Package ‘cplm’

March 5, 2019

Type Package

Title Compound Poisson Linear Models

Version 0.7-8

Author Yanwei (Wayne) Zhang

Maintainer Yanwei (Wayne) Zhang <actuary_zhang@hotmail.com>

Description Likelihood-based and Bayesian methods for various compound Poisson linear models based on Zhang, Yanwei (2013) <https://link.springer.com/article/10.1007/s11222-012-9343-7>.

Imports biglm, ggplot2, minqa, nlme, reshape2, statmod, stats, stats4, tweedie

Depends R (>= 3.2.0), coda, Matrix, splines, methods

LinkingTo Matrix

License GPL (>= 2)

URL https://github.com/actuaryzhang/cplm

LazyLoad yes

LazyData yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-03-05 10:50:03 UTC

R topics documented:

`cplm-package` *Tweedie compound Poisson linear models*

Description

The Tweedie compound Poisson distribution is a mixture of a degenerate distribution at the origin and a continuous distribution on the positive real line. It has been applied in a wide range of fields in which continuous data with exact zeros regularly arise. Nevertheless, statistical inference based on full likelihood and Bayesian methods is not available in most statistical software, largely because the distribution has an intractable density function and numerical methods that allow fast and accurate evaluation of the density did not appear until fairly recently. The `cplm` package provides likelihood-based and Bayesian procedures for fitting common Tweedie compound Poisson linear models. In particular, models with hierarchical structures or extra zero inflation can be handled. Further, the package implements the Gini index based on an ordered version of the Lorenz curve as a robust model comparison tool involving zero-inflated and highly skewed distributions.

The following features of the package may be of special interest to the users:

1. All methods available in the package enable the index parameter (i.e., the unknown variance function) to be estimated from the data.
2. The compound Poisson generalized linear model handles large data set using the bounded memory regression facility in `biglm`.
3. For mixed models, we provide likelihood-based methods using Laplace approximation and adaptive Gauss-Hermit quadrature.
4. A convenient interface is offered to fit additive models (penalized splines) using the mixed model estimation procedure.
5. Self-tuned Markov chain Monte Carlo procedures are available for both GLM-type and mixed models.
6. The package also implements a zero-inflated compound Poisson model, in which the observed frequency of zeros can generally be more adequately modeled.
7. We provide the Gini index based on an ordered Lorenz curve, which is better suited for model comparison involving the compound Poisson distribution.

More information is available on the hosting web site of the project `http://code.google.com/p/cplm/`

Author(s)

Yanwei (Wayne) Zhang <actuary_zhang@hotmail.com>

References

Dunn, P.K. and Smyth, G.K. (2005). Series evaluation of Tweedie exponential dispersion models densities. Statistics and Computing, *15, 267-280.*

Frees, E. W., Meyers, G. and Cummings, D. A. (2011). Summarizing Insurance Scores Using a Gini Index. Journal of the American Statistical Association, *495, 1085 - 1098.*

Zhang, Y (2013). Likelihood-based and Bayesian Methods for Tweedie Compound Poisson Linear Mixed Models, Statistics and Computing, *23, 743-757.*

`bcplm` *Bayesian Compound Poisson Linear Models*

Description

This function fits Tweedie compound Poisson linear models using Markov Chain Monte Carlo methods.

Usage

```
bcplm(formula, link = "log", data, inits = NULL,
  weights, offset, subset, na.action, contrasts = NULL,
  n.chains = 3, n.iter = 2000, n.burnin = floor(n.iter / 2),
  n.thin = max(1, floor(n.chains * (n.iter - n.burnin) / n.sims)),
  n.sims = 1000, n.report = 2, prior.beta.mean = NULL,
  prior.beta.var = NULL, bound.phi = 100, bound.p = c(1.01, 1.99),
  tune.iter = 5000, n.tune = floor(tune.iter/100),
  basisGenerators = c("tp", "bsp", "sp2d"), doFit = TRUE, ...)
```

Arguments

`formula`	an object of class `formula`. See `glm` and `cpglmm` for details.
`link`	a specification for the model link function. This can be either a literal character string or a numeric number. If it is a character string, it must be one of "log", "identity", "sqrt" or "inverse". If it is numeric, it is the same as the `link.power` argument in the `tweedie` function. The default is `link = "log"`.
`inits`	a list of initial values to be used for each chain. It must be of length `n.chains`. Each element is a named list with the following components: 'beta' (fixed effects), 'phi' (dispersion), and 'p' (index parameter). If the formula indicates a mixed model, it must also contain two additional members 'u' (random effects) and 'Sigma' (variance components). 'Sigma' must be a list of the same format as the `ST` slot in `cpglmm`. If not supplied, the function will generate initial values automatically.
`data, subset, weights, na.action, offset, contrasts`	further model specification arguments as in `cpglm`; see there for details.

`n.chains`	an integer indicating the number of Markov chains (default: `3`).
`n.iter`	the number of total iterations per chain (including burn in; default: `2000`)
`n.burnin`	the length of burn in, i.e. number of iterations to discard at the beginning. Default is `n.iter/2`, that is, discarding the first half of the simulations.
`n.thin`	thinning rate. Must be a positive integer. Set `n.thin > 1` to save memory and computation time if `n.iter` is large. Default is `max(1, floor(n.chains * (n.iter - n.burnin) / 10` which will only thin if there are at least `2000` simulations.
`n.sims`	The approximate number of simulations to keep after thinning (all chains combined).
`n.report`	if greater than zero, fitting information will be printed out `n.report` times for each chain.
`prior.beta.mean`	a vector of prior means for the fixed effects. Default is a vector of zeros.
`prior.beta.var`	a vector of prior variances for the fixed effects. Default is a vector of `10000`'s.
`bound.phi`	a numeric value indicating the upper bound of the uniform prior for the dispersion parameter. The default is `100`. The lower bound is set to be `0` in the function.
`bound.p`	a vector of lower and upper bounds for the index parameter p. The default is `c(1.01, 1.99)`.
`tune.iter`	the number of iterations used for tuning the proposal variances used in the Metropolis-Hastings updates. These iterations will not be included in the final output. Default is `5000`. Set it to be zero if the tuning process is not desired.
`n.tune`	a positive integer (default: `20`). The `tune.iter` iterations is divided into `n.tune` loops. Proposal variances are updated at the end of each loop if acceptance rates are outside the desired interval.
`basisGenerators`	a character vector of names of functions that generate spline bases. See `tp` for details.
`doFit`	if `FALSE`, the constructed `"bcplm_input"` object is returned before the model is fitted.
`...`	not used.

Details

This function provides Markov chain Monte Carlo [MCMC] methods for fitting Tweedie compound Poisson linear models within the Bayesian framework. Both generalized linear models and mixed models can be handled. In computing the posterior distribution, the series evaluation method (see, e.g., `dtweedie`) is employed to evaluate the compound Poisson density.

In the Bayesian model, prior distributions have to be specified for all parameters in the model. Here, Normal distributions are used for the fixed effects (β), a Uniform distribution for the dispersion parameter (ϕ), a Uniform distribution for the index parameter (p). If a mixed model is specified, prior distributions must be specified for the variance component. If there is one random effect in a group, the inverse Gamma (scale = `0.001`, shape = `0.001`) is specified as the prior. If there is more than one random effects in a group, the inverse Wishart (identity matrix as the scale and the dimension of the covariance matrix as the shape) is specified as the prior.

Prior means and variances of the fixed effects can be supplied using the argument `prior.beta.mean` and `prior.beta.var`, respectively. The prior distribution of ϕ is uniform on (0, `bound.phi`). And the bounds of the Uniform for p can be specified in the argument `bound.p`. See details in section 'Arguments'.

In implementing the MCMC, a Gibbs sampler is constructed in which parameters are updated one at a time given the current values of all the other parameters. Specifically, we use the random-walk Metropolis-Hastings algorithm in updating each parameter except for the variance components, which can be simulated directly due to conjugacy.

Before the MCMC, there is a tuning process where the proposal variances of the (truncated) Normal proposal distributions are updated according to the sample variances computed from the simulations in each tuning loop. The goal is to make the acceptance rate roughly between 40% and 60% for univariate M-H updates. The argument `tune.iter` determines how many iterations are used for the tuning process, and `n.tune` determines how many loops these iterations should be divided into. These iterations will not be used in the final output.

The simulated values of all model parameters are stored in the `sims.list` slot of the returned `bcplm` object. It is a list of `n.chains` matrices and each matrix has approximately `n.sims` rows. The `sims.list` slot is further coerced to be of class `"mcmc.list"` so that various methods from the `coda` package can be directly applied to get Markov chain diagnostics, posterior summary and plots. See `coda` for available methods.

Value

`bcplm` returns an object of class `"bcplm"`. See `bcplm-class` for details of the return values as well as various methods available for this class.

Author(s)

Yanwei (Wayne) Zhang `<actuary_zhang@hotmail.com>`

References

Zhang, Y (2013). Likelihood-based and Bayesian Methods for Tweedie Compound Poisson Linear Mixed Models, Statistics and Computing, *23, 743-757.*

See Also

The users are recommended to see the documentation for `bcplm-class`, `cpglm`, `cpglmm`, `mcmc`, and `tweedie` for related information.

Examples

```
## Not run:

# fit the FineRoot data with Bayesian models
# Bayesian cpglm
set.seed(10)
fit1 <- bcplm(RLD ~ factor(Zone) * factor(Stock),
            data = FineRoot, tune.iter = 2000,
            n.iter = 6000, n.burnin = 1000, n.thin = 5)

gelman.diag(fit1$sims.list)
# diagnostic plots
acfplot(fit1$sims.list, lag.max = 20)
xyplot(fit1$sims.list)
densityplot(fit1$sims.list)
summary(fit1)
plot(fit1)


# now fit the Bayesian model to an insurance loss triangle
# (see Peters et al. 2009)
fit2 <- bcplm(increLoss ~ factor(year) + factor(lag),
            data = ClaimTriangle, n.iter = 12000,
            n.burnin = 2000, n.thin = 10, bound.p = c(1.1, 1.95))
gelman.diag(fit2$sims.list)
summary(fit2)

# mixed models
set.seed(10)
fit3 <- bcplm(RLD ~ Stock * Zone + (1|Plant),
            data = FineRoot, n.iter = 15000,
            n.burnin = 5000, n.thin = 10)
gelman.diag(fit3$sims.list)
summary(fit3)




## End(Not run)
```

`class-methods` *Classes and Methods for a Compound Poisson Linear Model Object*

Description

Documented here are the `"cplm"` class and its derived classes `"cpglm"`, `"cpglmm"`, `"bcplm"` and `"zcpglm"`. Several primitive methods and statistical methods are created to facilitate the extraction of specific slots and further statistical analysis. `"gini"` is a class that stores the Gini indices and associated standard errors that could be used to perform model comparison involving the compound Poisson distribution. `"NullNum"`, `"NullList"`, `"NullFunc"` and `"ListFrame"` are virtual classes for `c("NULL", "numeric")`, `c("NULL","list")`, `c("NULL","function")` and `c("list","data.frame")`, respectively.

Objects from the Class

`"cplm"` Objects can be created by calls of the form `new("cplm", ...)`.

`"cpglm"` Objects can be created by calls from `new("cpglm", ...)` or `cpglm`.

`"cpglmm"` Objects can be created by calls of the form `new("cpglmm", ...)`, or a call to `cpglmm`.

`"summary.cpglmm"` Objects can be created by calls of the form `new("summary.cpglmm", ...)`, or a call to `summary` on a `cpglmm` object.

`"bcplm"` Objects can be created by calls from `new("bcplm", ...)` or `bcplm`.

`"zcpglm"` Objects can be created by calls from `new("zcpglm", ...)` or `zcpglm`.

`"gini"` Objects can be created by calls from `new("gini", ...)` or `gini`.

`"NullNum"`, `"NullList"`, `"NullFunc"` These are virtual classes and no objects may be created from them.

Slots

The `"cplm"` class defines the slots common in all the model classes in the `cplm` package, and thus the utility methods defined on the `"cplm"` class such as `[`, `names` and so on are applicable to all of the derived classes.

`call`**:** the matched call.

`formula`**:** the formula supplied, class `"formula"`

`contrasts`**:** the contrasts used, class `"NullList"`

`link.power`**:** index of power link function, class `"numeric"`. See `tweedie`.

`model.frame`**:** the data frame used. class `"ListFrame"`.

`inits`**:** initial values used, class `"NullList"`.

The `"cpglm"` class extends `"cplm"` directly. Most of the slots have the same definition as those in `glm`. The following slots are in addition to those in `"cplm"`:

`coefficients`**:** estimated mean parameters, class `"numeric"`.

`residuals`**:** the working residuals, that is the residuals in the final iteration of the IWLS fit, class `"numeric"`

`fitted.values`**:** the fitted mean values, obtained by transforming the linear predictors by the inverse of the link function, class `"numeric"`

`linear.predictors`**:** the fitted linear predictors, class `"numeric"`

`weights`**:** working weights from the last iteration of the iterative least square, class `"numeric"`

`df.residual`**:** residual degrees of freedom, class `"integer"`

`deviance`**:** up to a constant, minus twice the maximized log-likelihood. Where sensible, the constant is chosen so that a saturated model has deviance zero. This is computed using `tweedie.dev`.

`aic`**:** a version of Akaike's Information Criterion, minus twice the maximized log-likelihood plus twice the number of mean parameters. This is computed using the tweedie density approximation as in `dtweedie`.

`offset`**:** the offset vector used, class `"NullNum"`,

`prior.weights`**:** the weights initially supplied, a vector of `1`s if none were, class `"NullNum"`

`y`**:** the response vector used.

`control`**:** the value of the control argument used, class `"list"`

`p`**:** the maximum likelihood estimate of the index parameter.

`phi`**:** the maximum likelihood estimate of the dispersion parameter.

`vcov`**:** estimated variance-covariance matrix, class `"matrix"`

`iter`**:** the number of Fisher's scoring iterations in the final GLM.

`converged`**:** indicating whether the algorithm has converged, class `"logical"`.

`na.action`**:** method of handling `NA`'s, class `"NullFunc"`.

The `"cpglmm"` class extends `"cplm"` and the old version of `"mer"` class from `lme4` directly, and has the following additional slots:

`p`**:** estimated value of the index parameter, class `"numeric"`

`phi`**:** estimated value of the dispersion parameter, class `"numeric"`

`bound.p`**:** the specified bounds of the index parameter, class `"numeric"`

`vcov`**:** estimated variance-covariance matrix, class `"matrix"`

`smooths`**:** a list of smooth terms

The slots it used from the old `"mer"` class has the following slots (copied from `lme4_0.999999-2`):

`env`**:** An environment (class `"environment"`) created for the evaluation of the nonlinear model function.

`nlmodel`**:** The nonlinear model function as an object of class `"call"`.

`frame`**:** The model frame (class `"data.frame"`).

`call`**:** The matched call to the function that created the object. (class `"call"`).

`flist`**:** The list of grouping factors for the random effects.

`X`**:** Model matrix for the fixed effects.

`Zt`**:** The transpose of model matrix for the random effects, stored as a compressed column-oriented sparse matrix (class `"dgCMatrix"`).

`pWt`**:** Numeric prior weights vector. This may be of length zero (0), indicating unit prior weights.

`offset`**:** Numeric offset vector. This may be of length zero (0), indicating no offset.

`y`**:** The response vector (class `"numeric"`).

`Gp`**:** Integer vector of group pointers within the random effects vector. The elements of `Gp` are 0-based indices of the first element from each random-effects term. Thus the first element is always 0. The last element is the total length of the random effects vector.

`dims`**:** A named integer vector of dimensions. Some of the dimensions are n, the number of observations, p, the number of fixed effects, q, the total number of random effects, s, the number of parameters in the nonlinear model function and nt, the number of random-effects terms in the model.

`ST`**:** A list of S and T factors in the TSST' Cholesky factorization of the relative variance matrices of the random effects associated with each random-effects term. The unit lower triangular matrix, T, and the diagonal matrix, S, for each term are stored as a single matrix with diagonal elements from S and off-diagonal elements from T.

`V`**:** Numeric gradient matrix (class `"matrix"`) of the nonlinear model function.

`A`**:** Scaled sparse model matrix (class "`dgCMatrix`") for the the unit, orthogonal random effects, U.

`Cm`**:** Reduced, weighted sparse model matrix (class "`dgCMatrix`") for the unit, orthogonal random effects, U. .

`Cx`**:** The "`x`" slot in the weighted sparse model matrix (class "`dgCMatrix`") for the unit, orthogonal random effects, U, in generalized linear mixed models. For these models the matrices A and C have the same sparsity pattern and only the "`x`" slot of C needs to be stored.

`L`**:** The sparse lower Cholesky factor of $P(AA' + I)P'$ (class "`dCHMfactor`") where P is the fill-reducing permutation calculated from the pattern of nonzeros in A.

`deviance`**:** Named numeric vector containing the deviance corresponding to the maximum likelihood (the "`ML`" element) and "`REML`" criteria and various components. The "`ldL2`" element is twice the logarithm of the determinant of the Cholesky factor in the `L` slot. The "`usqr`" component is the value of the random-effects quadratic form.

`fixef`**:** Numeric vector of fixed effects.

`ranef`**:** Numeric vector of random effects on the original scale.

`u`**:** Numeric vector of orthogonal, constant variance, random effects.

`eta`**:** The linear predictor at the current values of the parameters and the random effects.

`mu`**:** The means of the responses at the current parameter values.

`muEta`**:** The diagonal of the Jacobian of μ by η. Has length zero (0) except for generalized mixed models.

`var`**:** The diagonal of the conditional variance of Y given the random effects, up to prior weights. In generalized mixed models this is the value of the variance function for the `glm` family.

`resid`**:** The residuals, $y - \mu$, weighted by the `sqrtrWt` slot (when its length is > 0).

`sqrtXWt`**:** The square root of the weights applied to the model matrices X and Z. This may be of length zero (0), indicating unit weights.

`sqrtrWt`**:** The square root of the weights applied to the residuals to obtain the weighted residual sum of squares. This may be of length zero (0), indicating unit weights.

`RZX`**:** The dense solution (class "`matrix`") to $LRZX = ST'Z'X = AX$.

`RX`**:** The upper Cholesky factor (class "`matrix`") of the downdated $X'X$.

The "`summary.cpglmm`" class *contains* the "`cpglmm`" class and has the following additional slots:

`methTitle`**:** character string specifying a method title

`logLik`**:** the same as `logLik(object)`.

`ngrps`**:** the number of levels per grouping factor in the `flist` slot.

`sigma`**:** the scale factor for the variance-covariance estimates

`coefs`**:** the matrix of estimates, standard errors, etc. for the fixed-effects coefficients

`REmat`**:** the formatted Random-Effects matrix

`AICtab`**:** a named vector of values of AIC, BIC, log-likelihood and deviance

The "`bcplm`" class extends "`cplm`" directly, and has the following additional slots:

`dims`**:** a named integer vector of dimensions.

`sims.list`: an object of class `"mcmc.list"`. It is a list of `n.chains` `mcmc` objects, each `mcmc` object storing the simulation result from a Markov chain. See `mcmc` and `mcmc.convert`. Since this is an `"mcmc.list"` object, most methods defined in the `coda` package can be directly applied to it.

`Zt`: the transpose of model matrix for the random effects, stored as a compressed column-oriented sparse matrix (class `"dgCMatrix"`).

`flist`: the list of grouping factors for the random effects.

`prop.var`: a named list of proposal variance-covariance matrix used in the Metropolis-Hasting update.

The `"zcpglm"` class extends `"cplm"` directly and has the following additional slots:

`coefficients`: a list of estimated mean parameters for the zero-inflation and the compound Poisson parts, respectively

`residuals`: raw residuals, class `"numeric"`

`fitted.values`: the fitted mean values, that is (1 - probability of zero state) * compound Poisson mean, class `"numeric"`

`df.residual`: residual degrees of freedom, class `"integer"`

`offset`: a list of the offset vectors used in each model

`prior.weights`: the weights initially supplied, a vector of `1`s if none were, class `"numeric"`

`y`: the response vector used.

`control`: the value of the control argument used, class `"list"`

`p`: the maximum likelihood estimate of the index parameter.

`phi`: the maximum likelihood estimate of the dispersion parameter.

`vcov`: estimated variance-covariance matrix, class `"matrix"`

`converged`: indicating whether the algorithm has converged, class `"logical"`.

`na.action`: method of handling `NA`'s, class `"NullFunc"`.

`llik`: loglikelihood, class `numeric`.

The `"gini"` class has the following slots:

`call`: the matched call.

`gini`: a matrix of the Gini indices. The row names are corresponding to the base while the column names are corresponding to the scores.

`sd`: a matrix of standard errors for each computed Gini index.

`lorenz`: a list of matrices that determine the graph of the ordered Lorenz curve associated with each base and score combination. For each base, there is an associated matrix.

Extends

Class `"cpglm"` extends class `"cplm"`, directly.

Class `"cpglmm"` extends class `"cplm"`, directly;

Class `"summary.cpglmm"` extends class `"cpglmm"`, directly; class `"cplm"`, by class `"cpglmm"`, distance 2.

Class `"bcplm"` extends class `"cplm"`, directly.

Class `"zcpglm"` extends class `"cplm"`, directly.

Methods

The following methods are defined for the class `"cplm"`, which are also applicable to all of the derived classes:

$ `signature(x = "cplm")`: extract a slot of `x` with a specified slot name, just as in list.

[[`signature(x = "cplm", i = "numeric", j = "missing")`: extract the i-th slot of a `"cpglm"` object, just as in list.

[[`signature(x = "cplm", i = "character", j = "missing")`: extract the slots of a `"cpglm"` object with names in `i`, just as in list.

[`signature(x = "cplm", i = "numeric", j = "missing", drop="missing")`: extract the i-th slot of a `"cpglm"` object, just as in list. `i` could be a vector.

[`signature(x = "cplm", i = "character", j = "missing", drop="missing")`: extract the slots of a `"cpglm"` object with names in `i`, just as in list. `i` could be a vector.

names `signature(x = "cplm")`: return the slot names.

terms `signature(x = "cplm")`: extract the `terms` object from the model frame. See `terms`.

formula `signature(x = "cplm")`: extract the `formula` slot. See `formula`.

model.matrix `signature(object = "cplm")`: extract the design matrix.

show `signature(object = "cplm")`: method for show.

vcov `signature(object = "cplm")`: extract the variance-covariance matrix of a `"cplm"` object.

The following methods are defined for the `"cpglm"` class:

coef `signature(object = "cpglm")`: extract the estimated coefficients.

fitted `signature(object = "cpglm")`: return the fitted values.

residuals `signature(object = "cpglm")`: extract residuals from a `cpglm` object. You can also specify a `type` argument to indicate the type of residuals to be computed. See `glm.summaries`.

resid `signature(object = "cpglm")`: same as `residuals`.

AIC `signature(object = "cpglm",k="missing")`: extract the AIC information from the `"cpglm"` object. See `AIC`.

deviance `signature(object = "cpglm")`: extract the deviance from the `"cpglm"` object. See `deviance`.

summary `signature(object = "cpglm")`: the same as `glm.summaries` except that both the dispersion and the index parameter are estimated using maximum likelihood estimation.

predict `signature(object = "cpglm")`: generate predictions for new data sets

The following are written for `"cpglmm"`:

print `signature(x = "cpglmm")`: print the object

summary `signature(object = "cpglmm")`: summary results

predict `signature(object = "cpglmm")`: generate predictions for new data sets

VarCorr `signature(x = "cpglmm")`: estimation for the variance components

vcov `signature(object = "cpglmm")`: variance-covariance matrix for fixed effects

The following methods are available for the class `"bcplm"`:

plot `signature(x = "bcplm", y = "missing")`: summarize the `"bcplm"` object with a trace of the sampled output and a density estimate for each variable in the chain. See `plot.mcmc`.

summary `signature(object = "bcplm")`: produce two sets of summary statistics. See `summary.mcmc`.

VarCorr `signature(x = "bcplm")`: estimation for the variance components if the random effects are present

fixef `signature(object = "bcplm")`: extract fixed effects. Additional arguments include: `sd = FALSE`: extract standard errors; `quantiles = NULL`: compute empirical quantiles. These additional statistics are stored as attributes in the returned results.

The following methods are defined for the `"zcpglm"` class:

coef `signature(object = "zcpglm")`: extract the estimated coefficients.

fitted `signature(object = "zcpglm")`: return the fitted values.

residuals `signature(object = "zcpglm")`: extract raw residuals.

resid `signature(object = "zcpglm")`: same as `residuals`.

summary `signature(object = "zcpglm")`: summary statistics using Z-test.

predict `signature(object = "zcpglm")`: generate predicted values for a new data set. Argument `type = c("response", "zero", "tweedie")`specifies which component of the fitted values should be computed.

The following methods are defined for the `"gini"` class:

plot `signature(x = "gini", y = "missing")`: plot the ordered Lorenz curve from each model comparison. If `overlay = TRUE` (the default), different curves are plotted on the same graph for each base.

show `signature(object = "gini")`: print the computed Gini indices and standard errors.

Author(s)

Wayne Zhang `<actuary_zhang@hotmail.com>`

See Also

See also `cpglm`, `cpglmm`, `bcplm`, `zcpglm`, `glm`.

`cpglm` *Compound Poisson Generalized Linear Models*

Description

This function fits compound Poisson generalized linear models.

Usage

```
cpglm(formula, link = "log", data, weights, offset,
          subset, na.action = NULL, contrasts = NULL,
          control = list(), chunksize = 0,
          optimizer = "nlminb", ...)
```

Arguments

`formula`	an object of class `formula`. See also in `glm`.
`link`	a specification for the model link function. This can be either a literal character string or a numeric number. If it is a character string, it must be one of "log", "identity", "sqrt" or "inverse". If it is numeric, it is the same as the `link.power` argument in the `tweedie` function. The default is `link = "log"`.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model.
`weights`	an optional vector of weights. Should be either `NULL` or a numeric vector. When it is numeric, it must be positive. Zero weights are not allowed in `cpglm`.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of options, and is `na.fail` if that is unset. Another possible value is `NULL`, no action. Value `na.exclude` can be useful.
`offset`	this can be used to specify an a priori known component to be included in the linear predictor during fitting. This should be either `NULL` or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used.
`contrasts`	an optional list. See `contrasts.arg`.
`control`	a list of parameters for controling the fitting process. See 'Details' below.
`chunksize`	an integer that indicates the size of chunks for processing the data frame as used in `bigglm`. The value of this argument also determines how the model is estimated. When it is `0` (the default), regular Fisher's scoring algorithms are used, which may run into memory issues when handling large data sets. In contrast, a value greater than `0` indicates that the `bigglm` is employed to fit the GLMs. The function `bigglm` relies on the bounded memory regression technique, and thus is well suited to large data GLMs.
`optimizer`	a character string that determines which optimization routine is to be used in estimating the index and the dispersion parameters. Possible choices are `"nlminb"` (the default, see `nlminb`), `"bobyqa"` (`bobyqa`) and `"L-BFGS-B"` (`optim`).
`...`	additional arguments to be passed to `bigglm`. Not used when `chunksize = 0`. The `maxit` argument defaults to `50` in `cpglm` if not specified.

Details

This function implements the profile likelihood approach in Tweedie compound Poisson generalized linear models. First, the index and the dispersion parameters are estimated by maximizing (numerically) the profile likelihood (profile out the mean parameters as they are determined for a given value of the index parameter). Then the mean parameters are estimated using a GLM with the above-estimated index parameter. To compute the profile likelihood, one must resort to numerical methods provided in the `tweedie` package for approximating the density of the compound Poisson distribution. Indeed, the function `tweedie.profile` in that package makes available the profile likelihood approach. The `cpglm` function differs from `tweedie.profile` in two aspects. First, the user does not need to specify the grid of possible values the index parameter can take. Rather, the optimization of the profile likelihood is automated. Second, big data sets can be handled where the `bigglm` function from the `biglm` package is used in fitting GLMs. The `bigglm` is invoked when the argument `chunksize` is greater than 0. It is also to be noted that only MLE estimate for the dispersion parameter is included here, while `tweedie.profile` provides several other possibilities.

The package used to implement a second approach using the Monte Carlo EM algorithm, but it is now removed because it does not offer obvious advantages over the profile likelihood approach for this model.

The `control` argument is a list that can supply various controlling elements used in the optimization process, and it has the following components:

`bound.p` a vector of lower and upper bounds for the index parameter p used in the optimization. The default is `c(1.01, 1.99)`.

`trace` if greater than 0, tracing information on the progress of the fitting is produced. For `optimizer = "nlminb"` or `optimizer = "L-BFGS-B"`, this is the same as the `trace` control parameter, and for `optimizer = "bobyqa"`, this is the same as the `iprint` control parameter. See the corresponding documentation for details.

`max.iter` maximum number of iterations allowed in the optimization. The default is `300`.

`max.fun` maximum number of function evaluations allowed in the optimizer. The default is `2000`.

Value

`cpglm` returns an object of class `"cpglm"`. See `cpglm-class` for details of the return values as well as various methods available for this class.

Author(s)

Yanwei (Wayne) Zhang `<actuary_zhang@hotmail.com>`

References

Dunn, P.K. and Smyth, G.K. (2005). Series evaluation of Tweedie exponential dispersion models densities. Statistics and Computing, *15, 267-280.*

See Also

The users are recommended to see the documentation for `cpglm-class`, `glm`, `tweedie`, and `tweedie.profile` for related information.

Examples

```
fit1 <- cpglm(RLD ~ factor(Zone) * factor(Stock),
  data = FineRoot)

# residual and qq plot
parold <- par(mfrow = c(2, 2), mar = c(5, 5, 2, 1))
# 1. regular plot
r1 <- resid(fit1) / sqrt(fit1$phi)
```

```
plot(r1 ~ fitted(fit1), cex = 0.5)
qqnorm(r1, cex = 0.5)
# 2. quantile residual plot to avoid overlapping
u <- tweedie::ptweedie(fit1$y, fit1$p, fitted(fit1), fit1$phi)
u[fit1$y == 0] <- runif(sum(fit1$y == 0), 0, u[fit1$y == 0])
r2 <- qnorm(u)
plot(r2 ~ fitted(fit1), cex = 0.5)
qqnorm(r2, cex = 0.5)
par(parold)

# use bigglm
fit2 <- cpglm(RLD ~ factor(Zone),
  data = FineRoot, chunksize = 250)
```

`cpglmm` *Compound Poisson Generalized Linear Mixed Models*

Description

Laplace approximation and adaptive Gauss-Hermite quadrature methods for compound Poisson mixed and additive models.

Usage

```
cpglmm(formula, link = "log", data, weights, offset, subset,
    na.action, inits = NULL,  contrasts = NULL,
    control = list(), basisGenerators = c("tp", "bsp", "sp2d"),
    optimizer = "nlminb", doFit = TRUE, nAGQ = 1)
```

Arguments

`formula`	a two-sided linear formula object describing the model structure, with the response on the left of a ~ operator and the terms, separated by + operators, on the right. The vertical bar character "|" separates an expression for a model matrix and a grouping factor. The right side can also include basis generators. See `lme4` and `basisGenerators` below.
`link`	a specification for the model link function. This can be either a literal character string or a numeric number. If it is a character string, it must be one of "log", "identity", "sqrt" or "inverse". If it is numeric, it is the same as the `link.power` argument in the `tweedie` function. The default is `link="log"`.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model.
`subset, weights, na.action, offset, contrasts`	further model specification arguments as in `cpglm`; see there for details.

`inits`	a named list with three components 'beta', 'phi', 'p', 'Sigma' that supply the initial values used in the optimization. If not supplied, the function will generate initial values automatically, which are based on a GLM with the supplied model structure.
`control`	a list of parameters for controlling the fitting process. See `cpglm`. The parameter `PQL.init` is not used.
`basisGenerators`	a character vector of names of functions that generate spline bases. This is used when smoothing effects are to be included in the model. See `tp` for details.
`optimizer`	a character string that determines which optimization routine is to be used. Possible choices are `"nlminb"` (the default, see `nlminb`), `"bobyqa"` (`bobyqa`) and `"L-BFGS-B"` (`optim`).
`doFit`	if `FALSE`, the constructed `"cpglmm"` object is returned before the model is fitted.
`nAGQ`	a positive integer - the number of points per axis for evaluating the adaptive Gauss-Hermite approximation to the log-likelihood. This defaults to 1, corresponding to the Laplacian approximation. Values greater than 1 produce greater accuracy in the evaluation of the log-likelihood at the expense of speed.

Details

Estimation of compound Poisson mixed models in existing software has been limited to the Penalized Quasi-Likelihood [PQL] approach (e.g., see `glmmPQL`). While straightforward and fast, this method is not equipped to estimate the unknown variance function, i.e., the index parameter. In contrast, the function `cpglmm` implements true likelihood-based inferential procedures, i.e., the Laplace approximation and the Adaptive Gauss-Hermite Quadrature (for single grouping factor), so that all parameters in the model can be estimated using maximum likelihood estimation.

This implementation is based on the older `lme4` package (the `0.9*` version), with changes made on updating of the mean, the variance function and the marginal loglikelihood. For the Laplace method, the contribution of the dispersion parameter to the approximated loglikelihood is explicitly accounted for, which should be more accurate and more consistent with the quadrature estimate. Indeed, both the dispersion parameter and the index parameter are included as a part of the optimization process. In computing the marginal loglikelihood, the density of the compound Poisson distribution is approximated using numerical methods provided in the `tweedie` package. For details of the Laplace approximation and the Gauss-Hermite quadrature method for generalized linear mixed models, see the documentation associated with `lme4`.

In addition, similar to the package `amer` (already retired from CRAN), we provide convenient interfaces for fitting additive models using penalized splines. See the 'example' section for one such application.

Value

`cpglmm` returns an object of class `cpglmm`. See `cpglmm-class` for details of the return values as well as various method available for this class.

Author(s)

Yanwei (Wayne)) Zhang `<actuary_zhang@hotmail.com>`

References

Zhang Y (2013). Likelihood-based and Bayesian Methods for Tweedie Compound Poisson Linear Mixed Models, Statistics and Computing, *23, 743-757.* https://github.com/actuaryzhang/cplm/files/144051/TweediePaper.pdf

Bates D, Maechler M, Bolker B and Walker S (2015). `lme4`*: Linear mixed-effects models using Eigen and S4..*

See Also

The users are recommended to see `cpglm` for a general introduction to the compound Poisson distribution, `lme4` for syntax and usage of mixed-effect models and `cpglmm-class` for detailed explanation of the return value.

Examples

```
## Not run:
# use Stock and Spacing as main effects and Plant as random effect
(f1 <- cpglmm(RLD ~ Stock + Spacing +  (1|Plant), data = FineRoot))

coef(f1); fixef(f1); ranef(f1)  #coefficients
VarCorr(f1)  #variance components

# add another random effect
(f2 <- update(f1, . ~ . + (1|Zone)))
# test the additional random effect
anova(f1,f2)

# try a different optimizer
(f3 <- cpglmm(RLD ~  Stock + Spacing +  (1|Plant),
            data = FineRoot, optimizer = "bobyqa",
            control = list(trace = 2)))

# adaptive G-H quadrature
(f4 <- cpglmm(RLD ~  Stock + Spacing +  (1|Plant),
            data = FineRoot, nAGQ = 3))

# a model with smoothing effects
(f5 <- cpglmm(increLoss ~ tp(lag, k = 4) + (1|year) ,
            data = ClaimTriangle))

## End(Not run)
```

`datasets` *Data sets in the cplm pakcage*

Description

The data sets included in package is described here.

Usage

```
data(FineRoot)
data(ClaimTriangle)
data(AutoClaim)
```

Format

`FineRoot`: a data set used for the study of the fine root length density of plants. It is a data frame with 511 records and 5 variables:

`Plant`: identifier of the apple tree, 1-8

`Stock`: root stokcing, one of three different root stocks: Mark, MM106 and M26

`Spacing`: between-row $\times$ within-row spacings, one of the following two: 4×2 meters and 5×3 meters

`Zone`: inner or outer

`RLD`: root length density

`ClaimTriangle`: a data set from an insurance loss reserving triangle. It is a data frame with 55 records and 3 variables:

`year`: the year when the accident occurs

`lag`: development lag

`increLoss`: incremental insurance loss in 1000s

`AutoClaim`: a motor insurance data set retrieved from the SAS Enterprise Miner database. It is a data frame with 10296 records and 29 variables:

`POLICYNO`: "character", the policy number

`PLCYDATE`: "Date", policy effective date

`CLM_FREQ5`: "integer", the number of claims in the past 5 years

`CLM_AMT5`: "integer", the total claim amount in the past 5 years

`CLM_AMT`: "integer", the claim amount in the current insured period

`KIDSDRIV`: "integer", the number of driving children

`TRAVTIME`: "integer", the distance to work

`CAR_USE`: "factor", the primary use of the vehicle: "Commercial", "Private".

`BLUEBOOK`: "integer", the value of the vehicle

`RETAINED`: "integer", the number of years as a customer

`NPOLICY`: "integer", the number of policies

`CAR_TYPE`: "factor", the type of the car: "Panel Truck", "Pickup", "Sedan", "Sports Car", "SUV", "Van".

`RED_CAR`: "factor", whether the color of the car is red: "no", "yes".

`REVOLKED`: "factor", whether the dirver's license was invoked in the past 7 years: "No", "Yes",

`MVR_PTS`: "integer", MVR violation records

`CLM_FLAG`: "factor", whether a claim is reported: "No", "Yes".

`AGE`: "integer", the age of the driver

`HOMEKIDS`: "integer", the number of children

`YOJ`: "integer", years at current job

`INCOME`: "integer", annual income

`GENDER`: "factor", the gender of the driver: "F", "M".

`MARRIED`: "factor", married or not: "No", "Yes".

`PARENT1`: "factor", single parent: "No", "Yes".

`JOBCLASS`: "factor": "Unknown", "Blue Collar", "Clerical", "Doctor", "Home Maker", "Lawyer", "Manager", "Professional", "Student".

`MAX_EDUC`: "factor", max education level:"<High School", "Bachelors", "High School", "Masters", "PhD".

`HOME_VAL`: "integer", the value of the insured's home

`SAMEHOME`: "integer", years in the current address

`DENSITY`: "factor", home/work area: "Highly Rural", "Highly Urban", "Rural", "Urban".

`IN_YY`: "logical", whether the record is used in the Yip and Yau (2005) paper.

Source

de Silva, H. N., Hall, A. J., Tustin, D. S. and Gandar, P. W. (1999). Analysis of distribution of root length density of apple trees on different dwarfing rootstocks. Annals of Botany, *83: 335-345.*

Dunn, P.K. and Smyth, G.K. (2005). Series evaluation of Tweedie exponential dispersionmodels densities. Statistics and Computing, *15, 267-280.*

Peters G. W., Shevchenko P. V. and Wuthrich M. V. (2009). Model Uncertainty in Claims Reserving within Tweedie's Compound Poisson Models. Astin Bulletin, *39(1), 1-33.*

Yip, K. C. H. and Yau, K. K. W. (2005). On Modeling Claim Frequency Data In General Insurance With Extra Zeros. Insurance: Mathematics and Economics, *36(2), 153-163.*

`getF` *Get and plot the smoothing function values*

Description

Get and plot the estimated smoothing function values

Usage

```
getF(object, which, n=100, newdata, interval=c("NONE", "MCMC",
    "RW"), addConst=TRUE, varying=1, level=0.9, sims=1000)

plotF(object, which, n=100, interval="RW", addConst=TRUE,
    trans=I, level=0.9, sims=1000, auto.layout=TRUE, rug=TRUE,
    legendPos="topright", ...)
```

Arguments

`object`	a fitted `cpglmm` object.
`which`	(optional) an integer vector or a character vector of names giving the smooths for which fitted values are desired. Defaults to all.
`n`	if no `newdata` is given, fitted values for a regular grid with `n` values in the range of the respective covariates are returned
`newdata`	An optional data frame in which to look for variables with which to predict
`interval`	what mehod should be used to compute pointwise confidence/HPD intervals: RW= bias-adjusted empirical bayes
`addConst`	boolean should the global intercept and intercepts for the levels of the by-variable be included in the fitted values (and their CIs) can also be a vector of the same length as `which`
`varying`	value of thevarying-covariate (see `tp`) to be used if no newdata is supplied. Defaults to 1.
`level`	level for the confidence/HPD intervals
`sims`	how many iterates should be generated for the MCMC-based HPD-intervals
`trans`	a function that should be applied to the fitted values and ci's before plotting (e.g. the inverse link function to get plots on the scale of the reponse)
`auto.layout`	automagically set plot layout via `par()$mfrow`
`rug`	add `rug`-plots of the observed covariate locations
`legendPos`	a (vector of) keyword(s) where to put labels of by-variables (see `legend`). "none" if you don't want a legend.
`...`	arguments passed on to the low-level plot functions (`plot`, `matlines`), `legend`, and `title`

Value

a list with one `data.frame` for each function, giving `newdata` or the values of the generated grid plus the fitted values (and confidence/HPD intervals).

Note

These are from the `amer` package that has retired from CRAN. The formula used for the pointwise bias-adjusted CIs is taken from Ruppert and Wand's 'Semiparametric Regression' (2003), p. 140. These leave out the uncertainty associated with the variance component estimates.

Author(s)

Fabian Scheipl `<fabian.scheipl@googlemail.com>`

See Also

See the vignette for examples

`gini` *The Gini index*

Description

Compute Gini indices and their standard errors.

Usage

```
gini(loss, score, base = NULL, data, ...)
```

Arguments

`loss`	a character that contains the name of the response variable.
`score`	a character vector that contains the list of the scores, which are the predictions from the set of models to be compared.
`base`	a character that contains the name of a baseline statistic. If `NULL` (the default), each score will be successively used as the base.
`data`	a data frame containing the variables listed in the above arguments.
`...`	not used.

Details

For model comparison involving the compound Poisson distribution, the usual mean squared loss function is not quite informative for capturing the differences between predictions and observations, due to the high proportions of zeros and the skewed heavy-tailed distribution of the positive losses. For this reason, Frees et al. (2011) develop an ordered version of the Lorenz curve and the associated Gini index as a statistical measure of the association between distributions, through which different predictive models can be compared. The idea is that a score (model) with a greater Gini index produces a greater separation among the observations. In the insurance context, a higher Gini index indicates greater ability to distinguish good risks from bad risks. Therefore, the model with the highest Gini index is preferred.

This function computes the Gini indices and their asymptotic standard errors based on the ordered Lorenz curve. These metrics are mainly used for model comparison. Depending on the problem, there are generally two ways to do this. Take insurance predictive modeling as an example. First, when there is a baseline premium, we can compute the Gini index for each score (predictions from the model), and select the model with the highest Gini index. Second, when there is no baseline premium (`base = NULL`), we successively specify the prediction from each model as the baseline premium and use the remaining models as the scores. This results in a matrix of Gini indices, and we select the model that is least vulnerable to alternative models using a "mini-max" argument - we select the score that provides the smallest of the maximal Gini indices, taken over competing scores.

Value

`gini` returns an object of class `"gini"`. See `gini-class` for details of the return values as well as various methods available for this class.

Author(s)

Yanwei (Wayne) Zhang <actuary_zhang@hotmail.com>

References

Frees, E. W., Meyers, G. and Cummings, D. A. (2011). Summarizing Insurance Scores Using a Gini Index. Journal of the American Statistical Association, *495, 1085 - 1098.*

See Also

The users are recommended to see the documentation for `gini-class` for related information.

Examples

```
## Not run:

# Let's fit a series of models and compare them using the Gini index
da <- subset(AutoClaim, IN_YY == 1)
da <- transform(da, CLM_AMT = CLM_AMT / 1000)

P1 <- cpglm(CLM_AMT ~ 1, data = da, offset = log(NPOLICY))


P2 <- cpglm(CLM_AMT ~ factor(CAR_USE) + factor(REVOLKED) +
              factor(GENDER) + factor(AREA) +
              factor(MARRIED) + factor(CAR_TYPE),
            data = da, offset = log(NPOLICY))

P3 <- cpglm(CLM_AMT ~ factor(CAR_USE) + factor(REVOLKED) +
              factor(GENDER) + factor(AREA) +
              factor(MARRIED) + factor(CAR_TYPE) +
              TRAVTIME + MVR_PTS + INCOME,
            data = da, offset = log(NPOLICY))

da <- transform(da, P1 = fitted(P1), P2 = fitted(P2), P3 = fitted(P3))

# compute the Gini indices
gg <- gini(loss = "CLM_AMT", score  = paste("P", 1:3, sep = ""),
           data = da)
gg

# plot the Lorenz curves
theme_set(theme_bw())
plot(gg)
plot(gg, overlay = FALSE)


## End(Not run)
```

sp2d *2-dimentional Radial Spline*

Description

2-dimentional radial spline generator used in `cpglmm`

Usage

```
sp2d(x1, x2, k = max(20, min(length(x1)/4, 150)), by = NULL,
allPen = FALSE, varying = NULL, diag = FALSE,
knots1 = quantile(x1, probs = 1:k/(k + 1)),
knots2 = quantile(x1, probs = 1:k/(k + 1)))
```

Arguments

`x1`	the first covariate in the coordinate
`x2`	the second covariate in the coordinate
`k`	number of knots
`by`	not used. This is just for compatibility with `amer`.
`allPen`	not used. This is just for compatibility with `amer`.
`varying`	not used. This is just for compatibility with `amer`.
`diag`	not used. This is just for compatibility with `amer`.
`knots1`	vector of knot locations for the first covariate in the coordinate
`knots2`	vector of knot locations for the second covariate in the coordinate

Author(s)

Fabian Scheipl <`fabian.scheipl@googlemail.com`>

tp *Generate basis functions for penalized spline smoothing.*

Description

`tp` generates a truncated power basis and `bsp` generates a reparameterized b-spline basis for penalized spline smoothing.

Usage

```
tp(x, degree=1, k=15, by=NULL, allPen=FALSE, varying=NULL, diag=FALSE,
    knots=quantile(x, probs = (1:(k - degree))/(k - degree  + 1)),
    centerscale=NULL, scaledknots=FALSE)

bsp(x, k=15, spline.degree=3, diff.ord=2, knots, by,
    allPen=FALSE, varying, diag=FALSE)
```

Arguments

`x`	covariate for the smooth function
`degree`	integer: degree of truncated polynomials (0: piecewise constant, 1: piecewise linear etc..)
`k`	integer: dimensionality of the basis (i.e.: number of knots + degree)
`by`	factor variable: estimate separate functions for each level - this assumes standard treatment contrasts for the supplied factor.
`allPen`	boolean: if TRUE, make design for group-specific curves with common smoothing parameter: all parameters (including the normally unpenalized basis functions in X) are penalized, every level of "by" has the same amount of smoothing if FALSE, make design for separate curves for each by-level: separate smoothing parameters for every level of "by", unpenalized estimates for the coefficients associated with X
`varying`	numeric: if not NULL, a varying coefficient model is fit: f(x,varying) = f(x)*varying
`diag`	logical: force a diagonal covariance-matrix for the random effects for X if `allPen=TRUE`?
`knots`	vector of knot locations (optional). Defaults to quantile-based knots at the $i/(k+1-$degree)-quantiles for $i = 1, \dots, k-$degree.
`centerscale`	numeric(2): center&scale x by these values if not NULL
`scaledknots`	boolean: are knot locations given for the rescaled x-values?
`spline.degree`	integer: degree of B-splines (defaults to cubic)
`diff.ord`	integer: order of the difference penalty on the un-reparamerized spline coefficients. Defaults to 2, that is, penalized deviations from linearity.

Details

`tp` generates truncated power bases which have `degree` unpenalized basis functions, namely $x^1, \dots, x^{degree}$ and $k-$`degree` penalized basis functions that contain the positive part $(x-\kappa_j)^{degree}$ for knots $\kappa_j, j = 1, dots, k-$`degree`. This function can be used as a reference when implementing other `basisGenerators` that can be used for additive models through `cpglmm`.

`bsp` generate a b-spline basis with equidistant knots in mixed model reparameterization.

Value

list with entries: `"X"`: For `tp`, it is an `n x degree` design matrix for unpenalized part (without intercept) (or a list of those for every level of by if allPen=F); and for `bsp`, it is an `n x (diff.ord - 1)` design matrix for unpenalized part (without intercept).

`"Z"`: For `tp`, it is an `n x (k-degree)` design matrix for penalized part (or a list of those for every level of by if allPen=F); and for `bsp`, it is an `n x (k - diff.ord+1)` design matrix for penalized part.

Note

These are from the `amer` package that has retired from CRAN.

Author(s)

Fabian Scheipl <`fabian.scheipl@googlemail.com`>

`zcpglm` *Zero-inflated Compound Poisson Generalized Linear Models*

Description

This function fits zero-inflated compound Poisson generalized linear models.

Usage

```
zcpglm(formula, link = "log", data, weights, offset,
      subset, na.action = NULL, contrasts = NULL,
      control = list(), optimizer = "nlminb")
```

Arguments

`formula`	an object of class `formula`. See details below.
`link`	a specification for the compound Poisson model link function. See `cpglm`. For the zero-inflation model, the logit link is used.
`data, subset, na.action, contrasts, control`	see `cpglm` for details.
`weights, offset`	prior weights and offset. If specified, they will be used in both the zero-inflation and the compound Poisson model. See details below for specifying different offsets for each model.
`optimizer`	a character string that determines which optimization routine is to be used. Possible choices are `"nlminb"` (the default, see `nlminb`), `"bobyqa"` (`bobyqa`) and `"L-BFGS-B"` (`optim`).

Details

This function implements zero-inflated compound Poisson generalized linear models. This is similar to the zero-inflated Poisson model for count data, with the Poisson distribution replaced by the compound Poisson distribution. Specifically, the observation is allowed to com from a degenerate distribution at zero with a positive probability, in addition to the regular compound Poisson process. This latent zero-inflation part is specified using a logistic regression structure, while the compound Poisson component is modeled the same as in `cpglm`. Parameters are estimated by maximizing the marginal likelihood, and the variance-covariance matrix is computed numerically.

The formula specification is similar to that in `zeroinfl` (package `pscl`) except that we use `||` instead of | to separate the two parts of the model. For example, the formula `y ~ x1 + x2 || 1` indicates that `y ~ x1 + x2` is used for the compound Poison part and `~ 1` for the zero-inflation part. If the inflation part is omitted, it defaults to a model with an intercept. Offsets can be specified in both components of the model, e.g., `y ~ x1 + offset(x2) | z1 + z2 + offset(z3)`.

More details of this model are available in the package vignettes.

Value

`zcpglm` returns an object of class `"zcpglm"`. See `zcpglm-class` for details of the return values as well as various methods available for this class.

Author(s)

Yanwei (Wayne) Zhang `<actuary_zhang@hotmail.com>`

See Also

The users are recommended to see the documentation for `zcpglm-class`, `cpglm` and `zeroinfl` for related information.

Examples

```
da <- subset(AutoClaim, IN_YY == 1) # use data in the Yip and Yau paper
da <- transform(da, CLM_AMT5 = CLM_AMT5/1000,
                INCOME = INCOME/10000)
(Z0 <- zcpglm(CLM_AMT5 ~ CAR_USE + MARRIED + AREA + MVR_PTS||
             MVR_PTS + INCOME, data = da))
```

Index