Package: countreg (via r-universe)

September 4, 2024

Version 0.3-0

Date 2024-05-08

Title Count Data Regression

Description Regression models for count data, including negative binomial, zero-inflated, zero-truncated, and hurdle models. Drivers for combination with flexmix and mboost are also provided. Previously available functions for graphical goodness-of-fit assessment (rootograms etc.) are now provided in the 'topmodels' package on R-Forge.

LazyLoad yes

Depends R (>= 3.6.0), MASS

Imports graphics, grDevices, stats, utils, distributions3 (>= 0.2.1), Formula

Suggests AER, brglm2, car, flexmix (>= 2.3-11), gamlss, gamlss.dist, ggplot2, lmtest, mboost, methods, modelsummary, nonnest2, sandwich, tinytest, topmodels, knitr, rmarkdown

Additional_repositories `https://R-Forge.R-project.org`

VignetteBuilder knitr

License GPL-2 | GPL-3

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

RemoteUrl https://github.com/r-forge/countreg

RemoteRef HEAD

RemoteSha bd4b23fcfd716740cca26b847eeea7bf15cd63f2

Contents

`Binomial-extensions` *Extension of the Binomial Distribution*

Description

Score function, hessian, mean, and variance for the binomial distribution with parameters `prob` and `size`.

Usage

```
sbinom(x, prob, size, parameter = "prob", drop = TRUE)
hbinom(x, prob, size, parameter = "prob", drop = TRUE)
mean_binom(prob, size, drop = TRUE)
var_binom(prob, size, drop = TRUE)
```

Arguments

`x`	vector of quantiles.
`prob`	probability of success on each trial.
`size`	number of trials (zero or more).
`parameter`	character. Derivatives are computed wrt this paramter. Note: Only `"prob"` is implemented.
`drop`	logical. Should the result be a matrix (`drop = FALSE`) or should the dimension be dropped (`drop = TRUE`, the default)?

Details

The binomial distribution with `size` $= n$ and `prob` $= p$ has density

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

for $x = 0, \ldots, n$.

The score function is

$$s(p) = \frac{x}{p} - \frac{n-x}{1-p}$$

The hessian is

$$h(p) = -\frac{x}{p^2} - \frac{n-x}{(1-p)^2}$$

Value

`sbinom` gives the score function, i.e., the 1st derivative of the log-density wrt prob and `hbinom` gives the hessian, i.e., the 2nd derivative of the log-density wrt prob. `mean` and `var` give the mean and variance, respectively.

See Also

Binomial encompassing `dbinom`, `pbinom`, `qbinom` and `rbinom`.

Examples

```
## Simulate some data
set.seed(123)
y <- rbinom(50, size = 1, prob = 0.3)

## Plot log-likelihood function
par(mfrow = c(1,3))
ll <- function(x) {sum(dbinom(y, size = 1, prob = x, log = TRUE))}
curve(sapply(x, ll), xlab = expression(pi), ylab = "", main = "Log-likelihood")
abline(v = 0.3, lty = 3)

## Plot score function
curve(sapply(x, function(x) sum(sbinom(y, size = 1, x))),
      xlab = expression(pi), ylab = "", main = "Score")
```

```
abline(h = 0, lty = 3)
abline(v = 0.3, lty = 3)

## Plot hessian
curve(sapply(x, function(x) sum(hbinom(y, size = 1, x))),
      xlab = expression(pi), ylab = "", main = "Hessian")
abline(v = 0.3, lty = 3)
```

`CodParasites` *Parasite Infections in Cod*

Description

Data on parasite infection in cod along the coast of Finmark.

Usage

```
data("CodParasites")
```

Format

A data frame containing 1254 observations on 10 variables.

intensity Number of parasites.

prevalence Factor indicating presence of parasites (i.e., `intensity > 0`).

area Factor indicating sampling area.

year Factor indicating sampling year.

depth Depth at which the fish were caught.

weight Weight of the fish.

length Length of the fish.

sex Factor indicating sex of the fish.

stage Factor indicating stage of the fish.

age Age of the fish.

Details

The red king crab *Paralithodes camtschaticus* was deliberately introduced to the Barents Sea in the 1960s and 1970s from its native area in the North Pacific. The carapace of these crabs is used by the leech *Johanssonia arctica* to deposit its eggs. The leech in turn is a vector for the blood parasite *Trypanosoma murmanensis* that can infect marine fish, including cod.

Hemmingsen et al. (2005) examined cod for trypanosome infections during annual cruises along the coast of Finnmark in North Norway over three successive years and in four different areas (A1 Sørøya; A2 Magerøya; A3 Tanafjord; A4 Varangerfjord). They show that trypanosome infections are strongest in the area Varangerfjord where the density of of red king crabs is highest. Thus, there is evidence that the introduction of the foreign red king crabs had an indirect detrimental effect on the health of the native cod population. This situation stands out because it is not an introduced parasite that is dangerous for a native host, but rather an introduced host that promotes transmission of two endemic parasites.

Zuur et al. (2009) reanalyze the data using binary and count data regression models in Chapters 10.2.2, 11.3.2, 11.4.2, 11.5.2.

Source

The data are taken from the online supplements of Zuur et al. (2009). `http://highstat.com/index.php/mixed-effects-models-and-extensions-in-ecology-with-r`

References

Hemmingsen W, Jansen PA, MacKenzie K (2005). "Crabs, Leeches and Trypanosomes: An Unholy Trinity?", *Marine Pollution Bulletin* **50**(3), 336–339.

Zuur AF, Ieno EN, Walker NJ, Saveliev AA, Smith GM (2009). *Mixed Effects Models and Extensions in Ecology with R*, Springer-Verlag, New York.

Examples

```
## load data
data("CodParasites", package = "countreg")

## Table 1 from Hemmingsen et al. (2005)
## number of observations
xtabs(~ area + year, data = CodParasites)
## prevalence of parasites (NAs counted as "yes")
tab <- xtabs(~ area + year + factor(is.na(prevalence) | prevalence == "yes"),
  data = CodParasites)
round(100 * prop.table(tab, 1:2)[,,2], digits = 1)

## omit NAs in response
CodParasites <- subset(CodParasites, !is.na(intensity))

## exploratory displays for hurdle and counts
par(mfrow = c(2, 2))
plot(factor(intensity == 0) ~ interaction(year, area), data = CodParasites)
plot(factor(intensity == 0) ~ length, data = CodParasites, breaks = c(15, 3:8 * 10, 105))
plot(jitter(intensity) ~ interaction(year, area), data = CodParasites,
  subset = intensity > 0, log = "y")
plot(jitter(intensity) ~ length, data = CodParasites, subset = intensity > 0, log = "y")

## count data models
cp_p   <-    glm(intensity ~ length + area * year, data = CodParasites, family = poisson)
cp_nb  <- glm.nb(intensity ~ length + area * year, data = CodParasites)
cp_hp  <- hurdle(intensity ~ length + area * year, data = CodParasites, dist = "poisson")
cp_hnb <- hurdle(intensity ~ length + area * year, data = CodParasites, dist = "negbin")
AIC(cp_p, cp_nb, cp_hp, cp_hnb)
BIC(cp_p, cp_nb, cp_hp, cp_hnb)

## rootograms
```

```
if(require("topmodels")) {
par(mfrow = c(2, 2))
rootogram(cp_p, max = 50, main = "Poisson")
rootogram(cp_nb, max = 50, main = "Negative Binomial")
rootogram(cp_hp, max = 50, main = "Hurdle Poisson")
rootogram(cp_hnb, max = 50, main = "Hurdle Negative Binomial")
}
```

`CrabSatellites` *Horseshoe Crab Mating*

Description

Determinants for male satellites to nesting horseshoe crabs.

Usage

```
data("CrabSatellites")
```

Format

A data frame containing 173 observations on 5 variables.

color Ordered factor indicating color (light medium, medium, dark medium, dark).

spine Ordered factor indicating spine condition (both good, one worn or broken, both worn or broken).

width Carapace width (cm).

weight Weight (kg).

satellites Number of satellites.

Details

Brockmann (1996) investigates horshoe crab mating. The crabs arrive on the beach in pairs to spawn. Furthermore, unattached males also come to the beach, crowd around the nesting couples and compete with attached males for fertilizations. These so-called satellite males form large groups around some couples while ignoring others. Brockmann (1996) shows that the groupings are not driven by environmental factors but by properties of the nesting female crabs. Larger females that are in better condition attract more satellites.

Agresti (2002, 2013) reanalyzes the number of satellites using count models. Explanatory variables are the female crab's color, spine condition, weight, and carapace width. Color and spine condition are ordered factors but are treated as numeric in some analyses.

Source

Table 4.3 in Agresti (2002).

References

Agresti A (2002). *Categorical Data Analysis*, 2nd ed., John Wiley & Sons, Hoboken.

Agresti A (2013). *Categorical Data Analysis*, 3rd ed., John Wiley & Sons, Hoboken.

Brockmann HJ (1996). "Satellite Male Groups in Horseshoe Crabs, *Limulus polyphemus*", *Ethology*, **102**(1), 1–21.

Examples

```
## load data, use ordered factors as numeric, and
## grouped factor version of width
data("CrabSatellites", package = "countreg")
CrabSatellites <- transform(CrabSatellites,
  color = as.numeric(color),
  spine = as.numeric(spine),
  cwidth = cut(width, c(-Inf, seq(23.25, 29.25), Inf))
)

## Agresti, Table 4.4
aggregate(CrabSatellites$satellites, list(CrabSatellites$cwidth), function(x)
  round(c(Number = length(x), Sum = sum(x), Mean = mean(x), Var = var(x)), digits = 2))

## Agresti, Figure 4.4
plot(tapply(satellites, cwidth, mean) ~ tapply(width, cwidth, mean),
  data = CrabSatellites, ylim = c(0, 6), pch = 19)

## alternatively: exploratory displays for hurdle (= 0 vs. > 0) and counts (> 0)
par(mfrow = c(2, 2))
plot(factor(satellites == 0) ~ width, data = CrabSatellites, breaks = seq(20, 33.5, by = 1.5))
plot(factor(satellites == 0) ~ color, data = CrabSatellites, breaks = 1:5 - 0.5)
plot(jitter(satellites) ~ width, data = CrabSatellites, subset = satellites > 0, log = "y")
plot(jitter(satellites) ~ factor(color), data = CrabSatellites, subset = satellites > 0, log = "y")

## count data models
cs_p    <-    glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
cs_nb   <- glm.nb(satellites ~ width + color, data = CrabSatellites)
cs_hp   <- hurdle(satellites ~ width + color, data = CrabSatellites, dist = "poisson")
cs_hnb  <- hurdle(satellites ~ width + color, data = CrabSatellites, dist = "negbin")
cs_hnb2 <- hurdle(satellites ~ 1 | width + color, data = CrabSatellites, dist = "negbin")
AIC(cs_p, cs_nb, cs_hp, cs_hnb, cs_hnb2)
BIC(cs_p, cs_nb, cs_hp, cs_hnb, cs_hnb2)

## rootograms
if(require("topmodels")) {
par(mfrow = c(2, 2))
r_p   <- rootogram(cs_p,   xlim = c(0, 15), main = "Poisson")
r_nb  <- rootogram(cs_nb,  xlim = c(0, 15), main = "Negative Binomial")
r_hp  <- rootogram(cs_hp,  xlim = c(0, 15), main = "Hurdle Poisson")
r_hnb <- rootogram(cs_hnb, xlim = c(0, 15), main = "Hurdle Negative Binomial")
}

## fitted curves
```

```
par(mfrow = c(1, 1))
plot(jitter(satellites) ~ width, data = CrabSatellites)
nd <- data.frame(width = 20:34, color = 2)
pred <- function(m) predict(m, newdata = nd, type = "response")
cs_ag <- glm(satellites ~ width, data = CrabSatellites, family = poisson(link = "identity"),
  start = coef(lm(satellites ~ width, data = CrabSatellites)))
lines(pred(cs_ag)   ~ width, data = nd, col = 2, lwd = 1.5)
lines(pred(cs_p)    ~ width, data = nd, col = 3, lwd = 1.5)
lines(pred(cs_hnb)  ~ width, data = nd, col = 4, lwd = 1.5)
lines(pred(cs_hnb2) ~ width, data = nd, col = 4, lwd = 1.5, lty = 2)
legend("topleft", c("Hurdle NB", "Hurdle NB 2", "Poisson (id)", "Poisson (log)"),
  col = c(4, 4, 2, 3), lty = c(1, 2, 1, 1), lwd = 1.5, bty = "n")

## alternative displays: Q-Q residuals plot, barplot, residuals vs. fitted
if(require("topmodels")) {
par(mfrow= c(3, 2))
qqrplot(cs_p, range = c(0.05, 0.95), main = "Q-Q residuals plot: Poisson")
qqrplot(cs_hnb, range = c(0.05, 0.95), main = "Q-Q residuals plot: Hurdle NB")
} else {
par(mfrow= c(2, 2))
}

barplot(t(matrix(c(r_p$observed, r_p$expected), ncol = 2,
  dimnames = list(r_p$x, c("Observed", "Expected")))),
  beside = TRUE, main = "Barplot: Poisson",
  xlab = "satellites", ylab = "Frequency",
  legend.text = TRUE, args.legend = list(x = "topright", bty = "n"))
barplot(t(matrix(c(r_hnb$observed, r_hnb$expected), ncol = 2,
  dimnames = list(r_hnb$x, c("Observed", "Expected")))),
  beside = TRUE, main = "Barplot: Hurdle NB",
  xlab = "satellites", ylab = "Frequency",
  legend.text = TRUE, args.legend = list(x = "topright", bty = "n"))

plot(predict(cs_p, type = "response"),
  residuals(cs_p, type = "pearson"),
  xlab = "Fitted values", ylab = "Pearson residuals",
  main = "Residuals vs. fitted: Poisson")
plot(predict(cs_hnb, type = "response"),
  residuals(cs_hnb, type = "pearson"),
  xlab = "Fitted values", ylab = "Pearson residuals",
  main = "Residuals vs. fitted: Hurdle NB")
```

`disptest` *Dispersion Tests*

Description

Tests the null hypothesis of equidispersion in Poisson GLMs against the alternative of overdispersion and/or underdispersion.

Usage

```
disptest(object,
  type = c("lrtNB2", "scoreNB2", "scoreNB2adj", "scoreNB1", "scoreNB1adj", "scoreKatz"),
   trafo = NULL, alternative = c("greater", "two.sided", "less"))
```

Arguments

`object`	a fitted Poisson GLM of class `"glm"` as fitted by `glm` with family `poisson`.
`type`	type of test, one of `lrtNB2`, `scoreNB2`, `scoreNB2adj`, `scoreNB1`, `scoreNB1adj`, `scoreKatz`. See details.
`trafo`	a specification of the alternative (see also details), can be numeric or a (positive) function or `NULL` (the default).
`alternative`	a character string specifying the alternative hypothesis: `"greater"` corresponds to overdispersion, `"less"` to underdispersion and `"two.sided"` to either one.

Details

The standard Poisson GLM models the (conditional) mean $\mathsf{E}[y] = \mu$ which is assumed to be equal to the variance $\mathsf{VAR}[y] = \mu$. `disptest` assesses the hypothesis that this assumption holds (equidispersion) against the alternative that the variance is of the form:

$$\mathsf{VAR}[y] \quad = \quad \mu \; + \; \alpha \cdot \mathrm{trafo}(\mu).$$

Overdispersion corresponds to $\alpha > 0$ and underdispersion to $\alpha < 0$. The coefficient α can be estimated by an auxiliary OLS regression and tested with the corresponding t (or z) statistic which is asymptotically standard normal under the null hypothesis.

Common specifications of the transformation function trafo are $\mathrm{trafo}(\mu) = \mu^2$ or $\mathrm{trafo}(\mu) = \mu$. The former corresponds to a negative binomial (NB) model with quadratic variance function (called NB2 by Cameron and Trivedi, 2005), the latter to a NB model with linear variance function (called NB1 by Cameron and Trivedi, 2005) or quasi-Poisson model with dispersion parameter, i.e.,

$$\mathsf{VAR}[y] \quad = \quad (1 + \alpha) \cdot \mu = \mathrm{dispersion} \cdot \mu.$$

By default, for `trafo = NULL`, the latter dispersion formulation is used in `dispersiontest`. Otherwise, if `trafo` is specified, the test is formulated in terms of the parameter α. The transformation `trafo` can either be specified as a function or an integer corresponding to the function `function(x) x^trafo`, such that `trafo = 1` and `trafo = 2` yield the linear and quadratic formulations respectively.

Type `"lrtNB2"` is the LRT comparing the classical Poisson and negative binomial regression models. Note that this test has a non-standard null distribution here, since the negative binomial shape parameter (called `theta` in `glm.nb`) is on the boundary of the parameter space under the null hypothesis. Hence the asymptotic distribution of the LRT is that of the arithmetic mean of a point mass at zero and a χ^2_1 distribution, implying that the p-value is half that of the classical case.

Type `"scoreNB2"` corresponds to the statistic T_1 in Dean and Lawless (1989), type `"scoreNB2adj"` is their T_a. `"scoreNB2"` also appears in Lee (1986). Type `"scoreNB1"` corresponds to the statistic P_C in Dean (1992), type `"scoreNB1adj"` is her P'_C. Type `"scoreKatz"` is the score test against Katz alternatives derived by Lee (1986), these distributions permit overdispersion as well as underdispersion. The score tests against NB1 and NB2 alternatives are also the score tests against Generalized Poisson type 1 and type 2 alternatives (Yang, Hardin, and Addy, 2009).

Value

An object of class "htest".

References

Cameron AC, Trivedi PK (1990). “Regression-based Tests for Overdispersion in the Poisson Model”. *Journal of Econometrics*, **46**, 347–364.

Cameron AC, Trivedi PK (2005). *Microeconometrics: Methods and Applications*. Cambridge: Cambridge University Press.

Cameron AC, Trivedi PK (2013). *Regression Analysis of Count Data*, 2nd ed. Cambridge: Cambridge University Press.

Dean CB (1992). “Testing for Overdispersion in Poisson and Binomial Regression Models”. *Journal of the American Statistical Association*, **87**, 451–457.

Dean C, Lawless JF (1989). “Tests for Detecting Overdispersion in Poisson Regression Models”. *Journal of the American Statistical Association*, **84**, 467–472.

Jaggia S, Thosar S (1993). “Multiple Bids as a Consequence of Target Management Resistance: A Count Data Approach”. *Review of Quantitative Finance and Accounting*, **3**, 447–457.

Lee LF (1986). “Specification Test for Poisson Regression Models”. *International Economic Review*, **27**, 689–706.

Yang Z, Hardin JW, Addy CL (2009). “A Note on Dean’s Overdispersion Test”. *Journal of Statistical Planning and Inference*, **139** (10), 3675–3678.

See Also

glm, poisson, glm.nb

Examples

```
## Data with overdispersion
data("RecreationDemand", package = "AER")
rd_p <- glm(trips ~ ., data = RecreationDemand, family = poisson)

## Cameron and Trivedi (2013), p. 248
disptest(rd_p, type = "lrtNB2", alternative = "greater")


## Data with underdispersion
data("TakeoverBids", package = "countreg")
tb_p <- glm(bids ~ . + I(size^2), data = TakeoverBids, family = poisson)

## Jaggia and Thosar (1993), Table 3
## testing overdispersion
disptest(tb_p, type = "scoreNB2", alternative = "greater")
disptest(tb_p, type = "scoreNB2adj", alternative = "greater")

## testing underdispersion
disptest(tb_p, type = "scoreKatz", alternative = "two.sided")
```

FLXMRnegbin *FlexMix Interface to Negative Binomial Regression Models*

Description

FlexMix driver for fitting of negative binomial regression models.

Usage

```
FLXMRnegbin(formula = . ~ ., theta = NULL, offset = NULL,
  control = list(reltol = .Machine$double.eps^(1/1.5), maxit = 500))
```

Arguments

`formula`	formula. This is interpreted relative to the formula specified in the call to `flexmix` using `update.formula`. Default is to use the original `flexmix` model formula.
`theta`	numeric or `NULL`. Value of the theta parameter of the negative binomial model. If `NULL`, theta is estimated along with the regression coefficients.
`offset`	numeric. Optional offset vector for the linear predictor.
`control`	list with control parameters passed to `optim`.

Details

The driver function `FLXMRnegbin` enables estimation of finite mixtures of negative binomial regression models via `flexmix` or `stepFlexmix`. The driver is modeled after `FLXMRglm` and supports both fixed and unknown theta. In the M-step for fixed theta, `glm.fit` is employed along with the `negative.binomial` family. If the `theta` is unknown and has be estimated along with the regression coefficients, direct optimization using `optim` with analytical gradients is employed.

Value

An object of class `FLXMRglm`.

See Also

`flexmix`, `stepFlexmix`, `FLXMRglm`, `negative.binomial`

Examples

```
## artificial data from a two-component mixture of geometric regressions
set.seed(1)
d <- data.frame(x = runif(500, -1, 1))
d$cluster <- rep(1:2, each = 250)
d$y <- rnbinom(500, mu = exp(c(1, -1)[d$cluster] + c(0, 3)[d$cluster] * d$x), size = 1)

if(require("flexmix")) {
```

```
## fit mixture models with known correct theta and unknown theta
fm1 <- flexmix(y ~ x, data = d, k = 2, model = FLXMRnegbin(theta = 1))
fm0 <- flexmix(y ~ x, data = d, k = 2, model = FLXMRnegbin())

## parameter recovery
parameters(fm1)
parameters(fm0)

## refit to obtain joint summary
summary(refit(fm1, gradient = NULL))
summary(refit(fm0, gradient = NULL))

## refitting both components manually for rootograms
rf1 <- lapply(1:2, function(i)
  nbreg(y ~ x, data = d, theta = 1, weights = posterior(fm1)[,i]))
rf0 <- lapply(1:2, function(i)
  nbreg(y ~ x, data = d, weights = posterior(fm0)[,i]))

## Rootograms
if(require("topmodels")) {
par(mfrow = c(1, 2))

r11 <- rootogram(rf1[[1]])
r12 <- rootogram(rf1[[2]])

r01 <- rootogram(rf0[[1]])
r02 <- rootogram(rf0[[2]])

rootogram(glm.nb(y ~ x, data = d))
plot(r01)
plot(r02)
}
}

## Not run:
## two-component mixture model fro NMES1988 physician office visits
## (fitting takes some time...)
if(require("flexmix") & require("AER")) {

## data from AER
data("NMES1988", package = "AER")
nmes <- NMES1988[, c(1, 7:8, 13, 15, 18:19)]

## single-component model
nmes_nb <- glm.nb(visits ~ ., data = nmes)

## two-component model
set.seed(1090)
nmes_fnb <- stepFlexmix(visits ~ ., data = nmes, k = 2, model = FLXMRnegbin())

## refit to obtain summary with estimate of joint covariance matrix
summary(refit(nmes_fnb, gradient = NULL))
```

```
## refit individual models manually for rootograms
nmes_fnb_rf <- lapply(1:2, function(i)
  nbreg(visits ~ ., data = nmes, weights = posterior(nmes_fnb)[,i]))

par(mfrow = c(1, 3))
rootogram(nmes_nb, main = "Negative Binomial", xlim = c(0, 50), ylim = c(-1, 25))
rootogram(nmes_fnb_rf[[1]], main = "Mixture Negative Binomial (Component 1)",
  xlim = c(0, 50), ylim = c(-1, 25))
rootogram(nmes_fnb_rf[[2]], main = "Mixture Negative Binomial (Component 2)",
  xlim = c(0, 50), ylim = c(-1, 25))
}

## End(Not run)
```

GSOEP *Health Services (GSOEP)*

Description

Unbalanced panel concerning the usage of health services in Germany for the years 1984-1988, 1991 and 1994 consisting of 27,326 observations on 22 variables taken from the German Socioeconomic Panel (GSOEP). The number of visits to a doctor and the number of inpatient hospital visits are given along with several personal socioeconomic characteristics.

Usage

```
data("GSOEP")
```

Format

A data frame with 27,326 observations on 22 variables.

id identification number indicating 1 to 7 observations for each of the 7,293 cases.

female factor indicating whether a person is female (yes or no).

year factor giving the calendar year of the observation.

age age in years.

hsat personal health satisfaction judged on a scale from 0 (low) to 10 (high). See *Details*.

handdum dummy variable indicating whether a person is handicapped (=1) or not (=0). See *Details*.

handper degree of handicap expressed in percent from 0 to 100. See *Details*.

hhninc monthly household net income in 1'000 German marks.

hhkids factor indicating whether a household has kids below the age of 16 (yes or no).

educ years of schooling. See *Details*.

married factor indicating whether a person is married (yes or no).

school factor giving the highest schooling degree. `haupt` for Hauptschule, `real` for Realschule, `fach` for Fachschule and `abitur` for Gymnasium.

univ factor indicating whether a person has a university diploma (`yes` or `no`).

working factor indicating whether a person is employed (`yes` or `no`).

bluec factor indicating whether a person is a blue collar worker (`yes` or `no`).

whitec factor indicating whether a person is a white collar worker (`yes` or `no`).

self factor indicating whether a person is self employed (`yes` or `no`).

civil factor indicating whether a person is a civil servant (`yes` or `no`).

docvis number of doctor visits within the last three months.

hospvis number of hospital visits within the last calendar year.

public factor indicating whether a person is insured in public health insurance (`yes` or `no`).

addon factor indicating whether a person is insured in add-on insurance (`yes` or `no`).

Details

The data were first used by Riphahn, Wambach and Million (2003).

As noted by Greene (2007, 2008), there are 40 observations on `hsat` taking on values between 6 and 7 although the variable is supposed to be an integer. For three other variables similar problems occur: By construction, `handdum` should be a dummy variable but there are 18 observations taking on values between 0 and 1. Also, for `handper` 3,290 observations are not integer-valued. Finally, the precision of 148 values of `educ` is very suspicious.

As of 2013, a comment and a reply on data issues are available from the JAE archive. No attempts are made to fix these issues here. Data is used 'as is' in order to replicate Greene (2007, 2008, 2011).

Source

Journal of Applied Econometrics Data Archive.

http://qed.econ.queensu.ca/jae/2003-v18.4/riphahn-wambach-million/

References

Greene WH (2007). "Functional Form and Heterogeneity in Models for Count Data". *Foundations and Trends in Econometrics*, **1**(2), 113–218.

Greene WH (2008). "Functional Forms for the Negative Binomial Model for Count Data". *Economics Letters*, **99**, 585–590.

Greene WH (2011). *Econometric Analysis*, 7th edition. Upper Saddle River, NJ: Prentice Hall.

Riphahn RT, Wambach A, Million A (2003). "Incentive Effects in the Demand for Health Care: A Bivariate Panel Count Data Estimation". *Journal of Applied Econometrics*, **18**(4), 387–405.

Examples

```
data("GSOEP")

if(exists("nbp.u") & exists("nbp.o")) {

# We fit an NB1, an NB2 and an NBP model as in Greene (2007) ...
model.u <- docvis ~ age + I(age^2) + hsat + handdum + handper + married + educ +
           hhninc + hhkids + self + civil + bluec + working + public + addon + year
fm.nb1.u <- nbp.u(model.u, data = subset(GSOEP, female == "no"), p.fix = 1)
fm.nb2.u <- nbp.u(model.u, data = subset(GSOEP, female == "no"), p.fix = 2)
fm.nbp.u <- nbp.u(model.u, data = subset(GSOEP, female == "no"))

# ... and show that the former two are inferior to the last one by standard likelihood ratio tests
library("lmtest")
lrtest(fm.nb1.u, fm.nbp.u)
lrtest(fm.nb2.u, fm.nbp.u)

# model with observed heterogeneity
model.o <- docvis ~ age + I(age^2) + hsat + handdum + handper + married + educ +
       hhninc + hhkids + self + civil + bluec + working + public + addon + year | hhninc + educ
## fm.nbp.o <- nbp.o(model.o, data = subset(GSOEP, female == "no"))

# visualization of the fit
# plot(fm.nbp.o)

# things to replicate from Greene (2011):
# Ex. 11.16, Tab 11.13, pp 451-452: nonlinear regression
# Tab. 14.2, pp 580--581: logistic and Poisson
# Tab. 14.10, pp 611-613: geometric regression
# Ex. 17.7: binary Chow test (maybe not?)
# Tab 18.14, p 850: Poisson, various negbins
# Tab 18.17, p 860 (panel models, some of)
# Tab 18.19, p 866: hurdle models

# things to replicate from Greene (2008):
# Tab 2: Poisson, various negbins, presumably same as book

}
```

`hnbinom-extensions` *Extension of the Hurdle Negative Binomial Distribution*

Description

Score function, hessian, mean, and, variance for the (zero-)hurdle negative binomial distribution with parameters `mu` (= mean of the underlying negative binomial distribution), dispersion parameter `theta` (or equivalently `size`), and hurdle crossing probability `pi` (i.e., `1 - pi` is the probability for observed zeros).

Usage

```
shnbinom(x, mu, theta, size, pi, parameter = c("mu", "theta", "pi"), drop = TRUE)
hhnbinom(x, mu, theta, size, pi, parameter = c("mu", "theta", "pi"), drop = TRUE)
mean_hnbinom(mu, theta, size, pi, drop = TRUE)
var_hnbinom(mu, theta, size, pi, drop = TRUE)
```

Arguments

`x`	vector of (positive integer) quantiles.
`mu`	vector of non-negative means of the underlying negative binomial distribution.
`theta, size`	vector of strictly positive dispersion parameters (shape parameter of the gamma mixing distribution). Only one of `theta` or `size` must be specified.
`pi`	vector of hurdle crossing probabilities (i.e., `1 - pi` is the probability for observed zeros).
`parameter`	character. Should the derivative with respect to `"mu"` and/or `"theta"` and/or `"pi"` be computed?
`drop`	logical. Should the result be a matrix (`drop = FALSE`) or should the dimension be dropped (`drop = TRUE`, the default)?

Details

The underlying negative binomial distribution has density

$$f(x) = \frac{\Gamma(x+\theta)}{\Gamma(\theta)x!} \cdot \frac{\mu^y \theta^\theta}{(\mu+\theta)^{y+\theta}}$$

for $x = 0, 1, 2, \ldots$. The hurdle density is then simply obtained as

$$g(x) = \pi \cdot \frac{f(x)}{1 - f(0)}$$

for $x = 1, 2, \ldots$ and $g(0) = 1 - \pi$, respectively.

Value

`shnbinom` gives the score function (= derivative of the log-density with respect to mu and/or theta and/or pi). `hhnbinom` gives the hessian (= 2nd derivative of the log-density with respect to mu and/or theta and/or pi). `mean_hnbinom` and `var_hnbinom` give the mean and the variance, respectively.

See Also

`dhnbinom`, `dnbinom`, `hurdle`

`hpois-extensions` *Extension of the Hurdle Poisson Distribution*

Description

Score function, hessian, mean, and, variance for the (zero-)hurdle Poisson distribution with parameters `mu` (= mean of the underlying Poisson distribution) and hurdle crossing probability `pi` (i.e., `1 - pi` is the probability for observed zeros).

Usage

```
shpois(x, lambda, pi, parameter = c("lambda", "pi"), drop = TRUE)
hhpois(x, lambda, pi, parameter = c("lambda", "pi"), drop = TRUE)
mean_hpois(lambda, pi, drop = TRUE)
var_hpois(lambda, pi, drop = TRUE)
```

Arguments

`x`	vector of (positive integer) quantiles.
`lambda`	vector of non-negative means of the underlying Poisson distribution.
`pi`	vector of hurdle crossing probabilities (i.e., `1 - pi` is the probability for observed zeros).
`parameter`	character. Should the derivative with respect to `"lambda"` and/or `"pi"` be computed?
`drop`	logical. Should the result be a matrix (`drop = FALSE`) or should the dimension be dropped (`drop = TRUE`, the default)?

Details

The underlying Poisson distribution has density

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

for $x = 0, 1, 2, \ldots$. The hurdle density is then simply obtained as

$$g(x) = \pi \cdot \frac{f(x)}{1 - f(0)}$$

for $x = 1, 2, \ldots$ and $g(0) = 1 - \pi$, respectively.

Value

`shpois` gives the score function (= derivative of the log-density with respect to lambda and/or pi). `hhpois` gives the hessian (= 2nd derivative of the log-density with respect to lambda and/or pi). `mean_hpois` and `var_hpois` give the mean and the variance, respectively.

See Also

`dhpois`, `dpois`, `hurdle`

hurdle *Hurdle Models for Count Data Regression*

Description

Fit hurdle regression models for count data via maximum likelihood.

Usage

```
hurdle(formula, data, subset, na.action, weights, offset,
  dist = c("poisson", "negbin", "geometric", "binomial"),
  zero.dist = c("binomial", "poisson", "negbin", "geometric"),
  link = c("logit", "probit", "cloglog", "cauchit", "log"),
  size = NULL, control = hurdle.control(...),
  model = TRUE, y = TRUE, x = FALSE, ...)
```

Arguments

`formula`	symbolic description of the model, see details.
`data`, `subset`, `na.action`	arguments controlling formula processing via `model.frame`.
`weights`	optional numeric vector of weights.
`offset`	optional numeric vector with an a priori known component to be included in the linear predictor of the count model. See below for more information on offsets.
`dist`	character specification of count model family.
`zero.dist`	character specification of the zero hurdle model family.
`link`	character specification of link function in the binomial zero hurdle (only used if `zero.dist = "binomial"`.
`size`	size parameter in case the a binomial count model is used (`dist = "binomial"`). By default the maximum count is used.
`control`	a list of control arguments specified via `hurdle.control`.
`model`, `y`, `x`	logicals. If `TRUE` the corresponding components of the fit (model frame, response, model matrix) are returned.
`...`	arguments passed to `hurdle.control` in the default setup.

Details

Hurdle count models are two-component models with a truncated count component for positive counts and a hurdle component that models the zero counts. Thus, unlike zero-inflation models, there are *not* two sources of zeros: the count model is only employed if the hurdle for modeling the occurence of zeros is exceeded. The count model is typically a truncated Poisson or negative binomial regression (with log link). The geometric distribution is a special case of the negative binomial with size parameter equal to 1. For modeling the hurdle (occurence of positive counts) either a binomial model can be employed or a censored count distribution. Binomial logit and censored geometric models as the hurdle part both lead to the same likelihood function and thus to the same coefficient estimates. A censored negative binomial model for the zero hurdle is only identified if there is at least one non-constant regressor with (true) coefficient different from zero (and if all coefficients are close to zero the model can be poorly conditioned).

The `formula` can be used to specify both components of the model: If a `formula` of type `y ~ x1 + x2` is supplied, then the same regressors are employed in both components. This is equivalent to `y ~ x1 + x2 | x1 + x2`. Of course, a different set of regressors could be specified for the zero hurdle component, e.g., `y ~ x1 + x2 | z1 + z2 + z3` giving the count data model `y ~ x1 + x2` conditional on (`|`) the zero hurdle model `y ~ z1 + z2 + z3`.

Offsets can be specified in both parts of the model pertaining to count and zero hurdle model: `y ~ x1 + offset(x2) | z1 + z2 + offset(z3)`, where `x2` is used as an offset (i.e., with coefficient fixed to 1) in the count part and `z3` analogously in the zero hurdle part. By the rule stated above `y ~ x1 + offset(x2)` is expanded to `y ~ x1 + offset(x2) | x1 + offset(x2)`. Instead of using the `offset()` wrapper within the `formula`, the `offset` argument can also be employed which sets an offset only for the count model. Thus, `formula = y ~ x1` and `offset = x2` is equivalent to `formula = y ~ x1 + offset(x2) | x1`.

All parameters are estimated by maximum likelihood using `optim`, with control options set in `hurdle.control`. Starting values can be supplied, otherwise they are estimated by `glm.fit` (the default). By default, the two components of the model are estimated separately using two `optim` calls. Standard errors are derived numerically using the Hessian matrix returned by `optim`. See `hurdle.control` for details.

The returned fitted model object is of class `"hurdle"` and is similar to fitted `"glm"` objects. For elements such as `"coefficients"` or `"terms"` a list is returned with elements for the zero and count components, respectively. For details see below.

A set of standard extractor functions for fitted model objects is available for objects of class `"hurdle"`, including methods to the generic functions `print`, `summary`, `coef`, `vcov`, `logLik`, `residuals`, `predict`, `fitted`, `terms`, `model.matrix`. See `predict.hurdle` for more details on all methods.

Value

An object of class `"hurdle"`, i.e., a list with components including

`coefficients`	a list with elements `"count"` and `"zero"` containing the coefficients from the respective models,
`residuals`	a vector of raw residuals (observed - fitted),
`fitted.values`	a vector of fitted means,
`optim`	a list (of lists) with the output(s) from the `optim` call(s) for minimizing the negative log-likelihood(s),
`control`	the control arguments passed to the `optim` call,
`start`	the starting values for the parameters passed to the `optim` call(s),
`weights`	the case weights used,
`offset`	a list with elements `"count"` and `"zero"` containing the offset vectors (if any) from the respective models,
`n`	number of observations (with weights > 0),
`df.null`	residual degrees of freedom for the null model (= `n - 2`),

`df.residual`	residual degrees of freedom for fitted model,
`terms`	a list with elements `"count"`, `"zero"` and `"full"` containing the terms objects for the respective models,
`theta`	estimate of the additional θ parameter of the negative binomial model(s) (if negative binomial component is used),
`SE.logtheta`	standard error(s) for $\log(\theta)$,
`loglik`	log-likelihood of the fitted model,
`vcov`	covariance matrix of all coefficients in the model (derived from the Hessian of the `optim` output(s)),
`dist`	a list with elements `"count"` and `"zero"` with character strings describing the respective distributions used,
`link`	character string describing the link if a binomial zero hurdle model is used,
`linkinv`	the inverse link function corresponding to `link`,
`converged`	logical indicating successful convergence of `optim`,
`call`	the original function call,
`formula`	the original formula,
`levels`	levels of the categorical regressors,
`contrasts`	a list with elements `"count"` and `"zero"` containing the contrasts corresponding to `levels` from the respective models,
`model`	the full model frame (if `model = TRUE`),
`y`	the response count vector (if `y = TRUE`),
`x`	a list with elements `"count"` and `"zero"` containing the model matrices from the respective models (if `x = TRUE`).

References

Cameron AC, Trivedi PK (2013). *Regression Analysis of Count Data*, 2nd ed. New York: Cambridge University Press.

Cameron AC, Trivedi PK (2005). *Microeconometrics: Methods and Applications*. Cambridge: Cambridge University Press.

Mullahy J (1986). “Specification and Testing of Some Modified Count Data Models”. *Journal of Econometrics*. **33**, 341–365.

Zeileis A, Kleiber C, Jackman S (2008). “Regression Models for Count Data in R.” *Journal of Statistical Software*, **27**(8), 1–25. doi:10.18637/jss.v027.i08.

See Also

`hurdle.control`, `glm`, `glm.fit`, `glm.nb`, `zeroinfl`

Examples

```
## data
data("CrabSatellites", package = "countreg")
cs <- CrabSatellites[, c("satellites", "width", "color")]
cs$color <- as.numeric(cs$color)

## logit-poisson
## "satellites ~ ." is the same as "satellites ~ . | .", i.e.
## "satellites ~ width + color | width + color"
fm_hp1 <- hurdle(satellites ~ ., data = cs)
summary(fm_hp1)

## geometric-poisson
fm_hp2 <- hurdle(satellites ~ ., data = cs, zero = "geometric")
summary(fm_hp2)

## logit and geometric model are equivalent
coef(fm_hp1, model = "zero") - coef(fm_hp2, model = "zero")

## logit-negbin
fm_hnb1 <- hurdle(satellites ~ ., data = cs, dist = "negbin")
summary(fm_hnb1)

## negbin-negbin
## (poorly conditioned zero hurdle, note increased standard errors)
fm_hnb2 <- hurdle(satellites ~ ., data = cs, dist = "negbin", zero = "negbin")
summary(fm_hnb2)
```

`hurdle.control` *Control Parameters for Hurdle Count Data Regression*

Description

Various parameters that control fitting of hurdle regression models using `hurdle`.

Usage

```
hurdle.control(method = "BFGS", maxit = 10000, trace = FALSE,
  separate = TRUE, start = NULL, hessian = TRUE, ...)
```

Arguments

`method`	characters string specifying the `method` argument passed to `optim`.
`maxit`	integer specifying the `maxit` argument (maximal number of iterations) passed to `optim`.
`trace`	logical or integer controlling whether tracing information on the progress of the optimization should be produced (passed to `optim`).

`separate`	logical. Should the estimation of the parameters in the truncated count component and hurdle zero component be carried out separately? See details.
`start`	an optional list with elements `"count"` and `"zero"` (and potentially `"theta"`) containing the coefficients for the corresponding component.
`hessian`	logical. Should the Hessian be computed to derive an estimate of the variance-covariance matrix? If `FALSE`, the variance-covariance matrix contains only `NA`s.
`...`	arguments passed to `optim`.

Details

All parameters in `hurdle` are estimated by maximum likelihood using `optim` with control options set in `hurdle.control`. Most arguments are passed on directly to `optim`, only `trace` is also used within `hurdle` and `separate`/`start` control how `optim` is called.

Starting values can be supplied via `start` or estimated by `glm.fit` (default).

If `separate = TRUE` (default) the likelihoods of the truncated count component and the hurdle zero component will be maximized separately, otherwise the joint likelihood is set up and maximized. In case of `separate = FALSE` and both `dist == "negbin"` and `zero.dist == "negbin"` the theta parameter is restricted to be identical across both negative binomial distributions.

Standard errors are derived numerically using the Hessian matrix returned by `optim`. To supply starting values, `start` should be a list with elements `"count"` and `"zero"` and potentially `"theta"` (a named vector, for models with negative binomial components only) containing the starting values for the coefficients of the corresponding component of the model.

Value

A list with the arguments specified.

See Also

`hurdle`

Examples

```
data("CrabSatellites", package = "countreg")

## default start values
fm1 <- hurdle(satellites ~ width + as.numeric(color), data = CrabSatellites,
  dist = "negbin", zero = "negbin")

## user-supplied start values and other options
fm2 <- hurdle(satellites ~ width + as.numeric(color), data = CrabSatellites,
  dist = "negbin",
  zero = "negbin",
  trace = TRUE,
  separate = FALSE,
  start = list(count = c(0.5, 0, 0),
          zero = c(-10, 0.5, -0.5),
          theta = c(count = 1, zero = 1)))
```

hurdletest *Testing for the Presence of a Zero Hurdle*

Description

Wald test of the null hypothesis that no zero hurdle is required in hurdle regression models for count data.

Usage

```
hurdletest(object, ...)
```

Arguments

`object`	A fitted model object of class `"hurdle"` as returned by `hurdle`, see details for more information.
`...`	arguments passed to `linearHypothesis`.

Details

If the same count distribution and the same set of regressors is used in the hurdle model for both, the count component and the zero hurdle component, then a test of pairwise equality between all coefficients from the two components assesses the null hypothesis that no hurdle is needed in the model.

The function `hurdletest` is a simple convenience interface to the function `linearHypothesis` from the **car** packages that can be employed to carry out a Wald test for this hypothesis.

Value

An object of class `"anova"` as returned by `linearHypothesis`.

References

Cameron AC, Trivedi PK (1998). *Regression Analysis of Count Data*. New York: Cambridge University Press.

Cameron AC, Trivedi PK (2005). *Microeconometrics: Methods and Applications*. Cambridge: Cambridge University Press.

See Also

`hurdle`, `linearHypothesis`

Examples

```
data("CrabSatellites", package = "countreg")
cs <- CrabSatellites[, c("satellites", "width", "color")]
cs$color <- as.numeric(cs$color)
fm <- hurdle(satellites ~ ., data = cs, dist = "negbin", zero = "negbin")
if(require("car")) hurdletest(fm)
```

`MBnegbin` *mboost Families for Binary, (Zero-Truncated) Negative Binomial and Zero-Truncated Poisson Regression*

Description

Family generators for model-based boosting of count data regressions using mboost.

Usage

```
MBbinomial(link = "logit")

MBnegbin(theta = NULL, link = "log",
  control = list(reltol = .Machine$double.eps^(1/1.5), maxit = 500))

MBztpoisson(link = "log",
  control = list(reltol = .Machine$double.eps^(1/1.5), maxit = 500))

MBztnegbin(theta = NULL, link = "log",
  control = list(reltol = .Machine$double.eps^(1/1.5), maxit = 500))
```

Arguments

`link`	character or object of class `"link-glm"` for the link function linking the expectation and the predictor.
`theta`	numeric or `NULL`. Value of the theta parameter of the negative binomial model. If `NULL`, theta is estimated along with the regression coefficients.
`control`	list with control parameters passed to `optim`.

Details

The family generators `MBbinomial`, `MBnegbin`, `MBztpoisson`, `MBztnegbin` enable boosting of binary regressions, negative binomial count regressions, zero-truncated Poisson count regressions, and zero-truncated negative binomial count regressions, respectively. Family `MBbinomial` is comparable to `Binomial` but supports any link function (not just logit and probit). Family `MBnegbin` is comparable to `NBinomial` but is typically much faster because the nuisance parameter `theta` is estimated using analytical gradients (via `optim`) and setting better starting values. `MBztpoisson` and `MBztnegbin` enable zero-truncated Poisson and negative binomial regressions so that also the count parts of hurdle models can be easily estimated.

Value

An object of class `boost_family_glm`.

See Also

`mboost`, `glmboost`, `gamboost`, `Binomial`, `Poisson`, `NBinomial`

Examples

```
### Negative binomial regression for CrabSatellites ----------------------------

if(require("mboost")) {
## crab satellite data using ordered factors as numeric
data("CrabSatellites", package = "countreg")
CrabSatellites <- transform(CrabSatellites,
  color = as.numeric(color),
  spine = as.numeric(spine)
)

## comparison of ML and boosting with NBinomial() vs. MBnegbin()
system.time(m0 <- glm.nb(satellites ~ width + color, data = CrabSatellites))
system.time(m1 <- glmboost(satellites ~ width + color, data = CrabSatellites,
  family = NBinomial(), control = boost_control(mstop = 500)))
system.time(m2 <- glmboost(satellites ~ width + color, data = CrabSatellites,
  family = MBnegbin(), control = boost_control(mstop = 500)))
## note that mstop is _not_ tuned here to (ab)use mboost to get the ML estimator

## compare coefficients
cbind(c(coef(m0), "theta" = m0$theta),
  c(coef(m1, off2int = TRUE, which = ""), nuisance(m1)),
  c(coef(m1, off2int = TRUE, which = ""), nuisance(m1))
)
}

### Hurdle regression for CrabSatellites using spline terms --------------------


if(require("mboost")) {
## ML estimation
g <- hurdle(satellites ~ width + color, data = CrabSatellites, dist = "negbin")
summary(g)

## boosting of zero hurdle
g0 <- gamboost(factor(satellites > 0) ~ bbs(width) + bbs(color, knots = 3),
  data = CrabSatellites, family = MBbinomial())
set.seed(0)
g0cv <- cvrisk(g0)
g0[mstop(g0cv)]

## boosting of count regression
g1 <- gamboost(satellites ~ bbs(width) + bbs(color, knots = 3),
  data = subset(CrabSatellites, satellites > 0), family = MBztnegbin())
set.seed(1)
g1cv <- cvrisk(g1)
g1[mstop(g1cv)]

par(mfrow = c(1, 2))

## optimal mstop values
plot(g0cv)
plot(g1cv)
## -> no effects in covariates for count part

## partial effects in zero hurdle
plot(g0)
## -> large effect of width, moderate effect of color with
## width effect almost linear
}


### Hurdle regression for RecreationDemand using linear terms ------------------

## Not run:
library("mboost")
data("RecreationDemand", package = "AER")

### Zero hurdle ##

## ML vs. boosting
z0 <- glm(factor(trips > 0) ~ ., data = RecreationDemand, family = binomial)
z1 <- glmboost(factor(trips > 0) ~ ., data = RecreationDemand, family = MBbinomial(),
  control = boost_control(mstop = 5000))
plot(z1)

## tune mstop
set.seed(0)
z1cv <- cvrisk(z1)
z1cv
plot(z1cv)
## very flat (presumably due to separation?)
## -> stop earlier manually
z1[3000]

## compare coefficients
cbind(coef(z0), coef(z1, off2int = TRUE, which = ""))
## -> some shrunken entirely to zero,
## coefficient of variable with separation (userfee) shrunken considerably


### Count (zero-truncated)

## ML and boosting count part
c0 <- zerotrunc(trips ~ ., data = subset(RecreationDemand, trips > 0), dist = "negbin")
c1 <- glmboost(trips ~ ., data = subset(RecreationDemand, trips > 0),
  family = MBztnegbin(), control = boost_control(mstop = 5000))
```

```
plot(c1)

## tune mstop
set.seed(0)
c1cv <- cvrisk(c1)
c1cv
plot(c1cv)

## use mstop from cvrisk
c1[mstop(c1cv)]

## compare coefficients
cbind(c(coef(c0), "theta" = c0$theta),
  c(coef(c1, off2int = TRUE, which = ""), nuisance(c1)))
## -> similar

## End(Not run)
```

`nbreg` *Negative Binomial Count Data Regression*

Description

Fit negative binomial regression models for count data via maximum likelihood

Usage

```
nbreg(formula, data, subset, na.action, weights, offset, theta = NULL,
  dist = "NB2", link = "log", link.theta = "log", control = nbreg.control(...),
  model = TRUE, y = TRUE, x = FALSE, z = FALSE, hessA = TRUE, ...)
```

Arguments

`formula`	symbolic description of the model, see details.
`data, subset, na.action`	arguments controlling formula processing via `model.frame`.
`weights`	optional numeric vector of weights.
`offset`	optional numeric vector with an a priori known component to be included in the linear predictor. See below for more information on offsets.
`theta`	numeric. Optional. If specified, then the dispersion parameter is not estimated.
`dist`	character specification of the NB type. Either `"NB2"` or `"NB1"`. Lowercase versions `"nb2"` and `"nb1"` are equivalent.
`link`	character specification of the link function for the mean. Currently, only `"log"` is supported.
`link.theta`	character specification of the link function for the dispersion parameter. Currently, only `"log"` is supported.

`control`	a list of control arguments specified via `nbreg.control`.
`model, y, x, z`	logicals. If `TRUE` the corresponding components of the fit (model frame, response, model matrix) are returned.
`hessA`	logical. If `TRUE`, then the analytical Hessian is used to compute the covariance matrix of the estimator.
`...`	currently not used.

Details

The Negative Binomial Distribution is often used to model count data with overdispersion. Cameron and Trivedi (2013) offer two parametrization, negative binomial type 2 (NB2) and type 1 (NB1). NB2 is parametrized as follows

$$f(y|\mu,\theta) = \frac{\Gamma(\theta+y)}{\Gamma(\theta)y!}\left(\frac{\theta}{\theta+\mu}\right)^{\theta}\left(\frac{\mu}{\theta+\mu}\right)^{y}.$$

For NB1 replace θ with $\mu\theta$ on the RHS.

This function further allows us to model the dispersion parameter with covariates via a two-part `formula`. If a `formula` of type `y ~ x1 + x2` is supplied, then the regressors are employed in the mean and the dispersion parameter is estimated as a constant, i.e. a standard NB2 or NB1 is estimated. This is equivalent to `y ~ x1 + x2 | 1`. If a `formula` of type `y ~ x1 + x2 + x3 | z1 + z2` is given, then the mean mu is modeled using `x1 + x2` and the dispersion parameter θ with `z1 + z2`. If `dist = "NB2"`, then the function estimates the NBH model.

Offsets can be specified in both the mean and dispersion parameter θ: `y ~ x1 + x2 + offset(x3) | z1 + offset(z2)`, where `x3` is used as an offset (i.e., with coefficient fixed to 1) in the mean mu and `z2` analogously in θ. By the rule stated above `y ~ x1 + offset(x2)` is equivalent to `y ~ x1 + offset(x2) | 1`. Instead of using the `offset()` wrapper within the `formula`, the `offset` argument can also be employed which sets an offset only for mu. Thus, `formula = y ~ x1` and `offset = x2` is equivalent to `formula = y ~ x1 + offset(x2) | 1`.

All parameters are estimated by maximum likelihood using `optim`, with control options set in `nbreg.control`. Starting values can be supplied or are estimated by a Poisson regression in `glm.fit` (the default, starting values of coefficients in θ are set to zero to ensure compatibility with NB1). Standard errors are derived analytically or numerically using the Hessian matrix returned by `optim`. See `nbreg.control` for details.

The returned fitted model object is of class `"nbreg"` and is similar to fitted `"glm"` objects.

A set of standard extractor functions for fitted model objects is available for objects of class `"nbreg"`, including methods to the generic functions `print`, `summary`, `coef`, `vcov`, `logLik`, `residuals`, `predict`, `fitted`, `terms`, `model.matrix`. See `predict.nbreg` for more details on all methods.

Value

An object of class `"nbreg"`, i.e., a list with components including

`coefficients`	a vector containing the coefficients from the mean,
`coefficients.theta`	a vector containing the coefficients from the dispersion parameter theta,

`residuals`	a vector of raw residuals (observed - fitted),
`fitted.values`	a vector of fitted means,
`optim`	a list with the output from the `optim` call for maximizing the log-likelihood,
`control`	the control arguments passed to the `optim` call,
`start`	the starting values for the parameters passed to the `optim` call,
`weights`	the case weights used,
`offset`	a list with elements `"mu"` and `"theta"` containing the offset vectors (if any) from the respective parameters,
`n`	number of observations (with weights > 0),
`df.null`	residual degrees of freedom for the null model,
`df.residual`	residual degrees of freedom for fitted model,
`terms`	a list with elements `"mu"`, `"theta"` and `"full"` containing the terms objects for the respective parameters,
`SE.logtheta`	standard error for $\log(\theta)$,
`loglik`	log-likelihood of the fitted model,
`vcov`	covariance matrix of all coefficients in the model (derived from the analytical Hessian (`hessA = TRUE`) or from the Hessian of the `optim` output (`hessA = FALSE`)),
`dist`	character string describing the type of NB distribution used,
`link`	character string describing the link of the mean,
`link.theta`	character string describing the link of the dispersion parameter theta,
`converged`	logical indicating successful convergence of `optim`,
`call`	the original function call,
`formula`	the original formula,
`levels`	levels of the categorical regressors,
`contrasts`	a list with elements `"mu"` and `"theta"` containing the contrasts corresponding to `levels` from the respective parts,
`model`	the full model frame (if `model = TRUE`),
`y`	the response count vector (if `y = TRUE`),
`x`	the model matrix for the mean (if `x = TRUE`),
`z`	the model matrix for the mean (if `z = TRUE`),

References

Cameron AC, Trivedi PK (2013). *Regression Analysis of Count Data*, 2nd ed. New York: Cambridge University Press.

See Also

`nbreg.control`, `glm`, `glm.fit`, `glm.nb`,

Examples

```
data("CrabSatellites", package = "countreg")

## NB2
fm_nb2 <- nbreg(satellites ~ width + color, data = CrabSatellites)

## NB1
fm_nb1 <- nbreg(satellites ~ width + color, data = CrabSatellites, dist = "NB1")

## NBH
fm_nbh <- nbreg(satellites ~ width + color | weight, data = CrabSatellites)

## NB1 with variable theta
fm_nb1h <- nbreg(satellites ~ width + color | weight, data = CrabSatellites,
                dist = "NB1")

## Example not run:
## data
# data("GSOEP", package = "countreg")
# gsoep <- subset(GSOEP, year == "1984")

## NB2
# fm_nb2 <- nbreg(docvis ~ educ + public + addon,
#                  data = gsoep)

## NB1
# fm_nb1 <- nbreg(docvis ~ educ + public + addon,
#                  data = gsoep, dist = "NB1")

## NBH
# fm_nbh <- nbreg(docvis ~ educ + public + addon | married + public,
#                  data = gsoep)

## NB1 with variable theta
# fm_nb1h <- nbreg(docvis ~ educ + public + addon | married + public,
#                  data = gsoep, dist = "NB1")
```

`nbreg.control` *Control Parameters for Negative Binomial Count Data Regression*

Description

Various parameters that control fitting of negative binomial regression models using `nbreg`.

Usage

```
nbreg.control(method = "BFGS", maxit = 10000, start = NULL, hessian = TRUE,
    dot = "separate", ...)
```

Arguments

`method`	characters string specifying the `method` argument passed to `optim`.
`maxit`	integer specifying the `maxit` argument (maximal number of iterations) passed to `optim`.
`start`	an optional list with elements `"mu"` and `"theta"` containing the coefficients for the corresponding component.
`hessian`	logical. Should the numerically approximated Hessian be computed to derive an estimate of the variance-covariance matrix? If `FALSE` and parameter `hessA = FALSE` in `nbreg()`, the variance-covariance matrix contains only `NA`s.
`dot`	character. Controls how two-part `Formula`'s are processed. See model.frame.Formula.
`...`	arguments passed to `optim`.

Details

All parameters in `nbreg` are estimated by maximum likelihood using `optim` with control options set in `nbreg.control`. Most arguments are passed on directly to `optim` and `start` controls the choice of starting values for calling `optim`.

Starting values can be supplied or are estimated by a Poisson regression in `glm.fit` (the default, starting values of coefficients in θ are set to zero to ensure compatibility with NB1). Standard errors are derived using the analytical Hessian matrix or by numerical approximation of the Hessian.

Value

A list with the arguments specified.

See Also

`nbreg`

Examples

```
data("CrabSatellites", package = "countreg")

## default start values
fm1 <- nbreg(satellites ~ width + as.numeric(color), data = CrabSatellites)

## user-supplied start values
fm2 <- nbreg(satellites ~ width + as.numeric(color), data = CrabSatellites,
                start = list(mu = c(0, 0, 0), theta = c(0.5)))
```

NegBinomial-extensions *Extension of the Negative Binomial Distribution*

Description

Score function, hessian, mean, and variance for the negative binomial distribution with parameters mu and size.

Usage

```
snbinom(x, mu, size, parameter = c("mu", "size"), drop = TRUE)
hnbinom(x, mu, size, parameter = c("mu", "size"), drop = TRUE)
mean_nbinom(mu, size, drop = TRUE)
var_nbinom(mu, size, drop = TRUE)
```

Arguments

x	vector of quantiles.
mu	_mean_ of distribution.
size	dispersion parameter. Must be strictly positive.
parameter	character. Derivatives are computed wrt this paramter.
drop	logical. Should the result be a matrix (drop = FALSE) or should the dimension be dropped (drop = TRUE, the default)?

Details

The negative binomial with mu and size (or theta) has density

$$f(y|\mu,\theta) = \frac{\Gamma(\theta+y)}{\Gamma(\theta)\cdot y!} \cdot \frac{\mu^y \cdot \theta^\theta}{(\mu+\theta)^{\theta+y}}, \quad y \in \{0,1,2,\dots\}$$

Derivatives of the log-likelihood ℓ wrt μ:

$$\frac{\partial \ell}{\partial \mu} = \frac{y}{\mu} - \frac{y+\theta}{\mu+\theta}$$

$$\frac{\partial^2 \ell}{\partial \mu^2} = -\frac{y}{\mu^2} + \frac{y+\theta}{(\mu+\theta)^2}$$

Derivatives wrt θ:

$$\frac{\partial \ell}{\partial \theta} = \psi_0(y+\theta) - \psi_0(\theta) + \log(\theta) + 1 - \log(\mu+\theta) - \frac{y+\theta}{\mu+\theta}$$

$$\frac{\partial^2 \ell}{\partial \theta^2} = \psi_1(y+\theta) - \psi_1(\theta) + \frac{1}{\theta} - \frac{2}{\mu+\theta} + \frac{y+\theta}{(\mu+\theta)^2}$$

ψ_0 and ψ_1 denote the digamma and trigamma function, respectively.

The derivative wrt μ and θ:

$$\frac{\partial^2 \ell}{\partial \mu \partial \theta} == \frac{y - \mu}{(\mu + \theta)^2}$$

Value

snbinom gives the score function, i.e., the 1st derivative of the log-density wrt mu or theta and hnbinom gives the hessian, i.e., the 2nd derivative of the log-density wrt mu and/or theta. mean and var give the mean and variance, respectively.

Note

No parameter prob—as in dnbinom, pnbinom, qnbinom and rnbinom—is implemented in the functions snbinom and hnbinom.

See Also

NegBinomial encompassing dnbinom, pnbinom, qnbinom and rnbinom.

Examples

```
## Simulate some data
set.seed(123)
y <- rnbinom(1000, size = 2, mu = 2)

## Plot log-likelihood function
par(mfrow = c(1, 3))
ll <- function(x) {sum(dnbinom(y, size = x, mu = 2, log = TRUE))}
curve(sapply(x, ll), 1, 4, xlab = expression(theta), ylab = "",
      main = "Log-likelihood")
abline(v = 2, lty = 3)

## Plot score function
curve(sapply(x, function(x) sum(snbinom(y, size = x, mu = 2, parameter = "size"))),
      1, 4, xlab = expression(theta), ylab = "", main = "Score")
abline(h = 0, lty = 3)
abline(v = 2, lty = 3)

## Plot hessian
curve(sapply(x, function(x) sum(hnbinom(y, size = x, mu = 2, parameter = "size"))),
      1, 4, xlab = expression(theta), ylab = "", main = "Hessian")
abline(v = 2, lty = 3)
```

`OralHealthNL` *Oral Health in Children in The Netherlands*

Description

Data from a study on oral health status and the preventive dental behaviors of 9-year-old children in The Netherlands.

Usage

```
data("OralHealthNL")
```

Format

A data frame containing 440 observations on 8 variables.

dmfs Numeric index of decayed, missing, and filled surfaces (DMFS) in deciduous teeth.

education Factor indicating whether the highest completed education level of the mother is `"high"` (senior general secondary education, HAVO, or higher) or `"low"`.

gender Factor indicating gender of the child (`"female"` or `"male"`).

ethnicity Factor indicating whether the mother is `"immigrant"` (born abroad) or `"native"` (born in The Netherlands).

brushing Factor indicating whether the frequency of brushing teeth is `"< 2"` or `">= 2"` times per day.

breakfast Factor indicating whether the frequency of having breakfast is `"7"` or `"< 7"` days per week.

fooddrink Factor indicating whether the frequency of food and drinks in addition to the three main meals is `"<= 7"` or `"> 7"` times per day.

corah Factor indicating whether Corah's Dental Anxiety score is `"< 13"` or `">= 13"` (see also below).

Details

The data are from the study "Oral Health in Children and Adolescents in The Netherlands" (Schuller et al. 2011). The aim of this study was to describe the oral health status and the preventive dental behaviors of children from different age groups (Dusseldorp et al. 2015). Here, the subset of children at the age of 9 years is provided as analyzed by Hofstetter et al. (2016).

The data collection consisted of a clinical oral examination and a questionnaire survey, using a repeated cross-sectional design. Data contained information about demographic variables (ethnicity and educational level), nutrition, children's dental attendance, oral self-care, and dental anxiety. The score on Corah's Dental Anxiety Questionnaire was used as a measure of dental anxiety. This questionnaire consists of four questions with answer categories from 1 (low anxiety) to 5 (high anxiety). A total Corah score was computed by taking the sum of the four items and then dichotomized into 'lower than 13' and 'higher than or equal to 13'.

Source

Supplementary materials for Hofstetter et al. (2016). doi:10.1159/000448197

References

Dusseldorp E, Kamphuis M, Schuller AA (2015). "Impact of Lifestyle Factors on Caries Experience in Three Different Age Groups: 9, 15, and 21-Year-Olds", *Community Dentistry and Oral Epidemiology*, **43**(1), 9–16. doi:10.1111/cdoe.12123

Hofstetter H, Dusseldorp E, Zeileis A, Schuller AA (2016). "Modeling Caries Experience: Advantages of the Use of the Hurdle Model", *Caries Research*, **50**(6), 517–526. doi:10.1159/000448197

Schuller AA, Poorterman JHG, van Kempen CPF, Dusseldorp E, van Dommelen P, Verrips GHW (2011). *Kies voor tanden: Een onderzoek naar mondgezondheid en preventief tandheelkundig gedrag van jeugdigen. Tussenmeting 2009, een vervolg op de reeks TJZ-onderzoeken*. TNO, Leiden.

Examples

```
## Load data and omit NAs and one dmfs outlier
data("OralHealthNL", package = "countreg")
head(OralHealthNL)
OralHealthNL <- na.omit(subset(OralHealthNL, dmfs < 40))

## Visualization: Is dmfs > 0?
par(mfrow = c(2, 4))
plot(factor(dmfs > 0, levels = c(TRUE, FALSE), labels = c("> 0", "= 0")) ~ .,
  data = OralHealthNL, ylab = "dmfs")

## Count: How large is log(dmfs) given dmfs > 0?
par(mfrow = c(2, 4))
plot(log(dmfs) ~ ., data = OralHealthNL, subset = dmfs > 0, ylab = "dmfs")

## Relevel the factor variables so that non-risk group is the reference
OralHealthNL <- transform(OralHealthNL,
  ethnicity = relevel(ethnicity, ref = "native"),
  brushing = relevel(brushing, ref = ">= 2"),
  breakfast = relevel(breakfast, ref = "7")
)

## Count regression models
zinb <- zeroinfl(dmfs ~ ., data = OralHealthNL, dist = "negbin")
zip  <- zeroinfl(dmfs ~ ., data = OralHealthNL, dist = "poisson")
hnb  <-   hurdle(dmfs ~ ., data = OralHealthNL, dist = "negbin")
hp   <-   hurdle(dmfs ~ ., data = OralHealthNL, dist = "poisson")

## Model comparisons (Table 3)
## Information criteria
cbind(AIC(hnb, zinb, hp, zip), BIC = BIC(hnb, zinb, hp, zip)[, 2])
## Negative binomial vs. Poisson
if(require("lmtest")) lrtest(hnb, hp)
if(require("lmtest")) lrtest(zinb, zip)
## Zero-inflation vs. hurdle
```

```
if(require("nonnest2")) vuongtest(zinb, hnb)

## Coefficients, odds ratios, and rate ratios
## Negative binomial hurdle model (Table 3)
summary(hnb)
exp(confint(hnb))
## Negative binomial zero-inflated model (Table 4)
summary(hnb)
exp(confint(zinb))

## Rootograms (top left: Figure 1)
if(require("topmodels")) {
par(mfrow = c(2, 2))
rootogram(lm(OralHealthNL$dmfs ~ 1),
  style = "standing", scale = "raw",
  breaks = 0:23 - 0.5, xlim = c(-0.5, 22.5),
  xlab = "dmfs", main = "Normal distribution")
rootogram(hnb,
  style = "standing", scale = "raw",
  width = 1, xlim = c(-0.5, 22.5),
  xlab = "dmfs", main = "Negative binomial hurdle model")
rootogram(lm(OralHealthNL$dmfs ~ 1),
  breaks = 0:23 - 0.5, xlim = c(-0.5, 22.5),
  xlab = "dmfs", main = "Normal distribution")
abline(h = c(-1, 1), lty = 2)
rootogram(hnb,
  width = 1, xlim = c(-0.5, 22.5),
  xlab = "dmfs", main = "Negative binomial hurdle model")
abline(h = c(-1, 1), lty = 2)
par(mfrow = c(1, 1))
}

## Number of zeros
c(dmfs = sum(OralHealthNL$dmfs == 0),
  ZINB = sum(predict(zinb, type = "density", at = 0)),
  Hurdle = sum(predict(hnb, type = "density", at = 0)))
## Correlation of observations and fitted means
cor(cbind(dmfs = OralHealthNL$dmfs,
  ZINB = fitted(zinb), HNB = fitted(hnb)))

## Bias-reduced logistic regression (due to separation)
if(require("brglm2")) {
br <- glm(
  factor(dmfs == 0, levels = c(TRUE, FALSE), labels = c("= 0", "> 0")) ~ .,
  data = OralHealthNL, family = binomial, method = "brglmFit")
print(coeftest(br), digits = 1)
}
```

Poisson-extensions *Extension of the Poisson Distribution*

Description

Score function, hessian, mean, and variance for the Poisson distribution with parameter `lambda`.

Usage

```
spois(x, lambda, parameter = "lambda", drop = TRUE)
hpois(x, lambda, parameter = "lambda", drop = TRUE)
mean_pois(lambda, drop = TRUE)
var_pois(lambda, drop = TRUE)
```

Arguments

`x`	vector of quantiles.
`lambda`	vector of (non-negative) means.
`parameter`	character. Derivatives are computed wrt this paramter. Note: Only `"lambda"` is implemented.
`drop`	logical. Should the result be a matrix (`drop = FALSE`) or should the dimension be dropped (`drop = TRUE`, the default)?

Details

The Poisson distribution has density

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

for $x = 0, 1, 2, \ldots$.

The score function is

$$s(\lambda) = \frac{x}{\lambda} - 1$$

The hessian is

$$h(\lambda) = -\frac{x}{\lambda^2}$$

Value

`spois` gives the score function, i.e., the 1st derivative of the log-density wrt lambda and `hpois` gives the hessian, i.e., the 2nd derivative of the log-density wrt lambda. `mean` and `var` give the mean and variance, respectively.

See Also

Poisson encompassing `dpois`, `ppois`, `qpois` and `rpois`.

Examples

```
## Simulate some data
set.seed(123)
y <- rpois(50, lambda = 3)

## Plot log-likelihood function
par(mfrow = c(1,3))
ll <- function(x) {sum(dpois(y, x, log = TRUE))}
curve(sapply(x, ll), 1, 5, xlab = expression(lambda), ylab = "",
      main = "Log-likelihood")
abline(v = 3, lty = 3)

## Plot score function
curve(sapply(x, function(x) sum(spois(y, x))), 1, 5,
      xlab = expression(lambda), ylab = "", main = "Score")
abline(h = 0, lty = 3)
abline(v = 3, lty = 3)

## Plot hessian
curve( sapply(x, function(x) sum(hpois(y, x))), 1, 5,
      xlab = expression(lambda), ylab = "", main = "Hessian")
abline(v = 3, lty = 3)
```

`predict.hurdle` *Methods for hurdle Objects*

Description

Methods for extracting information from fitted hurdle regression model objects of class `"hurdle"`.

Usage

```
## S3 method for class 'hurdle'
predict(object, newdata,
  type = c("mean", "variance", "quantile", "probability", "density", "loglikelihood", "parameters", "di
   model = c("full", "count", "zero", "truncated"),
   na.action = na.pass, at = NULL, drop = TRUE, ...)
## S3 method for class 'hurdle'
residuals(object, type = c("pearson", "response"), ...)

## S3 method for class 'hurdle'
coef(object, model = c("full", "count", "zero"), ...)
## S3 method for class 'hurdle'
vcov(object, model = c("full", "count", "zero"), ...)

## S3 method for class 'hurdle'
terms(x, model = c("full", "count", "zero"), ...)
## S3 method for class 'hurdle'
model.matrix(object, model = c("count", "zero"), ...)
```

Arguments

`object, x`	an object of class `"hurdle"` as returned by `hurdle`.
`newdata`	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
`type`	character specifying the type of predictions or residuals, respectively. For details see below.
`model`	character specifying for which component of the model the terms or model matrix should be extracted.
`na.action`	function determining what should be done with missing values in `newdata`. The default is to predict `NA`.
`at`	optionally, if `type = "prob"`, a numeric vector at which the probabilities are evaluated. By default `0:max(y)` is used where `y` is the original observed response.
`drop`	logical. Should predictions be returned in a data frame or (if possible) dropped to a vector (default).
`...`	currently not used.

Details

A set of standard extractor functions for fitted model objects is available for objects of class `"hurdle"`, including methods to the generic functions `print` and `summary` which print the estimated coefficients along with some further information. The `summary` in particular supplies partial Wald tests based on the coefficients and the covariance matrix (estimated from the Hessian in the numerical optimization of the log-likelihood). As usual, the `summary` method returns an object of class `"summary.hurdle"` containing the relevant summary statistics which can subsequently be printed using the associated `print` method.

The methods for `coef` and `vcov` by default return a single vector of coefficients and their associated covariance matrix, respectively, i.e., all coefficients are concatenated. By setting the `model` argument, the estimates for the corresponding model component can be extracted.

Both the `fitted` and `predict` methods can compute fitted responses. The latter additionally provides the predicted density (i.e., probabilities for the observed counts), the predicted mean from the count component (without zero hurdle) and the predicted ratio of probabilities for observing a non-zero count. The latter is the ratio of probabilities for a non-zero implied by the zero hurdle component and a non-zero count in the non-truncated count distribution. See also Appendix C in Zeileis et al. (2008).

The `residuals` method can compute raw residuals (observed - fitted) and Pearson residuals (raw residuals scaled by square root of variance function).

The `terms` and `model.matrix` extractors can be used to extract the relevant information for either component of the model.

A `logLik` method is provided, hence `AIC` can be called to compute information criteria.

See Also

`hurdle`

Examples

```
data("CrabSatellites", package = "countreg")
fm <- hurdle(satellites ~ 1 | width + color, data = CrabSatellites)

plot(residuals(fm) ~ fitted(fm))

coef(fm)
coef(fm, model = "zero")

summary(fm)
logLik(fm)
```

`predict.nbreg` *Methods for nbreg Objects*

Description

Methods for extracting information from fitted negative binomial count regression model objects of class "nbreg".

Usage

```
## S3 method for class 'nbreg'
predict(object, newdata,
  type = c("response", "prob", "theta", "parameters"), na.action = na.pass, ...)
## S3 method for class 'nbreg'
residuals(object, type = c("pearson", "deviance", "response"), ...)

## S3 method for class 'nbreg'
coef(object, model = c("full", "mu", "theta"), ...)
## S3 method for class 'nbreg'
vcov(object, model = c("full", "mu", "theta"), ...)

## S3 method for class 'nbreg'
terms(x, model = c("full", "mu", "theta"), ...)
## S3 method for class 'nbreg'
model.matrix(object, model = c("mu", "theta"), ...)
```

Arguments

`object, x`	an object of class "nbreg" as returned by `nbreg`.
`newdata`	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
`type`	character specifying the type of predictions or residuals, respectively. For details see below.
`na.action`	function determining what should be done with missing values in `newdata`. The default is to predict `NA`.

`model`	character specifying for which component of the model the terms or model matrix should be extracted.
`...`	currently not used.

Details

A set of standard extractor functions for fitted model objects is available for objects of class `"nbreg"`, including methods to the generic functions `print` and `summary` which print the estimated coefficients along with some further information. The `summary` in particular supplies partial Wald tests based on the coefficients and the covariance matrix. As usual, the `summary` method returns an object of class `"summary.nbreg"` containing the relevant summary statistics which can subsequently be printed using the associated `print` method.

The methods for `coef` and `vcov` by default return a single vector of coefficients and their associated covariance matrix, respectively, i.e., all coefficients are concatenated. By setting the `model` argument, the estimates for the corresponding model component can be extracted.

Both the `fitted` and `predict` methods can compute fitted responses. The latter additionally provides the predicted density (i.e., probabilities for the observed counts) and the predicted dispersion parameter theta. The `residuals` method can compute raw residuals (observed - fitted), Pearson residuals (raw residuals scaled by square root of variance function), and deviance residuals. The latter are only supported for negative binomial type 2 models (`dist = NB2`) (includes NBH).

A `logLik` method is provided, hence `AIC` can be called to compute information criteria.

See Also

`nbreg`

Examples

```
data("CrabSatellites", package = "countreg")
fm <- nbreg(satellites ~ width + color, data = CrabSatellites)

plot(residuals(fm, type = "pearson") ~ fitted(fm))

coef(fm)
summary(fm)
logLik(fm)
AIC(fm)
```

`predict.zeroinfl` *Methods for zeroinfl Objects*

Description

Methods for extracting information from fitted zero-inflated regression model objects of class `"zeroinfl"`.

Usage

```
## S3 method for class 'zeroinfl'
predict(object, newdata,
  type = c("mean", "variance", "quantile", "probability", "density", "loglikelihood", "parameters", "di
   model = c("full", "count", "zero", "truncated"),
   na.action = na.pass, at = NULL, drop = TRUE, ...)
## S3 method for class 'zeroinfl'
residuals(object, type = c("pearson", "response"), ...)

## S3 method for class 'zeroinfl'
coef(object, model = c("full", "count", "zero"), ...)
## S3 method for class 'zeroinfl'
vcov(object, model = c("full", "count", "zero"), ...)

## S3 method for class 'zeroinfl'
terms(x, model = c("full", "count", "zero"), ...)
## S3 method for class 'zeroinfl'
model.matrix(object, model = c("count", "zero"), ...)
```

Arguments

`object, x`	an object of class `"zeroinfl"` as returned by `zeroinfl`.
`newdata`	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
`type`	character specifying the type of predictions or residuals, respectively. For details see below.
`model`	character specifying for which component of the model the terms or model matrix should be extracted.
`na.action`	function determining what should be done with missing values in `newdata`. The default is to predict `NA`.
`at`	optionally, if `type = "prob"`, a numeric vector at which the probabilities are evaluated. By default `0:max(y)` is used where `y` is the original observed response.
`drop`	logical. Should predictions be returned in a data frame or (if possible) dropped to a vector (default).
`...`	currently not used.

Details

A set of standard extractor functions for fitted model objects is available for objects of class `"zeroinfl"`, including methods to the generic functions `print` and `summary` which print the estimated coefficients along with some further information. The `summary` in particular supplies partial Wald tests based on the coefficients and the covariance matrix (estimated from the Hessian in the numerical optimization of the log-likelihood). As usual, the `summary` method returns an object of class `"summary.zeroinfl"` containing the relevant summary statistics which can subsequently be printed using the associated `print` method.

The methods for `coef` and `vcov` by default return a single vector of coefficients and their associated covariance matrix, respectively, i.e., all coefficients are concatenated. By setting the `model` argument, the estimates for the corresponding model components can be extracted.

Both the `fitted` and `predict` methods can compute fitted responses. The latter additionally provides the predicted density (i.e., probabilities for the observed counts), the predicted mean from the count component (without zero inflation) and the predicted probability for the zero component. The `residuals` method can compute raw residuals (observed - fitted) and Pearson residuals (raw residuals scaled by square root of variance function).

The `terms` and `model.matrix` extractors can be used to extract the relevant information for either component of the model.

A `logLik` method is provided, hence `AIC` can be called to compute information criteria.

See Also

`zeroinfl`

Examples

```
data("CrabSatellites", package = "countreg")
fm_zip <- zeroinfl(satellites ~ 1 | width + color, data = CrabSatellites)

plot(residuals(fm_zip) ~ fitted(fm_zip))

coef(fm_zip)
coef(fm_zip, model = "count")

summary(fm_zip)
logLik(fm_zip)
```

`predict.zerotrunc` *Methods for zerotrunc Objects*

Description

Methods for extracting information from fitted zero-truncated count regression model objects of class `"zerotrunc"`.

Usage

```
## S3 method for class 'zerotrunc'
predict(object, newdata,
  type = c("response", "prob", "count", "zero"), na.action = na.pass, ...)
## S3 method for class 'zerotrunc'
residuals(object, type = c("deviance", "pearson", "response"), ...)
```

Arguments

`object`	an object of class `"zerotrunc"` as returned by `zerotrunc`.
`newdata`	optionally, a data frame in which to look for variables with which to predict. If omitted, the original observations are used.
`type`	character specifying the type of predictions or residuals, respectively. For details see below.
`na.action`	function determining what should be done with missing values in `newdata`. The default is to predict `NA`.
`...`	currently not used.

Details

A set of standard extractor functions for fitted model objects is available for objects of class `"zerotrunc"`, including methods to the generic functions `print` and `summary` which print the estimated coefficients along with some further information. The `summary` in particular supplies partial Wald tests based on the coefficients and the covariance matrix (estimated from the Hessian in the numerical optimization of the log-likelihood). As usual, the `summary` method returns an object of class `"summary.zerotrunc"` containing the relevant summary statistics which can subsequently be printed using the associated `print` method.

Both the `fitted` and `predict` methods can compute fitted responses. The latter additionally provides the predicted density (i.e., probabilities for the observed counts), the predicted mean from the count component (without zero truncation) and the predicted probability for observing a non-zero count (in the un-truncated model). The `residuals` method can compute raw residuals (observed - fitted), Pearson residuals (raw residuals scaled by square root of variance function), and deviance residuals (contributions to the centered log-likelihood).

A `logLik` method is provided, hence `AIC` can be called to compute information criteria.

See Also

`zerotrunc`

Examples

```
data("CrabSatellites", package = "countreg")
fm <- zerotrunc(satellites ~ width + color, data = CrabSatellites, subset = satellites > 0)

plot(residuals(fm, type = "deviance") ~ fitted(fm))
plot(residuals(fm, type = "pearson") ~ fitted(fm))

coef(fm)
summary(fm)
logLik(fm)
AIC(fm)
```

`TakeoverBids` *Takeover Bids Data*

Description

Firms that were targets of takeover bids during the period 1978–1985.

Usage

```
data("TakeoverBids")
```

Format

A data frame containing 126 observations on 9 variables.

bids Number of takeover bids (after the initial bid received by the target firm).

legalrest factor. Equals `"yes"` if target management responded by lawsuit.

realrest factor. Equals `"yes"` if target management proposed changes in asset structure.

finrest factor. Equals `"yes"` if target management proposed changes in ownership structure.

whiteknight factor. Equals `"yes"` if target management invited friendly third-party bid.

bidpremium Bid price divided by price 14 working days before bid.

insthold Percentage of stock held by institutions.

size Total book value of assets (in billions of USD).

regulation factor. Equals `"yes"` if intervention by federal regulators.

Details

The data were originally used by Jaggia and Thosar (1993), where further details on the variables may be found.

Source

Journal of Applied Econometrics Data Archive for Cameron and Johansson (1997).

http://qed.econ.queensu.ca/jae/1997-v12.3/cameron-johansson/

References

Cameron AC, Johansson P (1997). "Count Data Regression Using Series Expansion: With Applications", *Journal of Applied Econometrics*, **12**(3), 203–224.

Cameron AC, Trivedi PK (2013). *Regression Analysis of Count Data*, 2nd ed. Cambridge: Cambridge University Press.

Jaggia S, Thosar S (1993). "Multiple Bids as a Consequence of Target Management Resistance: A Count Data Approach", *Review of Quantitative Finance and Accounting*, **3**, 447–457.

Examples

```
data("TakeoverBids", package = "countreg")

## Poisson model:
## Jaggia and Thosar (1993), Table 3
## Cameron and Johansson (1997), Table IV
tb_p <- glm(bids ~ . + I(size^2), data = TakeoverBids, family = poisson)
summary(tb_p)
logLik(tb_p)

## dispersion tests
## Cameron and Trivedi (2013, p. 185)
AER::dispersiontest(tb_p, alternative = "less", trafo = 2)
AER::dispersiontest(tb_p, alternative = "less", trafo = 1)

## visualization of underdispersion
if(require("topmodels")) {
rootogram(tb_p)
qqrplot(tb_p, range = c(0.05, 0.95))
}

## Parts of Cameron and Trivedi (2013), Table 5.4
summary(residuals(tb_p, type = "response"))
summary(residuals(tb_p, type = "pearson"))
summary(residuals(tb_p, type = "deviance"))

## hurdle Poisson model mitigates underdispersion
tb_hp <- hurdle(bids ~ . + I(size^2), data = TakeoverBids, dist = "poisson")
AIC(tb_p, tb_hp)
if(require("topmodels")) {
rootogram(tb_hp)
qqrplot(tb_hp, range = c(0.05, 0.95))
}
```

`zeroinfl` *Zero-inflated Count Data Regression*

Description

Fit zero-inflated regression models for count data via maximum likelihood.

Usage

```
zeroinfl(formula, data, subset, na.action, weights, offset,
  dist = c("poisson", "negbin", "geometric", "binomial"),
  link = c("logit", "probit", "cloglog", "cauchit", "log"),
  size = NULL, control = zeroinfl.control(...),
  model = TRUE, y = TRUE, x = FALSE, ...)
```

Arguments

`formula`	symbolic description of the model, see details.
`data`, `subset`, `na.action`	arguments controlling formula processing via `model.frame`.
`weights`	optional numeric vector of weights.
`offset`	optional numeric vector with an a priori known component to be included in the linear predictor of the count model. See below for more information on offsets.
`dist`	character specification of count model family (a log link is always used).
`link`	character specification of link function in the binary zero-inflation model (a binomial family is always used).
`size`	size parameter in case the a binomial count model is used (`dist = "binomial"`). By default the maximum count is used.
`control`	a list of control arguments specified via `zeroinfl.control`.
`model`, `y`, `x`	logicals. If `TRUE` the corresponding components of the fit (model frame, response, model matrix) are returned.
`...`	arguments passed to `zeroinfl.control` in the default setup.

Details

Zero-inflated count models are two-component mixture models combining a point mass at zero with a proper count distribution. Thus, there are two sources of zeros: zeros may come from both the point mass and from the count component. Usually the count model is a Poisson or negative binomial regression (with log link). The geometric distribution is a special case of the negative binomial with size parameter equal to 1. For modeling the unobserved state (zero vs. count), a binary model is used that captures the probability of zero inflation. in the simplest case only with an intercept but potentially containing regressors. For this zero-inflation model, a binomial model with different links can be used, typically logit or probit.

The `formula` can be used to specify both components of the model: If a `formula` of type `y ~ x1 + x2` is supplied, then the same regressors are employed in both components. This is equivalent to `y ~ x1 + x2 | x1 + x2`. Of course, a different set of regressors could be specified for the count and zero-inflation component, e.g., `y ~ x1 + x2 | z1 + z2 + z3` giving the count data model `y ~ x1 + x2` conditional on (`|`) the zero-inflation model `y ~ z1 + z2 + z3`. A simple inflation model where all zero counts have the same probability of belonging to the zero component can by specified by the formula `y ~ x1 + x2 | 1`.

Offsets can be specified in both components of the model pertaining to count and zero-inflation model: `y ~ x1 + offset(x2) | z1 + z2 + offset(z3)`, where `x2` is used as an offset (i.e., with coefficient fixed to 1) in the count component and `z3` analogously in the zero-inflation component. By the rule stated above `y ~ x1 + offset(x2)` is expanded to `y ~ x1 + offset(x2) | x1 + offset(x2)`. Instead of using the `offset()` wrapper within the `formula`, the `offset` argument can also be employed which sets an offset only for the count model. Thus, `formula = y ~ x1` and `offset = x2` is equivalent to `formula = y ~ x1 + offset(x2) | x1`.

All parameters are estimated by maximum likelihood using `optim`, with control options set in `zeroinfl.control`. Starting values can be supplied, estimated by the EM (expectation maximization) algorithm, or by `glm.fit` (the default). Standard errors are derived numerically using the Hessian matrix returned by `optim`. See `zeroinfl.control` for details.

The returned fitted model object is of class "zeroinfl" and is similar to fitted "glm" objects. For elements such as "coefficients" or "terms" a list is returned with elements for the zero and count component, respectively. For details see below.

A set of standard extractor functions for fitted model objects is available for objects of class "zeroinfl", including methods to the generic functions print, summary, coef, vcov, logLik, residuals, predict, fitted, terms, model.matrix. See predict.zeroinfl for more details on all methods.

Value

An object of class "zeroinfl", i.e., a list with components including

coefficients	a list with elements "count" and "zero" containing the coefficients from the respective models,
residuals	a vector of raw residuals (observed - fitted),
fitted.values	a vector of fitted means,
optim	a list with the output from the optim call for minimizing the negative log-likelihood,
control	the control arguments passed to the optim call,
start	the starting values for the parameters passed to the optim call,
weights	the case weights used,
offset	a list with elements "count" and "zero" containing the offset vectors (if any) from the respective models,
n	number of observations (with weights > 0),
df.null	residual degrees of freedom for the null model (= n - 2),
df.residual	residual degrees of freedom for fitted model,
terms	a list with elements "count", "zero" and "full" containing the terms objects for the respective models,
theta	estimate of the additional θ parameter of the negative binomial model (if a negative binomial regression is used),
SE.logtheta	standard error for $\log(\theta)$,
loglik	log-likelihood of the fitted model,
vcov	covariance matrix of all coefficients in the model (derived from the Hessian of the optim output),
dist	character string describing the count distribution used,
link	character string describing the link of the zero-inflation model,
linkinv	the inverse link function corresponding to link,
converged	logical indicating successful convergence of optim,
call	the original function call,
formula	the original formula,
levels	levels of the categorical regressors,

`contrasts`	a list with elements `"count"` and `"zero"` containing the contrasts corresponding to `levels` from the respective models,
`model`	the full model frame (if `model = TRUE`),
`y`	the response count vector (if `y = TRUE`),
`x`	a list with elements `"count"` and `"zero"` containing the model matrices from the respective models (if `x = TRUE`),

References

Cameron AC, Trivedi PK (2013). *Regression Analysis of Count Data*, 2nd ed. New York: Cambridge University Press.

Cameron AC, Trivedi PK (2005). *Microeconometrics: Methods and Applications*. Cambridge: Cambridge University Press.

Lambert D (1992). "Zero-Inflated Poisson Regression, with an Application to Defects in Manufacturing". *Technometrics*. **34**(1), 1–14.

Zeileis A, Kleiber C, Jackman S (2008). "Regression Models for Count Data in R." *Journal of Statistical Software*, **27**(8), 1–25. doi:10.18637/jss.v027.i08.

See Also

`zeroinfl.control`, `glm`, `glm.fit`, `glm.nb`, `hurdle`

Examples

```
## data
data("CrabSatellites", package = "countreg")
cs <- CrabSatellites[, c("satellites", "width", "color")]
cs$color <- as.numeric(cs$color)

## without inflation
## ("satellites ~ ." is "satellites ~ width + color")
fm_pois <- glm(satellites ~ ., data = cs, family = poisson)
fm_qpois <- glm(satellites ~ ., data = cs, family = quasipoisson)
fm_nb <- glm.nb(satellites ~ ., data = cs)

## with simple inflation (no regressors for zero component)
fm_zip <- zeroinfl(satellites ~ . | 1, data = cs)
fm_zinb <- zeroinfl(satellites ~ . | 1, data = cs, dist = "negbin")

## inflation with regressors
## ("satellites ~ . | ." is "satellites ~ width + color | width + color")
fm_zip2 <- zeroinfl(satellites ~ . | ., data = cs)
fm_zinb2 <- zeroinfl(satellites ~ . | ., data = cs, dist = "negbin")
```

`zeroinfl.control` *Control Parameters for Zero-inflated Count Data Regression*

Description

Various parameters that control fitting of zero-inflated regression models using `zeroinfl`.

Usage

```
zeroinfl.control(method = "BFGS", maxit = 10000, trace = FALSE,
  EM = FALSE, start = NULL, hessian = TRUE, ...)
```

Arguments

`method`	characters string specifying the `method` argument passed to `optim`.
`maxit`	integer specifying the `maxit` argument (maximal number of iterations) passed to `optim`.
`trace`	logical or integer controlling whether tracing information on the progress of the optimization should be produced (passed to `optim`).
`EM`	logical. Should starting values be estimated by the EM (expectation maximization) algorithm? See details.
`start`	an optional list with elements `"count"` and `"zero"` (and potentially `"theta"`) containing the coefficients for the corresponding component.
`hessian`	logical. Should the Hessian be computed to derive an estimate of the variance-covariance matrix? If `FALSE`, the variance-covariance matrix contains only `NA`s.
`...`	arguments passed to `optim`.

Details

All parameters in `zeroinfl` are estimated by maximum likelihood using `optim` with control options set in `zeroinfl.control`. Most arguments are passed on directly to `optim`, only `trace` is also used within `zeroinfl` and `EM`/`start` control the choice of starting values for calling `optim`.

Starting values can be supplied, estimated by the EM (expectation maximization) algorithm, or by `glm.fit` (the default). Standard errors are derived numerically using the Hessian matrix returned by `optim`. To supply starting values, `start` should be a list with elements `"count"` and `"zero"` and potentially `"theta"` (for negative binomial components only) containing the starting values for the coefficients of the corresponding component of the model.

Value

A list with the arguments specified.

See Also

`zeroinfl`

Examples

```
data("CrabSatellites", package = "countreg")

## default start values
fm1 <- zeroinfl(satellites ~ width + as.numeric(color), data = CrabSatellites)

## use EM algorithm for start values
fm2 <- zeroinfl(satellites ~ width + as.numeric(color), data = CrabSatellites, EM = TRUE)

## user-supplied start values
fm3 <- zeroinfl(satellites ~ width + as.numeric(color), data = CrabSatellites,
  start = list(count = c(0.5, 0, 0), zero = c(10, -0.5, 0.5)))
```

---

zerotrunc *Zero-Truncated Count Data Regression*

Description

Fit zero-truncated regression models for count data via maximum likelihood.

Usage

```
zerotrunc(formula, data, subset, na.action, weights, offset,
  dist = c("poisson", "negbin", "geometric"), theta = Inf,
  control = zerotrunc.control(...),
  model = TRUE, y = TRUE, x = FALSE, ...)
```

Arguments

`formula`	symbolic description of the model.
`data`, `subset`, `na.action`	arguments controlling formula processing via `model.frame`.
`weights`	optional numeric vector of weights.
`offset`	optional numeric vector with an a priori known component to be included in the linear predictor.
`dist`	character specification of the count distribution family.
`theta`	numeric. Alternative (and more flexible) specification of the count distribution family. Some values correspond to `dist` values: `theta = Inf` (`"poisson"`), `theta = 1` (`"geometric"`), `theta = NULL` (`"negbin"`). But every non-negative value for `theta` is allowed. When `theta` is given, `dist` must not be specified and vice versa.
`control`	a list of control arguments specified via `zerotrunc.control`.
`model`, `y`, `x`	logicals. If `TRUE` the corresponding components of the fit (model frame, response, model matrix) are returned.
`...`	arguments passed to `zerotrunc.control` in the default setup.

Details

All zero-truncated count data models in `zerotrunc` are obtained from the corresponding untruncated distribution using a log-link between the mean of the untruncated distribution and the linear predictor. All parameters are estimated by maximum likelihood using `optim`, with control options set in `zerotrunc.control`. Starting values can be supplied, otherwise they are estimated by `glm.fit` (the default). Standard errors are derived numerically using the Hessian matrix returned by `optim`. See `zerotrunc.control` for details.

The returned fitted model object is of class `"zerotrunc"` and is similar to fitted `"glm"` objects.

A set of standard extractor functions for fitted model objects is available for objects of class `"zerotrunc"`, including methods to the generic functions `print`, `summary`, `coef`, `vcov`, `logLik`, `residuals`, `predict`, `fitted`, `terms`, `model.frame`, `model.matrix`. See `predict.zerotrunc` for more details on all methods.

Value

An object of class `"zerotrunc"`, i.e., a list with components including

`coefficients`	estimated coefficients,
`residuals`	a vector of raw residuals (observed - fitted),
`fitted.values`	a vector of fitted means,
`optim`	a list with the output from the `optim` call for minimizing the negative log-likelihood,
`control`	the control arguments passed to the `optim` call,
`start`	the starting values for the parameters passed to the `optim` call(s),
`weights`	the case weights used (if any),
`offset`	the offset vector used (if any),
`n`	number of observations,
`df.null`	residual degrees of freedom for the null model,
`df.residual`	residual degrees of freedom for fitted model,
`terms`	terms objects for the model,
`theta`	(estimated) θ parameter of the negative binomial model,
`SE.logtheta`	standard error for $\log(\theta)$,
`loglik`	log-likelihood of the fitted model,
`vcov`	covariance matrix of the coefficients in the model (derived from the Hessian of the `optim` output),
`dist`	character describing the distribution used,
`converged`	logical indicating successful convergence of `optim`,
`call`	the original function call,
`formula`	the original formula,
`levels`	levels of the categorical regressors,
`contrasts`	contrasts corresponding to `levels` from the model,
`model`	the model frame (if `model = TRUE`),
`y`	the response count vector (if `y = TRUE`),
`x`	model matrix (if `x = TRUE`).

References

Cameron AC, Trivedi PK (2013). *Regression Analysis of Count Data*, 2nd ed. New York: Cambridge University Press.

Zeileis A, Kleiber C, Jackman S (2008). "Regression Models for Count Data in R." *Journal of Statistical Software*, **27**(8), 1–25. doi:10.18637/jss.v027.i08.

See Also

`zerotrunc.control`, `glm`, `glm.fit`, `glm.nb`, `zeroinfl`, `hurdle`

Examples

```
## data
data("CrabSatellites", package = "countreg")
cs <- CrabSatellites[, c("satellites", "width", "color")]
cs$color <- as.numeric(cs$color)
cs <- subset(cs, subset = satellites > 0)

## poisson
zt_p <- zerotrunc(satellites ~ ., data = cs)
## or equivalently
zt_p <- zerotrunc(satellites ~ ., data = cs, theta = Inf)
summary(zt_p)

## negbin
zt_nb <- zerotrunc(satellites ~ ., data = cs, dist = "negbin")
## or equivalently
zt_nb <- zerotrunc(satellites ~ ., data = cs, theta = NULL)
summary(zt_nb)
```

`zerotrunc.control` *Control Parameters for Zero-Truncated Count Data Regression*

Description

Various parameters that control fitting of zero-truncated count regression models using `zerotrunc`.

Usage

```
zerotrunc.control(method = "BFGS", maxit = 10000, start = NULL, ...)
```

Arguments

`method`	characters string specifying the `method` argument passed to `optim`.
`maxit`	integer specifying the `maxit` argument (maximal number of iterations) passed to `optim`.
`start`	an optional vector of starting values, see details.
`...`	arguments passed to `optim`.

Details

All parameters in zerotrunc are estimated by maximum likelihood using optim with control options set in zerotrunc.control. Most arguments are passed on directly to optim, only start is used to control how optim is called.

Starting values can be supplied via start or estimated by glm.fit (default). Standard errors are derived numerically using the Hessian matrix returned by optim. To supply starting values, start should be a vector with (at least) starting values for the regression coefficients. In case a negative binomial distribution with unknown theta is used, a starting value for theta may be supplied by adding an additional vector element (e.g., start = c(coef, theta)); by default theta = 1 is used as the starting value otherwise.

Value

A list with the arguments specified.

See Also

zerotrunc

Examples

```
data("CrabSatellites", package = "countreg")

## default start values
zt_nb <- zerotrunc(satellites ~ width + as.numeric(color), data = CrabSatellites,
  subset = satellites > 0, dist = "negbin")

## user-supplied start values and other options
zt_nb2 <- zerotrunc(satellites ~ width + as.numeric(color), data = CrabSatellites,
  subset = satellites > 0, dist = "negbin", start = c(0.5, 0, 0))
```

---

zinbinom-extensions *Extension of the Zero-Inflated Negative Binomial Distribution*

Description

Score function for the zero-inflated negative binomial distribution with parameters mu (= mean of the uninflated distribution), dispersion parameter theta (or equivalently size), and inflation probability pi (for structural zeros).

Usage

```
szinbinom(x, mu, theta, size, pi, parameter = c("mu", "theta", "pi"), drop = TRUE)
```

Arguments

`x`	vector of (non-negative integer) quantiles.
`mu`	vector of non-negative means of the uninflated negative binomial distribution.
`theta, size`	vector of strictly positive dispersion parameters (shape parameter of the gamma mixing distribution). Only one of `theta` or `size` must be specified.
`pi`	vector of zero inflation probabilities for structural zeros.
`parameter`	character. Should the derivative with respect to `"mu"` and/or `"theta"` and/or `"pi"` be computed?
`drop`	logical. Should the result be a matrix (`drop = FALSE`) or should the dimension be dropped (`drop = TRUE`, the default)?

Details

The uninflated negative binomial distribution has density

$$f(x) = \frac{\Gamma(x+\theta)}{\Gamma(\theta)x!} \cdot \frac{\mu^y \theta^\theta}{(\mu+\theta)^{y+\theta}}$$

for $x = 0, 1, 2, \ldots$. The zero-inflated density is then simply obtained as

$$g(x) = \pi \cdot I_{\{0\}}(x) + (1-\pi) \cdot f(x)$$

where I is the indicator function (for the point mass at zero).

Value

`szinbinom` gives the score function (= derivative of the log-density with respect to mu and/or theta and/or pi).

See Also

`dzinbinom`, `dnbinom`, `zeroinfl`

`zipois-extensions` *Extension of the Zero-Inflated Poisson Distribution*

Description

Score function for the zero-inflated Poisson distribution with parameters `lambda` (= mean of the uninflated distribution) and inflation probability `pi` (for structural zeros).

Usage

```
szipois(x, lambda, pi, parameter = c("lambda", "pi"), drop = TRUE)
```

Arguments

`x`	vector of (non-negative integer) quantiles.
`lambda`	vector of non-negative means of the uninflated Poisson distribution.
`pi`	vector of zero inflation probabilities for structural zeros.
`parameter`	character. Should the derivative with respect to `"mu"` and/or `"size"` be computed?
`drop`	logical. Should the result be a matrix (`drop = FALSE`) or should the dimension be dropped (`drop = TRUE`, the default)?

Details

The uninflated Poisson distribution has density

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

for $x = 0, 1, 2, \ldots$. The zero-inflated density is then simply obtained as

$$g(x) = \pi \cdot I_{\{0\}}(x) + (1 - \pi) \cdot f(x)$$

where I is the indicator function (for the point mass at zero).

Value

`szipois` gives the score function (= derivative of the log-density with respect to lambda and/or pi).

See Also

`dzipois`, `dpois`, `zeroinfl`

`zitest` *Zero Inflation Tests*

Description

Tests the null hypothesis of a Poisson GLM against the alternative of a zero-inflated version.

Usage

```
zitest(object, type = c("scoreZIP"))
```

Arguments

`object`	a fitted Poisson GLM of class `"glm"` as fitted by `glm` with family `poisson`.
`type`	type of test, currently only `scoreZIP`. See details.

Details

Currently alternative contains only intercept term in binary part, as in van den Broek (1995).

Note that under the null hypothesis the parameter is on the boundary of the parameter space, hence the p-value is non-standard.

Value

An object of class `"htest"`.

References

van den Broek J (1995). "A Score Test for Zero Inflation in a Poisson Distribution". *Biometrics*, **51**, 738–743.

See Also

`glm`, `poisson`, `glm.nb`

Examples

```
data("CrabSatellites", package = "countreg")
CrabSatellites <- transform(CrabSatellites,
  color = as.numeric(color),
  spine = as.numeric(spine),
  cwidth = cut(width, c(-Inf, seq(23.25, 29.25), Inf))
)

cs_p <- glm(satellites ~ width + color, data = CrabSatellites, family = poisson)
zitest(cs_p)
```

`ztnbinom-extensions` *Extension of the Zero-Truncated Negative Binomial Distribution*

Description

Score function, hessian, mean, and variance for the zero-truncated negative binomial distribution with parameters `mu` (= mean of the untruncated distribution) and dispersion parameter `theta` (or equivalently `size`).

Usage

```
sztnbinom(x, mu, theta, size, parameter = c("mu", "theta", "size"), drop = TRUE)
hztnbinom(x, mu, theta, size, parameter = c("mu", "theta"), drop = TRUE)
mean_ztnbinom(mu, theta, size, drop = TRUE)
var_ztnbinom(mu, theta, size, drop = TRUE)
```

Arguments

`x`	vector of (positive integer) quantiles.
`mu`	vector of non-negative means of the untruncated negative binomial distribution.
`theta, size`	vector of strictly positive dispersion parameters (shape parameter of the gamma mixing distribution). Only one of `theta` or `size` must be specified.
`parameter`	character. Should the derivative with respect to `"mu"` and/or `"theta"/"size"` be computed?
`drop`	logical. Should the result be a matrix (`drop = FALSE`) or should the dimension be dropped (`drop = TRUE`, the default)?

Details

The untruncated negative binomial distribution has density

$$f(x) = \frac{\Gamma(x+\theta)}{\Gamma(\theta)x!} \cdot \frac{\mu^y \theta^\theta}{(\mu+\theta)^{y+\theta}}$$

for $x = 0, 1, 2, \ldots$. The zero-truncated density is then simply obtained as

$$g(x) = \frac{f(x)}{1 - f(0)}$$

for $x = 1, 2, \ldots$.

Value

`sztnbinom` gives the score function (= derivative of the log-density with respect to mu and/or theta). `hztnbinom` gives the hessian (= 2nd derivative of the log-density with respect to mu and/or theta). `mean_ztnbinom` and `var_ztnbinom` give the mean and the variance, respectively.

See Also

`dztnbinom`, `dnbinom`, `zerotrunc`

`ztpois-extensions` *Extension of the Zero-Truncated Poisson Distribution*

Description

Score function, hessian, mean, and variance for the zero-truncated Poisson distribution with parameter `lambda` (= mean of the untruncated distribution) or `mean` (= of the truncated distribution).

Usage

```
sztpois(x, lambda, mean, parameter = "lambda", drop = TRUE)
hztpois(x, lambda, mean, parameter = "lambda", drop = TRUE)
mean_ztpois(lambda, mean, drop = TRUE)
var_ztpois(lambda, mean, drop = TRUE)
```

Arguments

`x`	vector of (positive integer) quantiles.
`lambda`	vector of (non-negative) means of the untruncated Poisson distribution. Only one of `lambda` or `mean` should be specified.
`mean`	vector of means (greater than 1) of the zero-truncated Poisson distribution. Only one of `lambda` or `mean` should be specified.
`parameter`	character. Should the derivative with respect to `"lambda"` or `"mean"` be computed?
`drop`	logical. Should the result be a matrix (`drop = FALSE`) or should the dimension be dropped (`drop = TRUE`, the default)?

Details

The untruncated Poisson distribution has density

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

for $x = 0, 1, 2, \ldots$. The zero-truncated density is then simply obtained as

$$g(x) = \frac{f(x)}{1 - f(0)}$$

for $x = 1, 2, \ldots$.

The zero-truncated distribution has expectation $E(X) = \mu = \lambda/(1 - \exp(-\lambda))$ and variance $Var(X) = \mu \cdot (\lambda + 1 - \mu)$, where λ is the expectation of the untruncated Poisson distribution.

Despite the simple form of the transformation $\mu(\lambda)$ the inverse $\lambda(\mu)$ has no closed-form solution and is computed numerically if needed.

Value

`sztpois` gives the score function (= derivative of the log-density with respect to lambda or mean). `hztpois` gives the hessian (= 2nd derivative of the log-density with respect to lambda or mean). `mean_ztpois` and `var_ztpois` give the mean and the variance, respectively.

See Also

`dztpois`, `ztpoisson`, `dpois`, `zerotrunc`

ztpoisson *Family Object for the Zero-Truncated Poisson Distribution*

Description

Family object for specification of zero-truncated Poisson models as a glm.

Usage

```
ztpoisson()
```

Details

The ztpoisson family allows to estimate zero-truncated Poisson regression models as generalized linear models. As in the zerotrunc function, the link function is a log-link between the mean λ of the untruncated Poisson distribution and the linear predictor. This corresponds to a non-canonical link between for the mean of the zero-truncated Poisson distribution which does not have a closed-form representation.

Note that for new family objects 'glm()' estimates a dispersion parameter by default. Thus, unlike for the poisson family the dispersion parameter is not fixed, unless dispersion = 1 is set explicitly .

Value

An object of class "family".

See Also

dztpois, poisson, zerotrunc

Examples

```
## data
data("CrabSatellites", package = "countreg")
cs <- subset(CrabSatellites, subset = satellites > 0)
cs$color <- as.numeric(cs$color)

## model
ztp1 <- glm(satellites ~ width + color, data = cs, family = ztpoisson)
ztp2 <- zerotrunc(satellites ~ width + color, data = cs)
summary(ztp1, dispersion = 1) ## to get fixed dispersion as for poisson
summary(ztp2)
```

Index