Regression Models for Count Data in R

Zeileis, Achim; Kleiber, Christian; Jackman, Simon

Published: 01/01/2007

Document Version
Publisher's PDF, also known as Version of record

Regression Models for Count Data in R

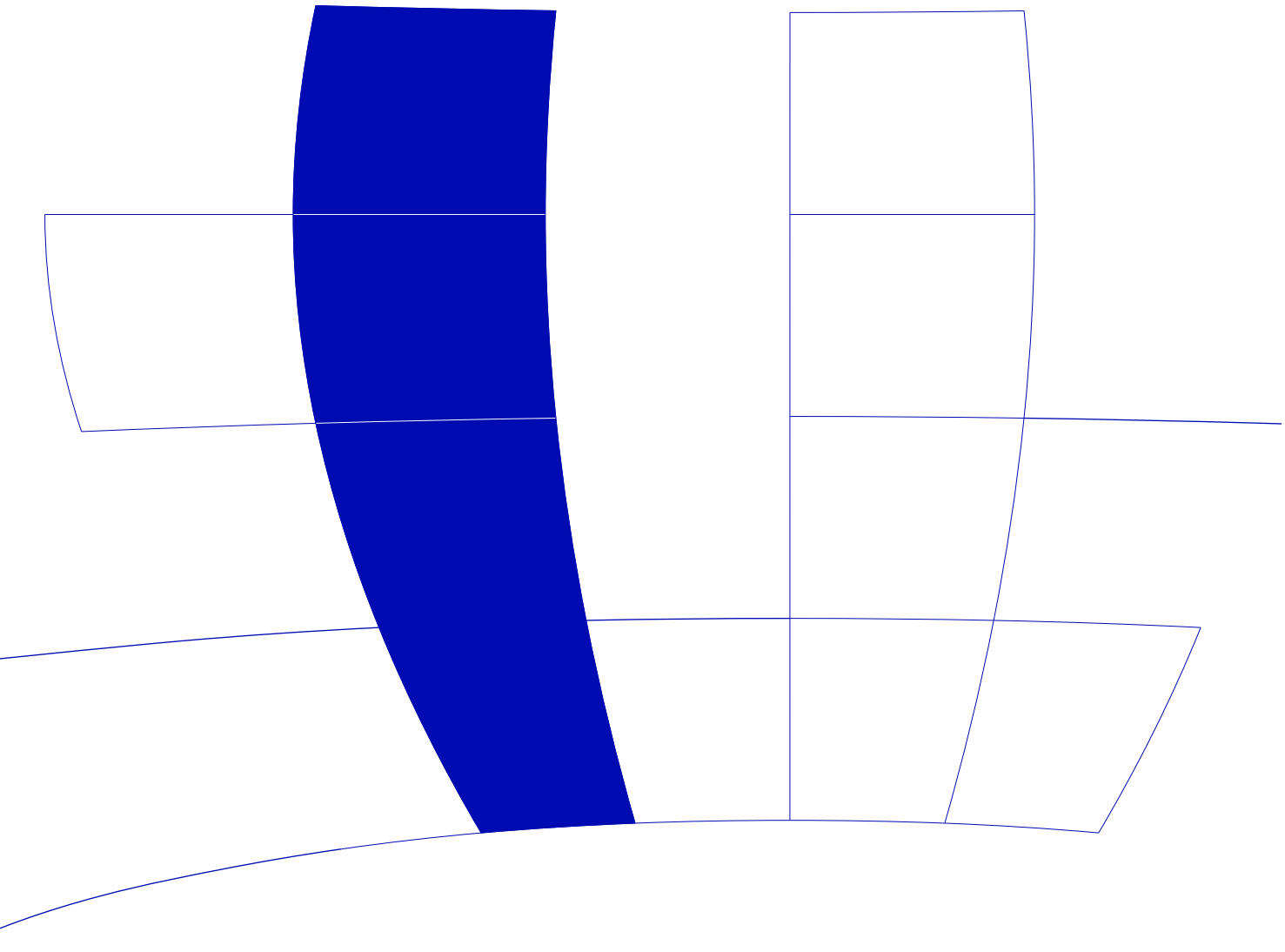
Achim Zeileis, Christian Kleiber, Simon Jackman

Department of Statistics and Mathematics
Wirtschaftsuniversität Wien

Research Report Series

Report 53
April 2007

http://statmath.wu-wien.ac.at/



Regression Models for Count Data in R

Achim Zeileis
Wirtschaftsuniversität Wien

Christian Kleiber
Universität Basel

Simon Jackman
Stanford University

Abstract

The classical Poisson, geometric and negative binomial regression models for count data belong to the family of generalized linear models and are available at the core of the statistics toolbox in the R system for statistical computing. After reviewing the conceptual and computational features of these methods, a new implementation of zero-inflated and hurdle regression models in the functions `zeroinfl()` and `hurdle()` from the package **pscl** is introduced. It reuses design and functionality of the basic R functions just as the underlying conceptual tools extend the classical models. Both model classes are able to incorporate over-dispersion and excess zeros—two problems that typically occur in count data sets in economics and the social and political sciences—better than their classical counterparts. Using cross-section data on the demand for medical care, it is illustrated how the classical as well as the zero-augmented models can be fitted, inspected and tested in practice.

Keywords: GLM, Poisson model, negative binomial model, zero-inflated model, hurdle model.

1. Introduction

Modeling count variables is a common task in microeconometrics, the social and political sciences. The classical Poisson regression model for count data is often of limited use in these disciplines because empirical count data sets typically exhibit over-dispersion and/or an excess number of zeros. The former issue can be addressed by extending the plain Poisson regression model in various directions: e.g., using sandwich covariances or estimating an additional dispersion parameter (in a so-called quasi-Poisson model). Another more formal way is to use a negative binomial regression. All of these models belong to the family of generalized linear models (GLMs, see Nelder and Wedderburn 1972; McCullagh and Nelder 1989). However, although these models typically can capture over-dispersion rather well, they are in many applications not sufficient for modeling excess zeros. Since Lambert (1992) there is increased interest, both in the statistics and econometrics literature, in models that address this issue by adding a second component responsible for the zeros to the count regression: Zero-inflation models are mixture models that combine a count component and a point mass at zero. Hurdle models (Mullahy 1986) take a somewhat different approach and combine a left-truncated count component with a right-censored hurdle component. An overview of count data models in econometrics, including zero-inflated and hurdle models is provided in Cameron and Trivedi (1998, 2005).

In R (R Development Core Team 2007), the GLMs are provided by the model fitting functions `glm()` (Chambers and Hastie 1992) in the **stats** package and `glm.nb()` in the **MASS** package (Venables and Ripley 2002) along with associated methods for diagnostics and inference. Here, we discuss the implementation of zero-inflated and hurdle models in the functions `zeroinfl()` and `hurdle()` in the **pscl** package (Jackman 2007). The design of both modeling functions as well as the methods operating on the associated fitted model objects follows that of the base R functionality so that the new software integrates easily into the computational toolbox for modeling count data in R.

The remainder of this paper is organized as follows: Section 2 discusses both the classic and zero-augmented count data models and their R implementations. In Section 3, all count regression

models discussed are applied to a microeconometric cross-section data set on the demand for medical care. The summary in Section 4 concludes the main part of the paper; further technical details are presented in the appendix.

2. Models and software

In this section, we briefly outline the theory and its implementation in R (R Development Core Team 2007) for some basic count data regression models as well as their zero-augmented extensions. The classic Poisson, geometric and negative binomial models are described in a generalized linear model (GLM) framework implemented in R by the `glm()` function (Chambers and Hastie 1992) in the **stats** package and the `glm.nb()` function in the **MASS** package (Venables and Ripley 2002). The zero-inflated and hurdle extensions of these models are provided by the functions `zeroinfl()` and `hurdle()` in package **pscl** (Jackman 2007). The original implementation of Jackman (2007) was improved by Kleiber and Zeileis (2008) for **pscl** to make the fitting functions and the fitted model objects more similar to their `glm()` and `glm.nb()` counterparts. The most important features of the new `zeroinfl()` and `hurdle()` functions are discussed below while some technical aspects are deferred to the appendix. An alternative implementation of zero-inflated count models is available in function `zicounts()` from package **zicounts** (Mwalili 2006). However, the interface of `zicounts()` (both in terms of the fitting function and the fitted model objects) is less standard. Therefore, it is less intuitive and re-using generic inference tools is more cumbersome and hence this package is not discussed here.

Additionally to zero-augmented models, there are many further extensions to the classical Poisson model which are not discussed here. Some important model classes include mixed-effects models—available in R in packages **lme4** and **nlme** (see Pinheiro and Bates 2000)—and finite mixture models—implemented in R in package **flexmix** (Leisch 2004)—or generalized estimating equations (GEE)—provided in R by package **geepack** (Halekoh, Højsgaard, and Yan 2006). Further information about the models and alternative R implementations can be found in the respective references.

2.1. Generalized linear models

Model frame

The basic count data regression models can be represented and understood using the GLM framework that emerged in the statistical literature in the early 1970s (Nelder and Wedderburn 1972). In the following, we briefly sketch some important aspects relating to the unifying conceptual properties and their implementation in R—for a detailed theoretical account of GLMs see McCullagh and Nelder (1989).

GLMs describe the dependence of a variable y_i $(i = 1, \ldots, n)$ on a set of regressors x_i. The conditional distribution of $y_i|x_i$ is a linear exponential family with probability density function

$$f(y; \lambda, \phi) \quad = \quad \exp\left(\frac{y \cdot \lambda - b(\lambda)}{\phi} + c(y, \phi)\right), \tag{1}$$

where λ is the canonical parameter that depends on the regressors via a linear predictor and ϕ is a dispersion parameter that is often known. The functions $b(\cdot)$ and $c(\cdot)$ are known and determine which member of the family is used, e.g., the normal, binomial or Poisson distribution. Conditional mean and variance of y_i are given by $\mathsf{E}[y_i \,|\, x_i] = \mu_i = b'(\lambda_i)$ and $\mathsf{VAR}[y_i \,|\, x_i] = \phi \cdot b''(\lambda_i)$. Thus, up to a scale or dispersion parameter ϕ, the distribution of y_i is determined by its mean .Its variance is proportional to $V(\mu) = b''(\lambda(\mu))$, also called variance function.

The dependence of the conditional mean $\mathsf{E}[y_i \,|\, x_i] = \mu_i$ on the regressors x_i is specified via

$$g(\mu_i) \quad = \quad x_i^\top \beta, \tag{2}$$

where $g(\cdot)$ is a known link function and β is the vector of regression coefficients which are typically estimated by maximum likelihood (ML) using the iterative weighted least squares (IWLS) algorithm.

Instead of viewing GLMs as models for the full likelihood (as determined by Equation 1), they can also be regarded as regression models for the mean only (as specified in Equation 2) where the estimating functions used for fitting the model are derived from a particular family. As illustrated in the remainder of this section, the estimating function point of view is particularly useful for relaxing the assumptions imposed by the Poisson likelihood.

R provides a very flexible implementation of the general GLM framework in the function `glm()` (Chambers and Hastie 1992) contained in the **stats** package. Its most important arguments are

```
glm(formula, data, subset, na.action, weights, offset,
  family = gaussian, start = NULL, control = glm.control(...),
  model = TRUE, y = TRUE, x = FALSE, ...)
```

where `formula` plus `data` is the now standard way of specifying regression relationships in R/S introduced in Chambers and Hastie (1992). The remaining arguments in the first line (`subset`, `na.action`, `weights`, and `offset`) are also standard for setting up formula-based regression models in R/S. The arguments in the second line control aspects specific to GLMs while the arguments in the last line specify which components are returned in the fitted model object (of class "`glm`" which inherits from "`lm`"). By default the model frame (`model`) and the vector $(y_1, \ldots, y_n)^\top$ (`y`) but not the model matrix (`x` containing $x_1, \ldots, x_n$ combined row-wise) are included. The `family` argument specifies the link $g(\mu)$ and variance function $V(\mu)$ of the model, `start` can be used to set starting values for β[1] and `control` contains control parameters for the IWLS algorithm. The high-level `glm()` interface relies on the function `glm.fit()` which carries out the actual model fitting (without taking a formula-based input or returning classed output).

For "`glm`" objects, a set of standard methods (including `print()`, `predict()`, `logLik()` and many others) are provided. Inference can easily be performed using the `summary()` method for assessing the regression coefficients via partial Wald tests or the `anova()` method for comparing nested models via analysis of deviance. These inference functions are complemented by further generic inference functions in contributed packages: e.g., **lmtest** (Zeileis and Hothorn 2002) provides a `coeftest()` function that also computes partial Wald tests but allows for specification of alternative (robust) standard errors. Similarly, `waldtest()` from **lmtest** and `linear.hypothesis()` from **car** (Fox 2002) assess nested models via Wald tests (using different specifications for the nested models). Finally, `lrtest()` from **lmtest** compares nested models via likelihood ratio (LR) tests based on an interface similar to `waldtest()` and `anova()`.

Poisson model

The simplest distribution used for modeling count data is the Poisson distribution with probability density function

$$f(y;\mu) \quad = \quad \frac{\exp(-\mu) \cdot \mu^y}{y!}, \tag{3}$$

which is of type (1) and thus Poisson regression is a special case of the GLM framework. The canonical link is $g(\mu) = \log(\mu)$ resulting in a log-linear relationship between mean and linear predictor. The variance in the Poisson model is identical to the mean, thus the dispersion is fixed to $\phi = 1$ and the variance function is $V(\mu) = \mu$.

In R, this can easily be specified in the `glm()` call just by setting `family = poisson` (where the default log link could also be changed in the `poisson()` call).

In practice, the Poisson model is often useful for describing the mean μ_i but underestimates the variance in the data, rendering all model-based tests liberal. One way of dealing with this is to use

[1] Alternatively, the algorithm can be initialized in terms of the linear predictor $x_i^\top \beta$ or the mean μ_i.

the same estimating functions for the mean, but to base the inference on the more robust sandwich covariance matrix estimator. In R, this estimator is provided by the `sandwich()` function in the **sandwich** package (Zeileis 2004, 2006).

Quasi-Poisson model

Another way of dealing with over-dispersion is to use the mean regression function and the variance function from the Poisson GLM but to leave the dispersion parameter ϕ unrestricted. Thus, ϕ is not assumed to be fixed to 1 but is estimated from the data. This strategy leads to the same coefficient estimates as the standard Poisson model but inference is adjusted for over-dispersion. Consequently, both models (quasi-Poisson and sandwich-adjusted Poisson) adopt the estimating function view of the Poisson model and do *not* correspond to models with fully specified likelihoods.

In R, the quasi-Poisson model with estimated dispersion parameter can also be fitted with the `glm()` function, simply setting `family = quasipoisson`.

Negative binomial models

A third way of modeling over-dispersed count data is to assume a negative binomial distribution for $y_i|x_i$ which can arise as a mixture of Poisson distributions. One parametrization of its probability density function is

$$f(y;\mu,\theta) \quad = \quad \frac{\Gamma(y+\theta)}{\Gamma(\theta)\cdot y!}\cdot\frac{\mu^y\cdot\theta^\theta}{(\mu+\theta)^{y+\theta}}, \tag{4}$$

with mean μ and scale parameter θ. For every fixed θ, this is of type (1) and thus is another special case of the GLM framework. It also has $\phi = 1$ but with variance function $V(\mu) = \mu + \frac{\mu^2}{\theta}$. Package **MASS** (Venables and Ripley 2002) provides the family function `negative.binomial()` that can directly be plugged into `glm()` provided the `theta` argument is specified. One application would be the geometric model, the special case where $\theta = 1$, and can consequently be fitted in R by setting `family = negative.binomial(theta = 1)` in the `glm()` call.

If θ is not known but to be estimated from the data, the negative binomial model is not a special case of the general GLM—however, an ML fit can easily be computed re-using GLM methodology by iterating estimation of β given θ and vice versa. This leads to ML estimates for both β and θ which can be computed using the `glm.nb()` from the **MASS** package. It returns a model of class "`negbin`" inheriting from "`glm`" for which appropriate methods to the generic functions described above are again available.

2.2. Zero-inflated models

In addition to over-dispersion, many empirical count data sets exhibit more zero observations than would be allowed for by the Poisson model. Therefore, starting from Lambert (1992) various zero-inflated regression models have been suggested that extend the basic count data models by augmenting them with a point mass at zero—see Cameron and Trivedi (1998, 2005) for an overview.

Zero-inflated count models are two-component mixture models combining a point mass at zero with a count distribution such as Poisson, geometric or negative binomial. Thus, there are two sources of zeros: zeros may come from both the point mass and from the count component. For modeling the unobserved state (zero vs. count), a binary model is used: in the simplest case only with an intercept but potentially containing regressors.

More formally, the zero-inflated density is a mixture of the point mass at zero $I_{\{0\}}(y)$, a count distribution $f_{\text{count}}(y;x,\beta)$ and a binomial GLM $g(\pi_i) = z_i^\top\gamma$ that may depend on further regressors z_i:

$$f_{\text{zeroinfl}}(y;x,y,\beta,\gamma) \quad = \quad \pi\cdot I_{\{0\}}(y) \;+\; (1-\pi)\cdot f_{\text{count}}(y;x,\beta), \tag{5}$$

where π is the unobserved probability of belonging to the point mass component. The correspond-

ing regression equation for the mean is

$$\log(\mu_i) \quad = \quad \pi_i \cdot 0 \; + \; (1 - \pi_i) \cdot x_i^\top \beta. \tag{6}$$

The vector of regressors in the zero-inflation model z_i and the regressors in the count component x_i need to be distinct—in the simplest case, $z_i = 1$ is just an intercept. The default link function $g(\pi)$ in binomial GLMs is the logit link, but other links such as the probit are also available. The full set of parameters of β, γ, and potentially θ (if a negative binomial count model is used) can be estimated by ML. Inference is typically performed for β and γ while θ is treated as a nuisance parameter even if a negative binomial model is used.

In R, zero-inflated count data models can be fitted with the `zeroinfl()` function from the **pscl** package (Jackman 2007). Both its fitting function and the returned model objects of class "`zeroinfl`" are modelled after the corresponding GLM functionality in R. The arguments of `zeroinfl()` are given by

```
zeroinfl(formula, data, subset, na.action, weights, offset,
  dist = "poisson", link = "logit", control = zeroinfl.control(...),
  model = TRUE, y = TRUE, x = FALSE, ...)
```

where the first line contains the standard model-frame specifications, the second line has the arguments specific to zero-inflated models and the arguments in the last line control some components of the return value.

The `formula` mainly describes the count regression relationship of y_i and x_i, i.e., `y ~ x1 + x2` specifies a regression where all zero counts have the same probability π_i of belonging to the zero component. This is equivalent to the model `y ~ x1 + x2 | 1`, making it more explicit that the zero-inflation model only has an intercept. Additionally, further regressors z_i can be added to the zero-inflation model: A typical formula is `y ~ x1 + x2 | z1 + z2 + z3` and, as noted above, the regressors in the zero and the count component need not be distinct.

The model likelihood can be specified by the `dist` and `link` arguments. The former determines the count data distribution (`"poisson"` by default, but it can also be set to `"negbin"` or `"geometric"`) for which always a log link is used. The zero-inflation component is always a binomial GLM whose link function is specified by `link` (defaulting to `"logit"`, but all link functions of the `binomial()` family are also supported).

ML estimation of all parameters is carried out using R's `optim()`, with control options set in `zeroinfl.control()`. Starting values can be user-supplied, estimated by the expectation maximization (EM) algorithm, or by `glm.fit()` (the default). The latter corresponds to the first iteration of the EM algorithm and initializes the unobserved state as $y_i > 0$, i.e., all zeros are in the point mass component and only the non-zero counts in the count component. The covariance matrix estimate is derived numerically using the Hessian matrix returned by `optim()`. Using EM estimation for deriving starting values is typically a bit slower but numerically more stable. It already maximizes the likelihood, but a single `optim()` iteration is used for determining the covariance matrix estimate. See Appendix A for further technical details.

The returned fitted model object of class "`zeroinfl`" is a list similar to "`glm`" objects. Some of its elements—such as `$coefficients` or `$terms`—are again lists with a zero and count component, respectively. For details see Appendix A.

A set of standard extractor functions for fitted model objects is available for objects of class "`zeroinfl`", including the usual `summary()` method that provides partial Wald tests for all coefficients. No `anova()` method is provided, but the general `coeftest()`, `waldtest()` from **lmtest**, and `linear.hypothesis()` from **car** can be used for Wald tests and `lrtest()` from **lmtest** for LR tests of nested models.

2.3. Hurdle models

Originally proposed by Mullahy (1986) in the econometrics literature, hurdle models are another

model class for dealing with excess zero counts (see Cameron and Trivedi 1998, 2005, for an overview). They are also two-component models but avoid modeling zeros from mixed sources: A truncated count component is employed for positive counts and a hurdle component models zero vs. larger counts. For the latter either a binomial model or a censored count distribution can be employed.

More formally, the hurdle model combines a count data model $f_{\text{count}}(y; x, \beta)$ (that is left-truncated $y > 0$) and a zero hurdle model $f_{\text{zero}}(y; z, \gamma)$ (right-censored at $y = 1$):

$$f_{\text{hurdle}}(y; x, z, \beta, \gamma) = \begin{cases} f_{\text{zero}}(0; z, \gamma) & \text{if } y = 0, \\ (1 - f_{\text{zero}}(0; z, \gamma)) \cdot f_{\text{count}}(y; x, \beta)/(1 - f_{\text{count}}(0; x, \beta)) & \text{if } y > 0 \end{cases} \tag{7}$$

The model parameters β, γ, and potentially one or two additional θ dispersion parameters (if f_{count} or f_{zero} or both are negative binomial densities) are estimated by ML where the specification of the likelihood has the advantage that the count and the hurdle component can be maximized separately. The corresponding mean regression relationship is given by

$$\log(\mu_i) \quad = \quad x_i^\top \beta + \log(1 - f_{\text{zero}}(0; z_i, \gamma)) - \log(1 - f_{\text{count}}(0; x_i, \beta)). \tag{8}$$

For interpreting the zero model as a hurdle, a binomial GLM is probably the most intuitive specification[2]. Another useful interpretation arises if the same regressors $x_i = z_i$ are used in the same count model in both components $f_{\text{count}} = f_{\text{zero}}$: A test of the hypothesis $\beta = \gamma$ then tests whether the hurdle is needed or not.

In R, hurdle models can be fitted with the `hurdle()` function from the **pscl** package. Both the fitting function interface and the returned model objects of class "`hurdle`" are almost identical to the corresponding `zeroinfl()` functionality and again modelled after the corresponding GLM functionality in R. The arguments of `hurdle()` are given by

```
hurdle(formula, data, subset, na.action, weights, offset,
  dist = "poisson", zero.dist = "binomial", link = "logit",
  control = hurdle.control(...),
  model = TRUE, y = TRUE, x = FALSE, ...)
```

where all arguments have almost the same meaning as for `zeroinfl()`, only the default processing for the `formula` is slightly different: 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`.

Again, ML estimates of all parameters are obtained from `optim()`, with control options set in `hurdle.control()`. Starting values can be supplied, otherwise they are estimated by `glm.fit()` (the default). Covariance matrix estimates are derived numerically using the Hessian matrix returned by `optim()`. See Appendix B for details.

The returned fitted model object is of class "`hurdle`" whose structure is virtually identical to that of "`zeroinfl`" models. As above, a set of standard extractor functions for fitted model objects is available for objects of class "`hurdle`", including the usual `summary()` method that provides partial Wald tests for all coefficients. No `anova()` method is provided, but the general `coeftest()`, `waldtest()` from **lmtest**, and `linear.hypothesis()` from **car** can be used for Wald tests and `lrtest()` from **lmtest** for LR tests of nested models. The function `hurdletest()` is a convenience interface to `linear.hypothesis()` for testing for the presence of a hurdle (which is only applicable if the same regressors and the same count distribution is used in both components).

[2]Note that binomial logit and censored geometric models as the hurdle part both lead to the same likelihood function and thus to the same coefficient estimates.

3. Application and illustrations

In the following, we illustrate all models described above by applying them to a cross-sectional data set. Before the parametric models are fitted, a basic exploratory analysis of the data set is carried out that addresses some problems typically encountered when visualizing count data. At the end of the section, all fitted models are compared highlighting that the modelled mean function is similar but the fitted likelihood is different and thus, the models differ with respect to explaining over-dispersion and/or the number of zero counts.

3.1. Demand for medical care by the elderly

Deb and Trivedi (1997) analyze data on 4406 individuals, aged 66 and over, who are covered by Medicare, a public insurance program. Originally obtained from the US National Medical Expenditure Survey in 1987/88, the data is available from the data archive of the *Journal of Applied Econometrics* at `http://www.econ.queensu.ca/jae/1997-v12.3/deb-trivedi/`. It was prepared for an R package accompanying Kleiber and Zeileis (2008) and is also available as `DebTrivedi.rda` in *Journal of Statistical Software* together with Zeileis (2006). The objective is to model the demand for medical care—as captured in the number of physician/non-physician office and hospital outpatient visits—by the covariates available for the patients. Here, we adopt the number of physician office visits `ofp` as the dependent variable and use the health status variables `hosp` (number of hospital stays), `health` (self-perceived health status), `numchron` (number of chronic conditions), as well as the socio-economic variables `gender`, `school` (number of years of education), and `privins` (private insurance indicator) as regressors. For convenience, we select the variables used from the full data set.

```
R> dt <- DebTrivedi[, c(1, 6:8, 13, 15, 18)]
```

To obtain first overview of the dependent variable, we employ a histogram of the observed count frequencies. In R various tools could be used, e.g., via `hist(dt$ofp, breaks = 0:90 - 0.5)` for a histogram with rectangles or via

```
R> plot(table(dt$ofp))
```

(see Figure 1) for a histogram with lines which brings out the extremely large counts a bit better. The histogram illustrates that the marginal distribution exhibits both substantial variation and a rather large number of zeros.

A natural second step in the exploratory analysis is to look at pairwise bivariate displays of the dependent variable against each of the regressors bringing out the partial relationships. In R, such bivariate displays can easily be generated with the formula `plot()` method, e.g., via `plot(y ~ x)`. This chooses different types of displays depending on the combination of quantitative and qualitative variables as dependent or regressor variable, respectively. However, count variables are treated as all numerical variables and therefore the command

```
R> plot(ofp ~ numchron, data = dt)
```

produces a simple scatterplot as shown in the left panel of Figure 2. This is clearly not useful as both variables are count variables which produces numerous ties in the bivariate distribution and thus obscuring a large number of points in the display. To overcome the problem, it is useful to group the number of chronic conditions into a factor with levels '0', '1', '2', and '3 or more' and produce a boxplot instead of a scatterplot. Furthermore, the picture is much clearer if the dependent variable is log-transformed (just as all count regression models discussed above also use a log lik by default). As there are zero counts as well, we use a convenience function `clog()` providing a continuity-corrected logarithm.

```
R> clog <- function(x) log(x + 0.5)
```

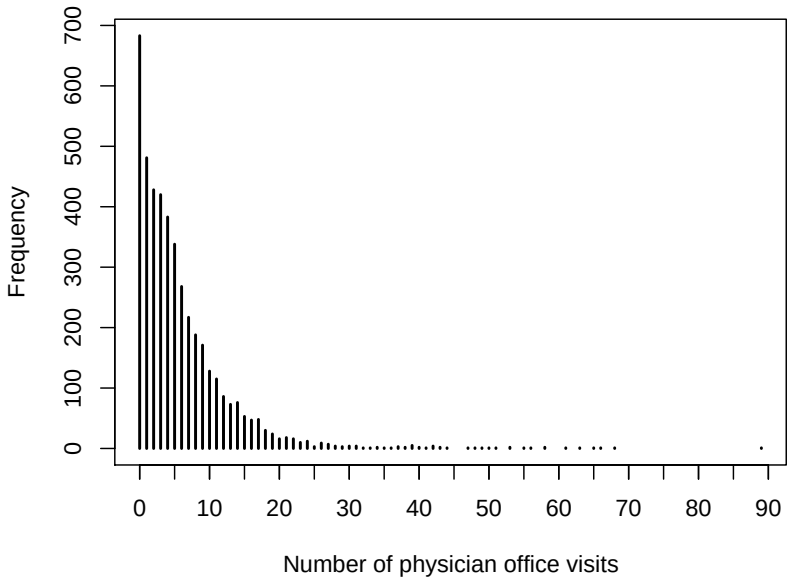


Figure 1: Frequency distribution for number of physician office visits.

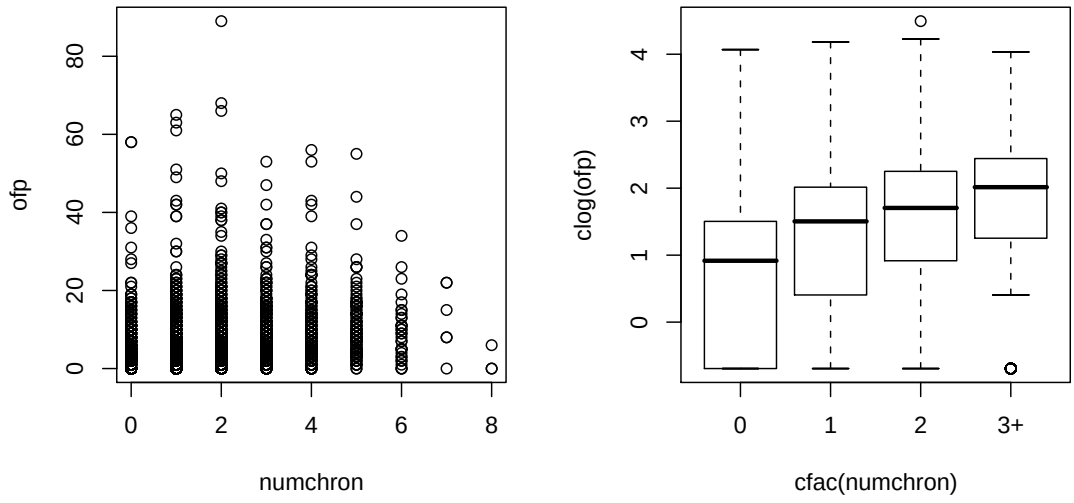


Figure 2: Bivariate explorative displays for number of physician office visits plotted against number of chronic conditions.

For transforming a count variable to a factor (for visualization purposes only), we define another convenience function `cfac()`

```
R> cfac <- function(x, breaks = NULL) {
+     if (is.null(breaks)) breaks <- unique(quantile(x, 0:10/10))
+     x <- cut(x, breaks, include.lowest = TRUE, right = FALSE)
+     levels(x) <- paste(breaks[-length(breaks)], ifelse(diff(breaks) >
+         1, c(paste("-", breaks[-c(1, length(breaks))] - 1, sep = ""),
+         "+"), ""), sep = "")
+     return(x)
+ }
```

which by default tries to take an educated guess how to choose the breaks between the categories. Clearly, the resulting exploratory display of the transformed variables produced by

```
R> plot(clog(ofp) ~ cfac(numchron), data = dt)
```

(shown in the right panel of Figure 2) brings out much better how the number of doctor visits increases with the number of chronic conditions.

Analogous displays for the number of physician office visits against all regressors can be produced via

```
R> plot(clog(ofp) ~ health, data = dt, varwidth = TRUE)
R> plot(clog(ofp) ~ cfac(numchron), data = dt)
R> plot(clog(ofp) ~ privins, data = dt, varwidth = TRUE)
R> plot(clog(ofp) ~ cfac(hosp, c(0:2, 8)), data = dt)
R> plot(clog(ofp) ~ gender, data = dt, varwidth = TRUE)
R> plot(cfac(ofp, c(0:2, 4, 6, 10, 100)) ~ school, data = dt, breaks = 9)
```

and are shown (with slightly enhanced labeling) in Figure 3. The last plot uses a different type of display. Here, the dependent count variable is not log-transformed but grouped into a factor and then a spinogram is produced. This also groups the regressor (as in a histogram) and then produces a highlighted mosaic plot. All displays show that the number of doctor visits in- or decreases with the regressors as expected: `ofp` decreases with the general health status but increases with the number of chronic conditions or hospital stays. The average number of visits is also slightly higher for patients with a private insurance and higher level of education. It is slightly lower for male compared to female patients. The overall impression from all displays is that the changes in the mean can only explain a modest amount of variation in the data.

3.2. Poisson regression

As a first attempt to capture the relationship between the number of physician office visits and all regressors `ofp ~ .` in a parametric regression model, we fit the basic Poisson regression model

```
R> fm_pois <- glm(ofp ~ ., data = dt, family = poisson)
```

and obtain the coefficient estimates along with associated partial Wald tests

```
R> summary(fm_pois)

Call:
glm(formula = ofp ~ ., family = poisson, data = dt)

Deviance Residuals:
```

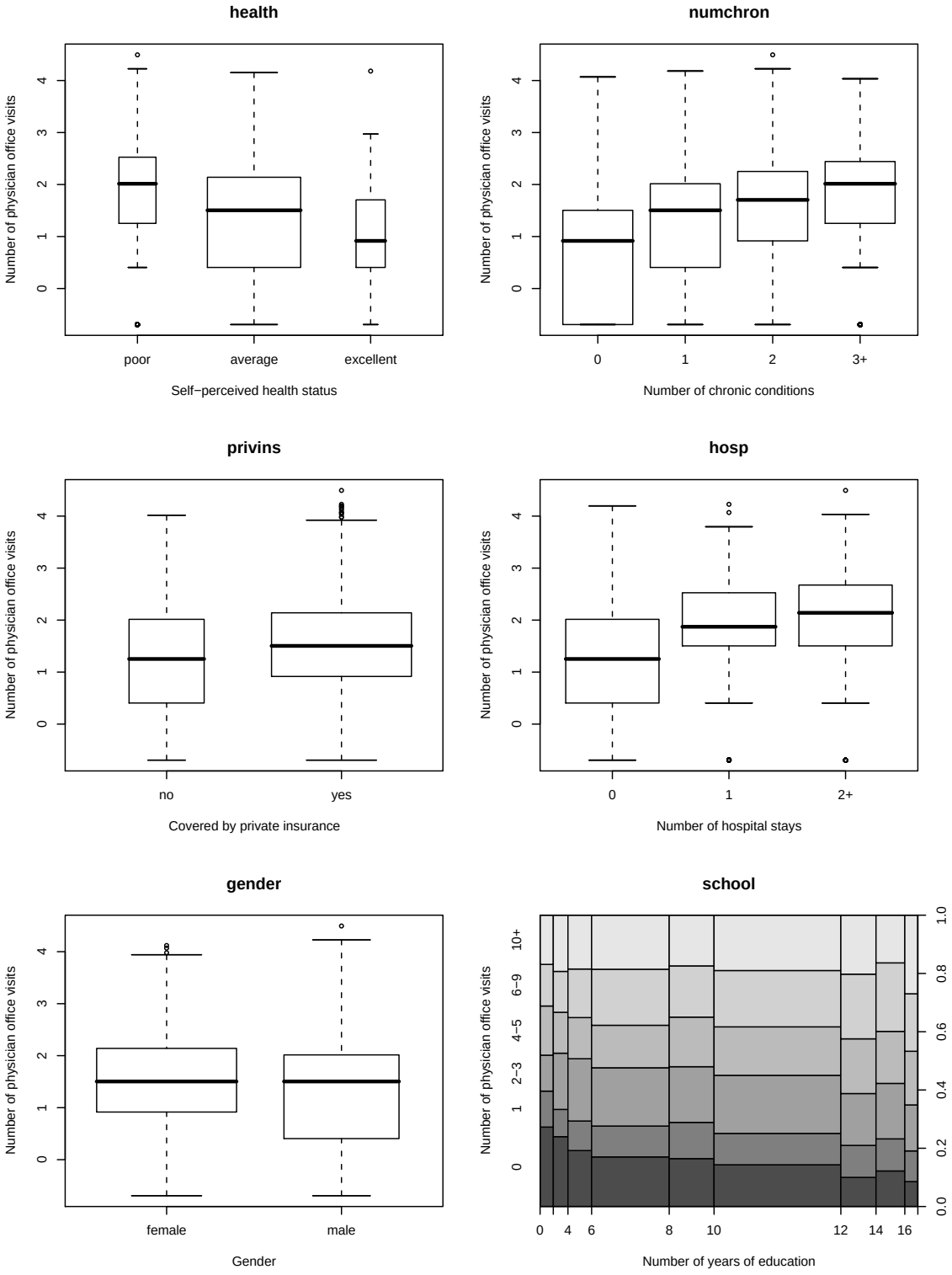


Figure 3: Number of physician office visits plotted against regressors used.

```
    Min       1Q   Median       3Q      Max
-8.4055  -1.9962  -0.6737   0.7049  16.3620

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      1.028874   0.023785  43.258   <2e-16 ***
hosp             0.164797   0.005997  27.478   <2e-16 ***
healthpoor       0.248307   0.017845  13.915   <2e-16 ***
healthexcellent -0.361993   0.030304 -11.945   <2e-16 ***
numchron         0.146639   0.004580  32.020   <2e-16 ***
gendermale      -0.112320   0.012945  -8.677   <2e-16 ***
school           0.026143   0.001843  14.182   <2e-16 ***
privinsyes       0.201687   0.016860  11.963   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 26943  on 4405  degrees of freedom
Residual deviance: 23168  on 4398  degrees of freedom
AIC: 35959

Number of Fisher Scoring iterations: 5
```

All coefficient estimates confirm the results from the exploratory analysis in Figure 3. In terms of significance, the health variables are more important than the socio-economic variables. However, the Wald test results might be too optimistic due to a misspecification of the likelihood. As the exploratory analysis suggested that over-dispersion is present in this data set, we re-compute the Wald tests using sandwich standard errors

```
R> coeftest(fm_pois, vcov = sandwich)

z test of coefficients:

                Estimate Std. Error z value  Pr(>|z|)
(Intercept)      1.028874   0.064530 15.9442 < 2.2e-16 ***
hosp             0.164797   0.021945  7.5095 5.935e-14 ***
healthpoor       0.248307   0.054022  4.5964 4.298e-06 ***
healthexcellent -0.361993   0.077449 -4.6740 2.954e-06 ***
numchron         0.146639   0.012908 11.3605 < 2.2e-16 ***
gendermale      -0.112320   0.035343 -3.1780  0.001483 **
school           0.026143   0.005084  5.1422 2.715e-07 ***
privinsyes       0.201687   0.043128  4.6765 2.919e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

All regressors are still significant but the standard errors seem to be more appropriate. This will also be confirmed by the following models that deal with over-dispersion (and excess zeros) in a more formal way.

3.3. Quasi-Poisson regression

The quasi-Poisson model

```
R> fm_qpois <- glm(ofp ~ ., data = dt, family = quasipoisson)
```

leads to an estimated dispersion of $\hat{\phi} = 6.706$ which is clearly larger than 1 confirming that over-dispersion is present in the data. The resulting partial Wald tests of the coefficients

```
R> summary(fm_qpois)

Call:
glm(formula = ofp ~ ., family = quasipoisson, data = dt)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-8.4055  -1.9962  -0.6737   0.7049  16.3620

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)      1.028874   0.061594  16.704  < 2e-16 ***
hosp             0.164797   0.015531  10.611  < 2e-16 ***
healthpoor       0.248307   0.046211   5.373 8.13e-08 ***
healthexcellent -0.361993   0.078476  -4.613 4.09e-06 ***
numchron         0.146639   0.011860  12.364  < 2e-16 ***
gendermale      -0.112320   0.033523  -3.351 0.000813 ***
school           0.026143   0.004774   5.477 4.58e-08 ***
privinsyes       0.201687   0.043661   4.619 3.96e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 6.706254)

    Null deviance: 26943  on 4405  degrees of freedom
Residual deviance: 23168  on 4398  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 5
```

are rather similar to the results obtained from the Poisson regression with sandwich standard errors, leading to the same conclusions.

3.4. Negative binomial regression

A more formal way to accommodate over-dispersion in a count data regression model is to use a negative binomial model.

```
R> fm_nbin <- glm.nb(ofp ~ ., data = dt)
```

However, both the regression coefficients and their associated partial Wald statistics are rather similar to the quasi-Poisson and the sandwich-adjusted Poisson results above:

```
R> summary(fm_nbin, correlation = FALSE)

Call:
glm.nb(formula = ofp ~ ., data = dt, init.theta = 1.20660353415217,
    link = log)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
```

```
-3.0469  -0.9955  -0.2948   0.2961   5.8185

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      0.929257   0.054591  17.022  < 2e-16 ***
hosp             0.217772   0.020176  10.793  < 2e-16 ***
healthpoor       0.305013   0.048511   6.288 3.23e-10 ***
healthexcellent -0.341807   0.060924  -5.610 2.02e-08 ***
numchron         0.174916   0.012092  14.466  < 2e-16 ***
gendermale      -0.126488   0.031216  -4.052 5.08e-05 ***
school           0.026815   0.004394   6.103 1.04e-09 ***
privinsyes       0.224402   0.039464   5.686 1.30e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(1.2066) family taken to be 1)

    Null deviance: 5743.7  on 4405  degrees of freedom
Residual deviance: 5044.5  on 4398  degrees of freedom
AIC: 24359

Number of Fisher Scoring iterations: 1


              Theta:  1.2066
          Std. Err.:  0.0336

 2 x log-likelihood:  -24341.1070
```

3.5. Hurdle regression

The exploratory analysis conveyed the impression that there might be more zero observations than explained by the basic count data distributions, hence a negative binomial hurdle model is fitted via

```
R> fm_hurdle0 <- hurdle(ofp ~ ., data = dt, dist = "negbin")
```

This uses the same type of count data model as in the preceeding section but it is now truncated for `ofp < 1` and has an additional hurdle component modeling zero vs. count observations. By default, the hurdle component is a binomial GLM which contains all regressors used in the count model. The associated coefficient estimates and partial Wald tests for both model components are displayed via

```
R> summary(fm_hurdle0)

Call:
hurdle(formula = ofp ~ ., data = dt, dist = "negbin")


Count model coefficients (truncated negbin with log link):
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      1.197590   0.058973  20.307  < 2e-16 ***
hosp             0.211898   0.021396   9.904  < 2e-16 ***
healthpoor       0.315975   0.048056   6.575 4.86e-11 ***
```

```
healthexcellent -0.331874   0.066093  -5.021 5.13e-07 ***
numchron         0.126423   0.012452  10.153  < 2e-16 ***
gendermale      -0.068320   0.032416  -2.108   0.0351 *
school           0.020705   0.004535   4.566 4.98e-06 ***
privinsyes       0.100133   0.042619   2.350   0.0188 *
Log(theta)       0.333253   0.042755   7.795 6.46e-15 ***
Zero hurdle model coefficients (binomial with logit link):
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      0.043147   0.139851   0.309 0.757687
hosp             0.312449   0.091437   3.417 0.000633 ***
healthpoor      -0.008716   0.161024  -0.054 0.956833
healthexcellent -0.289570   0.142682  -2.029 0.042409 *
numchron         0.535213   0.045378  11.794  < 2e-16 ***
gendermale      -0.415658   0.087608  -4.745 2.09e-06 ***
school           0.058541   0.011989   4.883 1.05e-06 ***
privinsyes       0.747120   0.100880   7.406 1.30e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Theta: count = 1.3955
Number of iterations in BFGS optimization: 16
Log-likelihood: -1.209e+04 on 17 Df
```

The coefficients in the count component resemble those from the previous models, but the model is improved by including the hurdle component. However, it might be possible to omit the `health` variable from the hurdle model. To test this hypothesis, the reduced model is fitted via

```
R> fm_hurdle <- hurdle(ofp ~ . | hosp + numchron + privins + school +
+     gender, data = dt, dist = "negbin")
```

and can then be compared to the full model in a Wald test

```
R> waldtest(fm_hurdle0, fm_hurdle)

Wald test

Model 1: ofp ~ .
Model 2: ofp ~ . | hosp + numchron + privins + school + gender
  Res.Df   Df  Chisq Pr(>Chisq)
1   4389
2   4391   -2 4.1213     0.1274
```

or LR test

```
R> lrtest(fm_hurdle0, fm_hurdle)

Likelihood ratio test

Model 1: ofp ~ .
Model 2: ofp ~ . | hosp + numchron + privins + school + gender
  #Df LogLik Df  Chisq Pr(>Chisq)
1  17 -12088
2  15 -12090 -2 3.9875     0.1362
```

both leading to similar (non-significant) results.

3.6. Zero-inflated regression

A different way of augmenting the negative binomial count model `fm_nbin` with additional probability weight for zero counts is a zero-inflated negative binomial (ZINB) regression. The default model is fitted via

```
R> fm_zinb0 <- zeroinfl(ofp ~ ., data = dt, dist = "negbin", EM = TRUE)
```

using EM estimation which is numerically more stable, especially for ZINB models. This has just an intercept in the zero-inflation model, but—as the hurdle model `fm_hurdle` fitted above has shown—the available regressors can be used for distinguishing between zero and larger counts. Therefore, a second model is fitted

```
R> fm_zinb <- zeroinfl(ofp ~ . | hosp + numchron + privins + school +
+     gender, data = dt, dist = "negbin", EM = TRUE)
```

that has the same variables in the zero-inflation part as the hurdle component in `fm_hurdle`. This improves the ZINB fit significantly which can again be brought out by a Wald test

```
R> waldtest(fm_zinb0, fm_zinb)

Wald test

Model 1: ofp ~ .
Model 2: ofp ~ . | hosp + numchron + privins + school + gender
  Res.Df   Df  Chisq Pr(>Chisq)
1   4396
2   4391    5 115.76  < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

or a LR test `lrtest(fm_zinb0, fm_zinb)` that produces virtually identical results. The chosen fitted model can again be inspected via

```
R> summary(fm_zinb)

Call:
zeroinfl(formula = ofp ~ . | hosp + numchron + privins + school + gender,
    data = dt, dist = "negbin", EM = TRUE)


Count model coefficients (negbin with log link):
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      1.193753   0.056659  21.069  < 2e-16 ***
hosp             0.201477   0.020359   9.896  < 2e-16 ***
healthpoor       0.285133   0.045092   6.323 2.56e-10 ***
healthexcellent -0.319339   0.060404  -5.287 1.25e-07 ***
numchron         0.128995   0.011930  10.812  < 2e-16 ***
gendermale      -0.080270   0.031024  -2.587  0.00967 **
school           0.021423   0.004357   4.916 8.82e-07 ***
privinsyes       0.125843   0.041587   3.026  0.00248 **
Log(theta)       0.394196   0.035034  11.252  < 2e-16 ***
```

```
Zero-inflation model coefficients (binomial with logit link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.04694    0.26852  -0.175  0.86123
hosp        -0.80004    0.42054  -1.902  0.05712 .
numchron    -1.24761    0.17823  -7.000 2.56e-12 ***
privinsyes  -1.17560    0.22008  -5.342 9.21e-08 ***
school      -0.08376    0.02625  -3.191  0.00142 **
gendermale   0.64765    0.20008   3.237  0.00121 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Theta = 1.4832
Number of iterations in BFGS optimization: 1
Log-likelihood: -1.209e+04 on 15 Df
```

3.7. Comparison

Having fitted several count data regression models to the demand for medical care data set, it is, of course, of interest to understand what these models have in common and what their differences are. As a first comparison, it is of natural interest to inspect the estimated regression coefficients in the count data model

```
R> fm <- list("ML-Pois" = fm_pois, "Quasi-Pois" = fm_qpois, NB = fm_nbin,
+     "Hurdle-NB" = fm_hurdle, ZINB = fm_zinb)
R> round(sapply(fm, function(x) coef(x)[1:8]), digits = 3)

                ML-Pois Quasi-Pois     NB Hurdle-NB   ZINB
(Intercept)       1.029      1.029  0.929     1.198  1.194
hosp              0.165      0.165  0.218     0.212  0.201
healthpoor        0.248      0.248  0.305     0.316  0.285
healthexcellent  -0.362     -0.362 -0.342    -0.332 -0.319
numchron          0.147      0.147  0.175     0.126  0.129
gendermale       -0.112     -0.112 -0.126    -0.068 -0.080
school            0.026      0.026  0.027     0.021  0.021
privinsyes        0.202      0.202  0.224     0.100  0.126
```

This shows that there are some small differences, especially between the GLMs and the zero-augmented models. However, the overall impression is that the estimated mean functions are rather similar. Moreover, the associated estimated standard errors are very similar as well:

```
R> round(cbind("ML-Pois" = sqrt(diag(vcov(fm_pois))),
+     "Adj-Pois" = sqrt(diag(sandwich(fm_pois))),
+     sapply(fm[-1], function(x) sqrt(diag(vcov(x)))[1:8])), digits = 3)

                ML-Pois Adj-Pois Quasi-Pois    NB Hurdle-NB  ZINB
(Intercept)       0.024    0.065      0.062 0.061     0.059 0.057
hosp              0.006    0.022      0.016 0.023     0.021 0.020
healthpoor        0.018    0.054      0.046 0.054     0.048 0.045
healthexcellent   0.030    0.077      0.078 0.068     0.066 0.060
numchron          0.005    0.013      0.012 0.014     0.012 0.012
gendermale        0.013    0.035      0.034 0.035     0.032 0.031
school            0.002    0.005      0.005 0.005     0.005 0.004
privinsyes        0.017    0.043      0.044 0.044     0.043 0.042
```

The only exception are the model-based standard errors for the Poisson model when treated as a fully specified model which is obviously not appropriate for this data set.

In summary, the models are not too different with respect to their fitted mean functions. The differences become obvious if not only the mean but the full likelihood is considered:

```
R> rbind(logLik = sapply(fm, function(x) round(logLik(x), digits = 0)),
+     Df = sapply(fm, function(x) attr(logLik(x), "df")))

       ML-Pois Quasi-Pois     NB Hurdle-NB   ZINB
logLik  -17972         NA -12171    -12090 -12091
Df            8          8      9        15     15
```

The ML Poisson model is clearly inferior to all other fits. The quasi-Poisson model (as the sandwich-adjusted Poisson model) is not associated with a fitted likelihood. The negative binomial already improves the fit dramatically but can in turn be improved by the hurdle and zero-inflated models which give almost identical fits. This also reflects that the over-dispersion in the data is captured better by the negative-binomial-based models than the plain Poisson model. Additionally it is of interest how the zero counts are captured by the various models. Therefore, the observed zero counts are compared to expected number of zero counts for the likelihood based models:

```
R> round(c(Obs = sum(dt$ofp < 1), "ML-Pois" = sum(dpois(0, fitted(fm_pois))),
+     "Adj-Pois" = NA, "Quasi-Pois" = NA,
+     NB = sum(dnbinom(0, mu = fitted(fm_nbin), size = fm_nbin$theta)),
+     "NB-Hurdle" = sum(predict(fm_hurdle, type = "prob")[, 1]),
+     ZINB = sum(predict(fm_zinb, type = "prob")[, 1])))

       Obs    ML-Pois   Adj-Pois Quasi-Pois         NB  NB-Hurdle       ZINB
       683         47         NA         NA        608        683        709
```

Thus, the ML Poisson model is again not appropriate whereas the negative-binomial-based models are much better in modeling the zero counts. By construction, the expected number of zero counts in the hurdle model matches the observed number.

In summary, the hurdle and zero-inflation models lead to the best fitted likelihoods on this data set. Above, their mean function for the count component was already shown to be very similar, below we take a look at the fitted zero components:

```
R> t(sapply(fm[4:5], function(x) round(x$coefficients$zero, digits = 3)))

          (Intercept)   hosp numchron privinsyes school gendermale
Hurdle-NB       0.016  0.318    0.548      0.746  0.057     -0.419
ZINB           -0.047 -0.800   -1.248     -1.176 -0.084      0.648
```

This shows that the absolute values are rather different—which is not surprising as they pertain to slightly different ways of modeling zero counts—but the signs of the coefficients match, i.e., are just inversed. For the hurdle model, the zero hurdle component describes the probability of observing a positive count whereas, for the ZINB model, the zero-inflation component predicts the probability of observing a zero count from the point mass component. Overall, both models lead to the same qualitative results and very similar model fits. Probably, the hurdle model is slightly preferable because it has the nicer interpretation: there is one process that controls whether a patient sees a physician or not, and a second process that determines how many office visits are made.

4. Summary

The model frame for basic count data models from the GLM framework as well as their implementation in the R system for statistical computing is reviewed. Starting from these basic tools, it is presented how zero-inflated and hurdle models extend the classical models and how likewise their R implementation in package **pscl** re-uses design and functionality of the corresponding R software. Hence, the new functions `zeroinfl()` and `hurdle()` are straightforward to apply for model fitting. Additionally, standard methods for diagnostics are provided and generic inference tools from other packages can easily be re-used.

Computational details

The results in this paper were obtained using R 2.4.1 with the packages **MASS** 7.2–30, **pscl** 0.90, **sandwich** 2.0–2, **car** 1.2–1, **lmtest** 0.9–19. R itself and all packages used are available from CRAN at http://CRAN.R-project.org/.

References

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

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

Chambers JM, Hastie TJ (eds.) (1992). *Statistical Models in S*. Chapman & Hall, London.

Deb P, Trivedi PK (1997). "Demand for Medical Care by the Elderly: A Finite Mixture Approach." *Journal of Applied Econometrics*, **12**, 313–336.

Fox J (2002). *An R and S-PLUS Companion to Applied Regression*. Sage Publications, Thousand Oaks, CA.

Halekoh U, Højsgaard S, Yan J (2006). "The R Package **geepack** for Generalized Estimating Equations." *Journal of Statistical Software*, **15**(2), 1–11. URL http://www.jstatsoft.org/v15/i02/.

Jackman S (2007). **pscl***: Classes and Methods for R Developed in the Political Science Computational Laboratory, Stanford University*. Department of Political Science, Stanford University, Stanford, California. R package version 0.90, URL http://pscl.stanford.edu/.

Kleiber C, Zeileis A (2008). *Applied Econometrics with R*. Springer-Verlag, New York. Forthcoming.

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

Leisch F (2004). "FlexMix: A General Framework for Finite Mixture Models and Latent Class Regression in R." *Journal of Statistical Software*, **11**(8), 1–18. URL http://www.jstatsoft.org/v11/i08/.

McCullagh P, Nelder JA (1989). *Generalized Linear Models*. Chapman & Hall, London, 2nd edition.

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

Mwalili SM (2006). **zicounts**: *Classical and Censored Zero-inflated Count Data Models*. R package version 1.1.4, URL http://CRAN.R-project.org/.

Nelder JA, Wedderburn RWM (1972). "Generalized Linear Models." *Journal of the Royal Statistical Society A*, **135**, 370–384.

Pinheiro JC, Bates DM (2000). *"Mixed-Effects Models in S and S-PLUS*. Springer-Verlag, New York.

R Development Core Team (2007). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3, URL http://www.R-project.org/.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Springer-Verlag, New York, 4th edition.

Zeileis A (2004). "Econometric Computing with HC and HAC Covariance Matrix Estimators." *Journal of Statistical Software*, **11**(10), 1–17. URL http://www.jstatsoft.org/v11/i10/.

Zeileis A (2006). "Object-oriented Computation of Sandwich Estimators." *Journal of Statistical Software*, **16**(9), 1–16. URL http://www.jstatsoft.org/v16/i09/.

Zeileis A, Hothorn T (2002). "Diagnostic Checking in Regression Relationships." *R News*, **2**(3), 7–10. URL http://CRAN.R-project.org/doc/Rnews/.

A. Technical details for zero-inflated models

The fitting of zero-inflated models via ML in `zeroinfl()` is controlled by the arguments in the `zeroinfl.control()` wrapper function:

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

This modifies some default arguments passed on to the optimizer `optim()`, such as `method`, `maxit` and `trace`. The latter is also used within `zeroinfl()` and can be set to produce more verbose output concerning the fitting process. The arguments `EM` and `start` control the choice of starting values for calling `optim()`, all remaining arguments passed through `...` are directly passed on to `optim()`.

By default, starting values are estimated by calling `glm.fit()` for both components of the model separately, corresponding to the first iteration of an EM (expectation maximization) approach where the unobserved state (zero vs. count component) is initialized as $y_i > 0$, i.e., all zeros are in the perfect component and only the non-zero counts in the count component. If `EM = TRUE`, this process is iterated until convergence of the parameters to the ML estimates. The optimizer is still called subsequently for a single iteration to obtain the Hessian matrix from which the estimated covariance matrix can be computed. If starting values are supplied, `start` needs to be set to a named list with the parameters for the `$count` and `$zero` part of the model (and potentially a `$theta` dispersion parameter if a negative binomial distribution is used).

The fitted model object of class "`zeroinfl`" is similar to "`glm`" objects and contains sufficient information on all aspects of the fitting process. In particular, the estimated parameters and associated covariances are contained as well as the result from the `optim()` call. Furthermore, the call, formula, terms structure etc. is contained, potentially also the model frame, dependent variable and regressor matrices.

Following `glm.nb()`, the θ parameter of the negative binomial distribution is treated as a nuisance parameter. Thus, the `$coefficients` component of the fitted model object just contains estimates of β and γ while the estimate of θ and its standard deviation (on a log scale) are kept in extra list elements `$theta` and `$SE.logtheta`.

B. Technical details for hurdle models

Both the interface of the `hurdle()` function as well as its fitted model objects are virtually identical to the corresponding "`zeroinfl`" functionality. Hence, we only provide some additional information for those aspects that differ from those discussed above. The details of the ML optimization are again provided by a `hurdle.control()` wrapper:

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

The only new argument here is the `separate` argument which controls whether the two components of the model are optimized separately (the default) or not. This is possible because there are no mixed sources for the zeros in the data (unlike in zero-inflation models).

C. Methods for fitted zero-inflated and hurdle models

Users typically should not need to compute on the internal structure of "`zeroinfl`" or "`hurdle`" objects because a set of standard extractor functions is provided, including methods to the generic functions `print()` and `summary()` which print the estimated coefficients along with 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.zeroinfl`" or "`summary.hurdle`", respectively, 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 their `model` argument, the estimates for a single component only can be extracted. Concatenating the parameters by default and providing a matching covariance matrix estimate (that does not contain the covariances of further nuisance parameters) facilitates the application of generic inference function such as `coeftest()`, `waldtest()`, and `linear.hypothesis()`. All of these compute Wald tests for which coefficient estimates and associated covariances is essentially all information required and can therefore be queried in an object-oriented way with the `coef()` and `vcov()` methods.

Similarly, 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 and `lrtest()` for conducting LR tests of nested models.

The `predict()` method computes predicted means (default) or probabilities (i.e., likelihood contributions) for observed or new observations. Predicted means for the observed data can also be obtained by the `fitted()` method. Deviations between observed counts y_i and predicted means $\hat{\mu}_i$ can be obtained by the `residuals()` method returning either raw residuals $y_i - \hat{\mu}_i$ or the scaled Pearson residuals $(y_i - \hat{\mu}_i)/\sqrt{\hat{\mu}_i}$ (the default).