
```

Arguments

`object`	object of singleRStaticCountData class.
`test`	type of test for significance of parameters `"t"` for t-test and `"z"` for normal approximation of students t distribution, by default `"z"` is used if there are more than 30 degrees of freedom and `"t"` is used in other cases.
`resType`	type of residuals to summarize any value that is allowed in `residuals.singleRStaticCountData` except for `"all"` is allowed. By default pearson residuals are used.
`correlation`	logical value indicating whether correlation matrix should be computed from covariance matrix by default `FALSE`.
`confint`	logical value indicating whether confidence intervals for regression parameters should be constructed. By default `FALSE`.
`cov`	covariance matrix corresponding to regression parameters. It is possible to give `cov` argument as a function of `object`. If not specified it will be constructed using `vcov.singleRStaticCountData` method. (i.e using Cramer-Rao lower bound)
`popSizeEst`	a `popSizeEstResults` class object. If not specified population size estimation results will be drawn from `object`. If any post-hoc procedures, such as sandwich covariance matrix estimation or bias reduction, were taken it is possible to include them in population size estimation results by calling `redoPopEstimation`.
`...`	additional optional arguments passed to the following functions: • `vcov.singleRStaticCountData` – if no `cov` argument was provided. • `cov` – if `cov` parameter specified at call was a function. • `confint.singleRStaticCountData` – if `confint` parameter was set to `TRUE` at function call. In particular it is possible to set confidence level in `...`.

Details

Works analogically to `summary.glm` but includes population size estimation results. If any additional statistics, such as confidence intervals for coefficients or coefficient correlation, are specified they will be printed.

Value

An object of `summarysingleRStaticCountData` class containing:

- `call` – A call which created `object`.
- `coefficients` – A dataframe with estimated regression coefficients and their summary statistics such as standard error Wald test statistic and p value for Wald test.
- `residuals` – A vector of residuals of type specified at call.
- `aic` – Akaike's information criterion.
- `bic` – Bayesian (Schwarz's) information criterion.
- `iter` – Number of iterations taken in fitting regression.
- `logL` – Logarithm of likelihood function evaluated at coefficients.
- `deviance` – Residual deviance.
- `populationSize` – Object with population size estimation results.
- `dfResidual` – Residual degrees of freedom.
- `sizeObserved` – Size of observed population.
- `correlation` – Correlation matrix if `correlation` parameter was set to `TRUE`
- `test` – Type of statistical test performed.
- `model` – Family class object specified in call for `object`.
- `skew` – If bootstrap sample was saved contains estimate of skewness.

See Also

`redoPopEstimation()` `stats::summary.glm()`

`vcov.singleRStaticCountData` *Obtain Covariance Matrix estimation*

Description

A `vcov` method for `singleRStaticCountData` class.

Usage

```
## S3 method for class 'singleRStaticCountData'
vcov(object, type = c("Fisher", "observedInform"), ...)
```

Arguments

`object`	object of singleRStaticCountData class.
`type`	type of estimate for covariance matrix for now either expected (Fisher) information matrix or observed information matrix.
`...`	additional arguments for method functions

Details

Returns a estimated covariance matrix for model coefficients calculated from analytic hessian or Fisher information matrix usually utilizing asymptotic effectiveness of maximum likelihood estimates. Covariance type is taken from control parameter that have been provided on call that created `object` if arguments `type` was not specified.

Value

A covariance matrix for fitted coefficients, rows and columns of which correspond to parameters returned by `coef` method.

See Also

`vcovHC.singleRStaticCountData()` `sandwich::sandwich()`

Index