What is Zero-Inflated Negative Binomial Regression and how is it used in R data analysis?

Authored by

stats writer

June 29, 2024

Zero-Inflated Negative Binomial Regression is a statistical method used in R data analysis to model count data with excessive zeros. It combines two models, the Zero-Inflated model and the Negative Binomial model, to account for both the excess zeros and overdispersion in the data. This method is commonly used in situations where the count data has a large number of zeros and traditional regression models, such as Poisson Regression, are inadequate. It allows for the identification of factors that contribute to the excess zeros and also estimates the effects of covariates on the non-zero counts. In R data analysis, this technique is implemented using the "zeroinfl" function in the "pscl" package. It is commonly used in fields such as epidemiology, ecology, and social sciences to analyze data with a high proportion of zeros.

Zero-Inflated Negative Binomial Regression | R Data Analysis Examples

Zero-inflated negative binomial regression is for modeling count variables with excessive zeros and it is usually for over-dispersed count outcome variables. Furthermore, theory suggests that the excess zeros are generated by a separate process from the count values and that the excess zeros can be modeled independently.

This page uses the following packages. Make sure that you can load them before trying to run the examples on this page. If you do not have a package installed, run: `install.packages("packagename")`, or if you see the version is out of date, run: `update.packages()`.

```
require(ggplot2)require(pscl)require(MASS)require(boot)
```

```
Version info: Code for this page was tested in R version 3.1.1 (2014-07-10)
On: 2014-08-11
With: boot 1.3-11; knitr 1.6; pscl 1.04.4; vcd 1.3-1; gam 1.09.1; coda 0.16-1;
mvtnorm 1.0-0; GGally 0.4.7; plyr 1.8.1; MASS 7.3-33; Hmisc 3.14-4; Formula
1.1-2; survival 2.37-7; psych 1.4.5; reshape2 1.4; msm 1.4; phia 0.1-5;
RColorBrewer 1.0-5; effects 3.0-0; colorspace 1.2-4; lattice 0.20-29; pequod
0.0-3; car 2.0-20; ggplot2 1.0.0
```

Please note: The purpose of this page is to show how to use various data analysis commands. It does not cover all aspects of the research process which researchers are expected to do. In particular, it does not cover data cleaning and checking, verification of assumptions, model diagnostics or potential follow-up analyses.

Examples of zero-inflated negative binomial regression

Example 1. School administrators study the attendance behavior of high school juniors at two schools. Predictors of the number of days of absence include gender of the student and standardized test scores in

math and language arts.

Example 2. The state wildlife biologists want to model how many fish are
being caught by fishermen at a state park. Visitors are asked how long they
stayed, how many people were in the group, were there children in the group and
how many fish were caught. Some visitors do not fish, but there is no data on
whether a person fished or not. Some visitors who did fish did not catch any
fish so there are excess zeros in the data because of the people that did not
fish.

Description of the Data

Let's pursue Example 2 from above.

We have data on 250 groups that went to a park. Each group was questioned
about how many fish they caught (`count`), how many children were in the
group (`child`), how many people were in the group (`persons`),

and
whether or not they brought a camper to the park (`camper`).

In addition to predicting the number of fish caught, there is interest in
predicting the existence of excess zeros, i.e., the probability that a group
caught zero fish. We will use the variables `child`, `persons`, and
`camper` in our model. Let's look at the data.

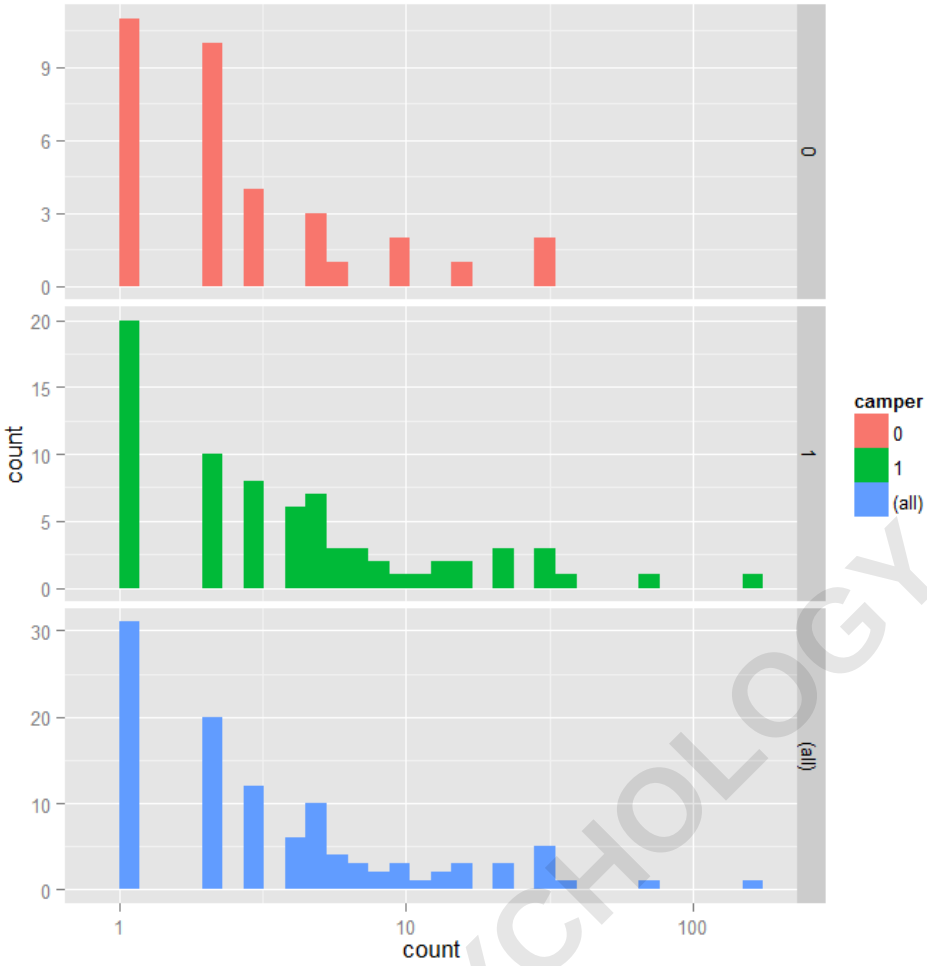
```
zinb<-
read.csv("https://stats.idre.ucla.edu/stat/data/fish.csv")zinb<-within(zinb, {nofish<-factor(nofish)livebait<-factor(livebait)camper<-factor(camper)})summary(zinb)
```

```
## nofish livebait camper persons child xb
## 0:176 0: 34 0:103 Min. :1.00 Min. :0.000 Min. :-3.275
## 1: 74 1:216 1:147 1st Qu.:2.00 1st Qu.:0.000 1st Qu.: 0.008
## Median :2.00 Median :0.000 Median : 0.955
## Mean :2.53 Mean :0.684 Mean : 0.974
## 3rd Qu.:4.00 3rd Qu.:1.000 3rd Qu.: 1.964
## Max. :4.00 Max. :3.000 Max. : 5.353
```

```
## zg count
## Min. :-5.626 Min. : 0.0
## 1st Qu.:-1.253 1st Qu.: 0.0
## Median : 0.605 Median : 0.0
## Mean : 0.252 Mean : 3.3
## 3rd Qu.: 1.993 3rd Qu.: 2.0
## Max. : 4.263 Max. :149.0

## histogram with x axis in log10 scaleggplot(zinb,aes(count,fill= camper))+geom_histogram()+scale_x_log10()+facet_grid(camper~.,margins=TRUE,scales="free_y")

## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



Analysis methods you might consider

Before we show how you can analyze this with a zero-inflated negative binomial analysis, let's consider some other methods that you might use.

Zero-inflated negative binomial regression

A zero-inflated model assumes that zero outcome is due to two different processes. For instance, in the example of fishing

presented here, the two
processes are that a subject has gone fishing vs. not gone fishing. If not gone fishing, the only
outcome possible is zero. If gone fishing, it is then a count
process. The two parts of the a zero-inflated model are a binary model, usually
a logit model to model which of the two processes the zero outcome is associated
with and a count model, in this case, a negative binomial model, to model the
count process. The expected count is expressed as a combination of the two
processes. Taking the example of fishing again:

$$
E(n_{text{fish caught}} = k) = P(text{not gone fishing}) * 0 + P(text{gone fishing}) * E(y = k | text{gone fishing})
$$

To understand the zero-inflated negative binomial regression,
let's start with the negative binomial model. There are multiple

parameterizations of the negative binomial model, we focus on NB2.
The negative binomial probability density function is:

$$
PDF(y; p, r) = frac{(y_i + r - 1)!}{y_i!(r-1)!}p_{i}^{r}(1 - p_i)^{y_i}
$$

where (p) is the probability of (r) successes. From this we can derive
the likelihood function, which is given by:

$$
L(mu; y, alpha) = prod_{i=1}^{n}expleft(y_i lnleft(frac{alphamu_i}{1 +alphamu_i}right)-frac{1}{alpha}ln(1 + alphamu_i) + lnGamma(y_i + frac{1}{alpha})-lnGamma(y_i + 1) - lnGamma(frac{1}{alpha})right)
$$

here we find the likelihood of the expected value, (mu) given the data and
(alpha) which allows for dispersion. Typically, this would be expressed as

a log likelihood, denoted by script L, (mathcal{L}):

$$
mathcal{L}(mu; y, alpha) = sum_{i=1}^{n}y_i lnleft(frac{alphamu_i}{1 + alphamu_i}right)-frac{1}{alpha}ln(1 + alphamu_i) + lnGamma(y_i + frac{1}{alpha})-lnGamma(y_i + 1) - lnGamma(frac{1}{alpha})
$$

which can be expressed in terms of our model by replacing (mu_i) with
(exp(x_i'beta)). Turning to the zero-inflated negative binomial model,
the expression of the likelihood function depends on whether the observed value
is a zero or greater than zero. From the logistic model of (y_i gt 1) versus
(y = 0):

$$
p = frac{1}{1 + e^{-x_i'beta}}
$$

and

$$
1 - p = frac{1}{1 + e^{x_i'beta}}
$$

then

$$
mathcal{L} = left{ begin{array}{ll} sum_{i=1}^{n} left &mbox{if } y_{i} = 0 \
sum_{i=1}^{n} left &mbox{if } y_{i} > 0 end{array} right.
$$

Finally, note that R does not estimate (alpha) but (theta), the
inverse of (alpha).

Now let's build up our model. We are going to use the variables
`child` and `camper` to model the count in the part of negative binomial model and the variable `persons` in the logit part of the model.
We use the `pscl` to run a zero-inflated negative binomial regression. We begin by estimating the model with the variables of interest.

```
m1<-zeroinfl(count~child+camper|persons,data=
zinb,dist="negbin")summary(m1)


##
## Call:
## zeroinfl(formula = count ~ child + camper | persons,
data = zinb,
## dist = "negbin", EM = TRUE)
##
## Pearson residuals:
## Min 1Q Median 3Q Max
## -0.586 -0.462 -0.389 -0.197 18.013
##
## Count model coefficients (negbin with log link):
## Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1.371 0.256 5.35 8.6e-08 ***
## child -1.515 0.196 -7.75 9.4e-15 ***
## camper1 0.879 0.269 3.26 0.0011 **
## Log(theta) -0.985 0.176 -5.60 2.1e-08 ***
##
## Zero-inflation model coefficients (binomial with logit
link):
## Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1.603 0.836 1.92 0.055 .
```

```
## persons -1.666 0.679 -2.45 0.014 *
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Theta = 0.373
## Number of iterations in BFGS optimization: 2
## Log-likelihood: -433 on 6 Df
```

The output looks very much like the output from two OLS regressions in R.

Below the model call, you will find a block of output containing negative
binomial regression coefficients for each of the variables along with
standard errors, z-scores, and p-values for the coefficients. A second block
follows that corresponds to the inflation model. This includes logit
coefficients for predicting excess zeros along with their standard errors,
z-scores, and p-values.

All of the predictors in both the count and inflation

portions of the
model are statistically significant. This model fits the data significantly
better than the null model, i.e., the intercept-only model. To show that
this is the case, we can compare with the current model to a null model
without predictors using chi-squared test on the difference of log
likelihoods.

```
m0<-update(m1, .~1)pchisq(2*(logLik(m1)-logLik(m0)),df=3,lower.tail=FALSE)

## 'log Lik.' 1.28e-13 (df=6)
```

From the output above, we can see that our overall model is statistically significant.

We can get confidence intervals for the parameters and the
exponentiated parameters using bootstrapping. For the negative binomial model, these would
be incident risk ratios, for the zero inflation model, odds

ratios. We use the
`boot` package. First, we get the coefficients from our original model to
use as start values for the model to speed up the time it takes to estimate. Then
we write a short function that takes data and indices as input and returns the
parameters we are interested in. Finally, we pass that
to the `boot` function and do 1200 replicates, using snow to distribute across
four cores. Note that you should adjust the number of cores to whatever your machine
has. Also, for final results, one may wish to increase the number of replications to
help ensure stable results.

```
dput(round(coef(m1,"count"),4))

## structure(c(1.3711, -1.5152, 0.879), .Names = c("(Intercept)",
## "child", "camper1"))

dput(round(coef(m1,"zero"),4))
```

```
## structure(c(1.6028, -1.6663), .Names = c("(Intercept)",
"persons"
## ))

f<-function(data,i) {require(pscl)m<-
zeroinfl(count~child+camper|persons,data=
data,dist="negbin",start=list(count=c(1.3711,-1.5152,0.8
79),zero=c(1.6028,-1.6663)))as.vector(t(do.call(rbind,coef
(summary(m)))))}set.seed(10)(res<-boot(zinb,
f,R=1200,parallel="snow",ncpus=4))

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = zinb, statistic = f, R = 1200, parallel =
"snow",
## ncpus = 4)
##
##
## Bootstrap Statistics :
## original bias std. error
## t1* 1.3711 -0.083023 0.39403
```

t2* 0.2561 -0.002622 0.03191
t3* -1.5153 -0.061487 0.26892
t4* 0.1956 0.006034 0.02027
t5* 0.8791 0.091431 0.47124
t6* 0.2693 0.001873 0.01998
t7* -0.9854 0.080120 0.21896
t8* 0.1760 0.002577 0.01689
t9* 1.6031 0.473597 1.59331
t10* 0.8365 3.767327 15.65780
t11* -1.6666 -0.462364 1.56789
t12* 0.6793 3.771994 15.69675

The results are alternating parameter estimates and standard
errors. That is, the first row has the first parameter estimate
from our model. The second has the standard error for the
first parameter. The third column contains the bootstrapped
standard errors, which are considerably larger than those estimated
by `zeroinfl`.

Now we can get the confidence intervals for all the parameters.
We start on the original scale with percentile and bias adjusted CIs.
We also compare these results with the regular confidence intervals
based on the standard errors.

```
## basic parameter estimates with percentile and bias adjusted CIsparms<-t(sapply(c(1,3,5,9,11),function(i) {out<-boot.ci(res,index=c(i, i+1),type=c("perc","bca"))with(out,c(Est= t0,pLL= percent,pUL= percent,bcaLL= bca,bcaUL= bca))}))## add row namesrow.names(parms)<-names(coef(m1))## print resultsparms

## Est pLL pUL bcaLL bcaUL
## count_(Intercept) 1.3711 0.5676 2.0620 0.7226 2.2923
## count_child -1.5153 -2.1382 -1.0887 -2.0175 -0.9593
## count_camper1 0.8791 0.0431 1.8331 -0.2016 1.6669
## zero_(Intercept) 1.6031 0.4344 8.2380 0.0282 3.5197
## zero_persons -1.6666 -8.5436 -1.1002 -7.8329 -1.0781

## compare with normal based
```

```
approximationconfint(m1)

## 2.5 % 97.5 %
## count_(Intercept) 0.86911 1.8731
## count_child -1.89860 -1.1319
## count_camper1 0.35127 1.4068
## zero_(Intercept) -0.03636 3.2419
## zero_persons -2.99701 -0.3355
```

The bootstrapped confidence intervals are considerably wider than the normal based approximation. The bootstrapped CIs are more consistent with the CIs from Stata when using robust standard errors.

Now we can estimate the incident risk ratio (IRR) for the negative binomial model and odds ratio (OR) for the logistic (zero inflation) model. This is done using almost identical code as before, but passing a transformation function to the `h` argument of `boot.ci`, in this case, `exp` to exponentiate.

```
## exponentiated parameter estimates with percentile
```

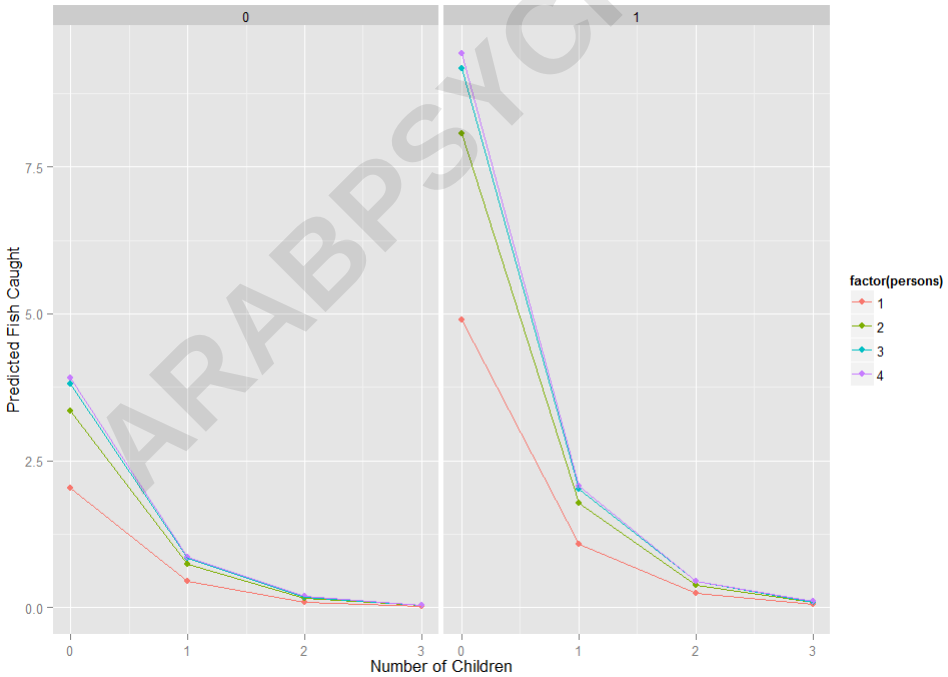
and bias adjusted CIsexpparms<-t(sapply(c(1,3,5,9,11),function(i) {out<-boot.ci(res,index=c(i, i+1),type=c("perc","bca"),h=exp)with(out,c(Est= t0,pLL= percent,pUL= percent,bcaLL= bca,bcaUL= bca))}))## add row namesrow.names(expparms)<-names(coef(m1))## print resultsexpparms

Est pLL pUL bcaLL bcaUL
count_(Intercept) 3.9395 1.7641 7.8615 2.0599 9.8981
count_child 0.2198 0.1179 0.3367 0.1330 0.3832
count_camper1 2.4086 1.0441 6.2534 0.8175 5.2958
zero_(Intercept) 4.9686 1.5441 3781.9642 1.0286 33.7757
zero_persons 0.1889 0.0002 0.3328 0.0004 0.3402

To better understand our model, we can compute the expected number of fish caught for different combinations of our predictors. In fact, since we are working with essentially categorical predictors, we can compute the expected values for all combinations using the `expand.grid` function to create

all combinations and then the `predict` function to do it. Finally we
create a graph.

```
newdata1<-
expand.grid(0:3,factor(0:1),1:4)colnames(newdata1)<-
c("child","camper","persons")newdata1$phat<-
predict(m1, newdata1)ggplot(newdata1,aes(x= child,y=
phat,colour=factor(persons)))+geom_point()+geom_line
()+facet_wrap(~camper)+labs(x="Number of
Children",y="Predicted Fish Caught")
```



Things to consider

Here are some issues that you may want to consider in the course of your research analysis.

References