What is the relationship between Zero-Truncated Negative Binomial distribution and R data analysis?

Authored by

stats writer

June 29, 2024

The Zero-Truncated Negative Binomial distribution is a probability distribution frequently used in R data analysis. This distribution is particularly useful in situations where the data has a skewed distribution and a large number of zero values. In R data analysis, the Zero-Truncated Negative Binomial distribution is often used to model count data, such as the number of customers, accidents, or failures in a given time period. This distribution allows for the estimation of the probability of non-zero values, which is a crucial aspect in many data analysis techniques. Therefore, the Zero-Truncated Negative Binomial distribution plays a significant role in accurately analyzing and interpreting data in R.

Zero-Truncated Negative Binomial | R Data Analysis Examples

Zero-truncated negative binomial regression is used to model count data for which the value zero cannot occur and for which over dispersion exists.

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 If you do not have a package installed, run: `install.packages("packagename")`, or if you see the version is out of date, run: `update.packages()`.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(foreign)
require(ggplot2)
require(VGAM)
require(boot)
```

```
Version info: Code for this page was tested in R Under development (unstable)
(2012-11-16 r61126)
On: 2012-12-15
With: boot 1.3-7; VGAM 0.9-0; ggplot2 0.9.3; foreign 0.8-51; knitr 0.9
```

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 verification, verification of assumptions, model diagnostics and potential follow-up analyses.

Examples of zero-truncated negative binomial

Example 1. A study of length of hospital stay, in days, as a function of age, kind of health insurance and whether or not the patient died while in the hospital. Length of hospital stay is recorded as a minimum of at least one day.

Example 2. A study of the number of journal articles published by tenured faculty as a function of discipline (fine arts, science, social science, humanities, medical, etc). To get tenure faculty must

publish, therefore,
there are no tenured faculty with zero publications.

Example 3. A study by the county traffic court on the number of tickets received by teenagers
as predicted by school performance, amount of driver training and gender. Only individuals
who have received at least one citation are in the traffic court files.

Description of the data

Let's pursue Example 1 from above.

We have a *hypothetical* data file, `ztp.dta` with 1,493 observations.
The length of hospital stay variable is `stay`.
The variable `age` gives the age group from 1 to 9 which will be treated as
interval in this example. The variables `hmo` and `died` are binary indicator variables
for HMO insured patients and patients who died while in the hospital, respectively.

Let's look at the data. These are the same data as were used in the

ztp example.

We import the Stata dataset using the `foreign` package.

```
dat <- read.dta("https://stats.idre.ucla.edu/stat/data/ztp.dta")

dat <- within(dat, {
hmo <- factor(hmo)
died <- factor(died)
})

summary(dat)

## stay age hmo died
## Min. : 1.00 Min. :1.00 0:1254 0:981
## 1st Qu.: 4.00 1st Qu.:4.00 1: 239 1:512
## Median : 8.00 Median :5.00
## Mean : 9.73 Mean :5.23
## 3rd Qu.:13.00 3rd Qu.:6.00
## Max. :74.00 Max. :9.00
```

Now let's look at some graphs of the data conditional on various combinations of the variables to get a sense of how the variables work together.

We will use the `ggplot2` package. First we can look at histograms of
`stay` broken down by `hmo` on the rows and `died` on the columns.
We also include the marginal distributions, thus the lower right corner represents
the overall histogram. We use a log base 10 scale to approximate the canonical link function of
the negative binomial distribution (natural logarithm).

```
ggplot(dat, aes(stay)) + geom_histogram() +
scale_x_log10() + facet_grid(hmo ~
died, 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.

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.

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

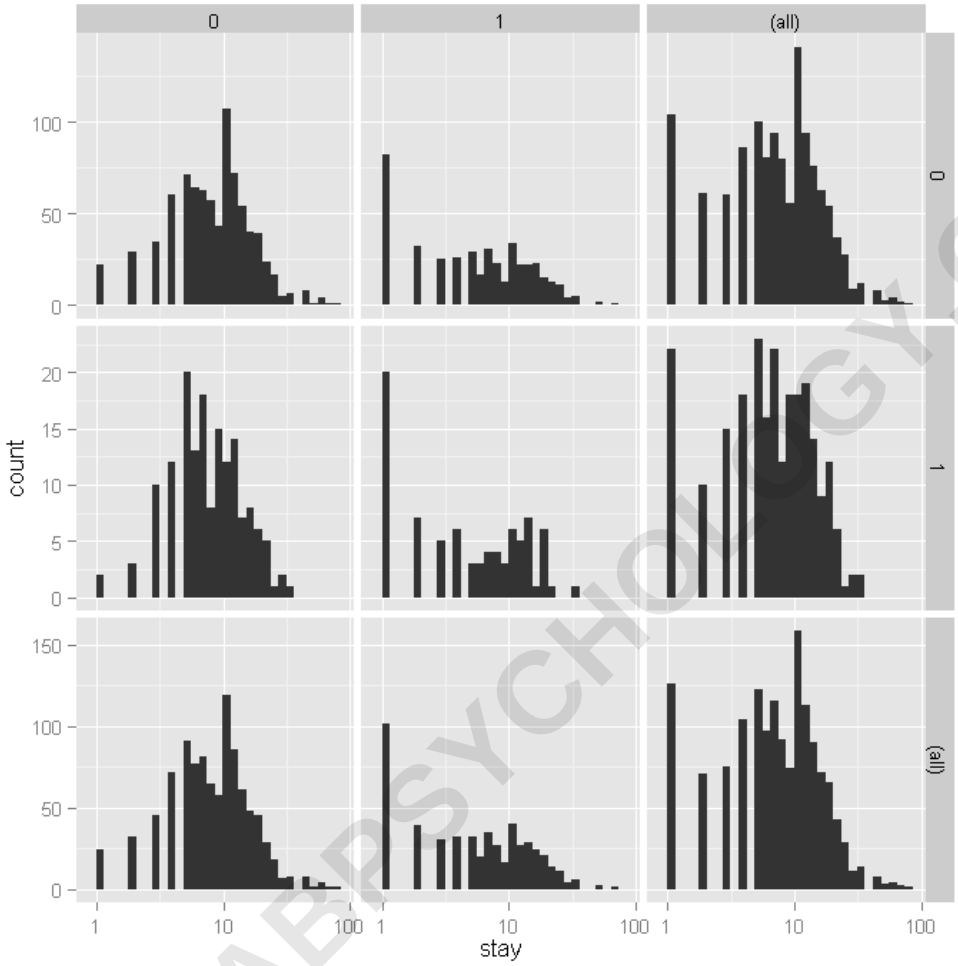
stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust
this.

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

stat_bin: binwidth defaulted to range/30. Use

'binwidth = x' to adjust

this.



From the histograms, it looks like the density of the distribution,

does vary across levels of `hmo` and `died`, with

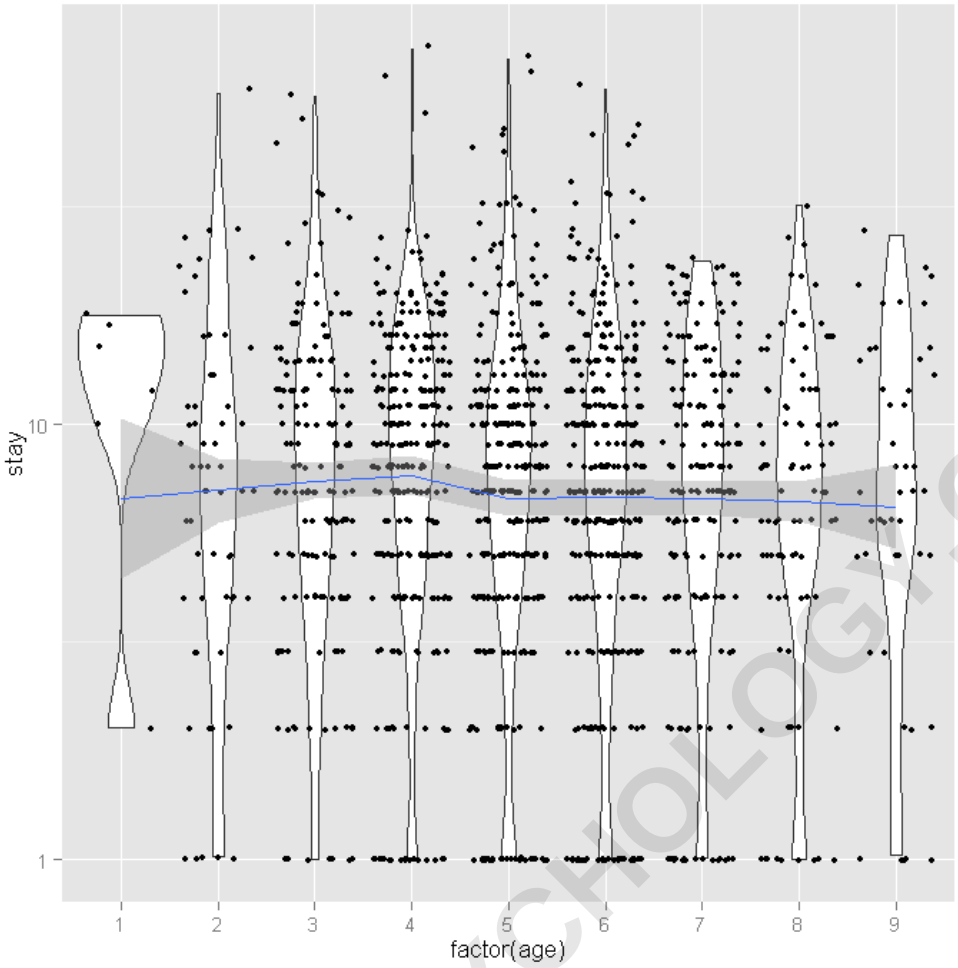
shorter stays for those in HMOs (1) and shorter for those who did die,

including what seems to be an inflated number of 1 day

stays.

To examine how `stay` varies across age groups, we can use conditional
violin plots which show a kernel density estimate of the distribution of stay
mirrored (hence the violin) and conditional on each age group. To further understand
the raw data going into each density estimate, we add raw data on top of the violin plots
with a small amount of random noise (jitter) to alleviate over plotting. Finally, to get a
sense of the overall trend, we add a locally weighted regression line.

```
ggplot(dat, aes(factor(age), stay)) +
geom_violin() +
geom_jitter(size=1.5) +
scale_y_log10() +
stat_smooth(aes(x = age, y = stay, group=1), method="loess")
```



The distribution of length of stay does not seem to vary much across age groups.

This observation from the raw data is corroborated by the relatively flat loess line.

Finally let's look at the proportion of people who lived or died across age groups

by whether or not they are in HMOs.

```
ggplot(dat, aes(age, fill=died)) +
```

geom_histogram(binwidth=.5, position="fill") +
facet_grid(hmo ~ ., margins=TRUE)

For the lowest ages, a smaller proportion of people in HMOs died, but
for higher ages, there does not seem to be a huge difference, with a
slightly higher proportion in HMOs dying if anything. Overall, as

age group increases, the proportion of those dying increases, as expected.

Analysis methods you might consider

Below is a list of some analysis methods you may have encountered.
Some of the methods listed are quite reasonable while others have either fallen out of favor or
have limitations.

Zero-truncated negative binomial regression

To fit the zero-truncated negative binomial model, we use the `vglm` function
in the `VGAM` package. This function fits a very flexible class of models
called vector generalized linear models to a wide range of assumed distributions.
In our case, we believe the data come from the negative binomial distribution,
but without zeros. Thus the values are strictly positive poisson,
for which we use the positive negative binomial family via the

`posnegbinomial` function passed to `vglm`.

```
m1 <- vglm(stay ~ age + hmo + died, family = posnegbinomial(), data = dat)
```

```
## "head(w)"
##
## 1
## 1
## 1
## 1
## 1
## 1
## "head(y)"
##
## 1 4
## 2 9
## 3 3
## 4 9
## 5 1
## 6 4
```

```
summary(m1)
```

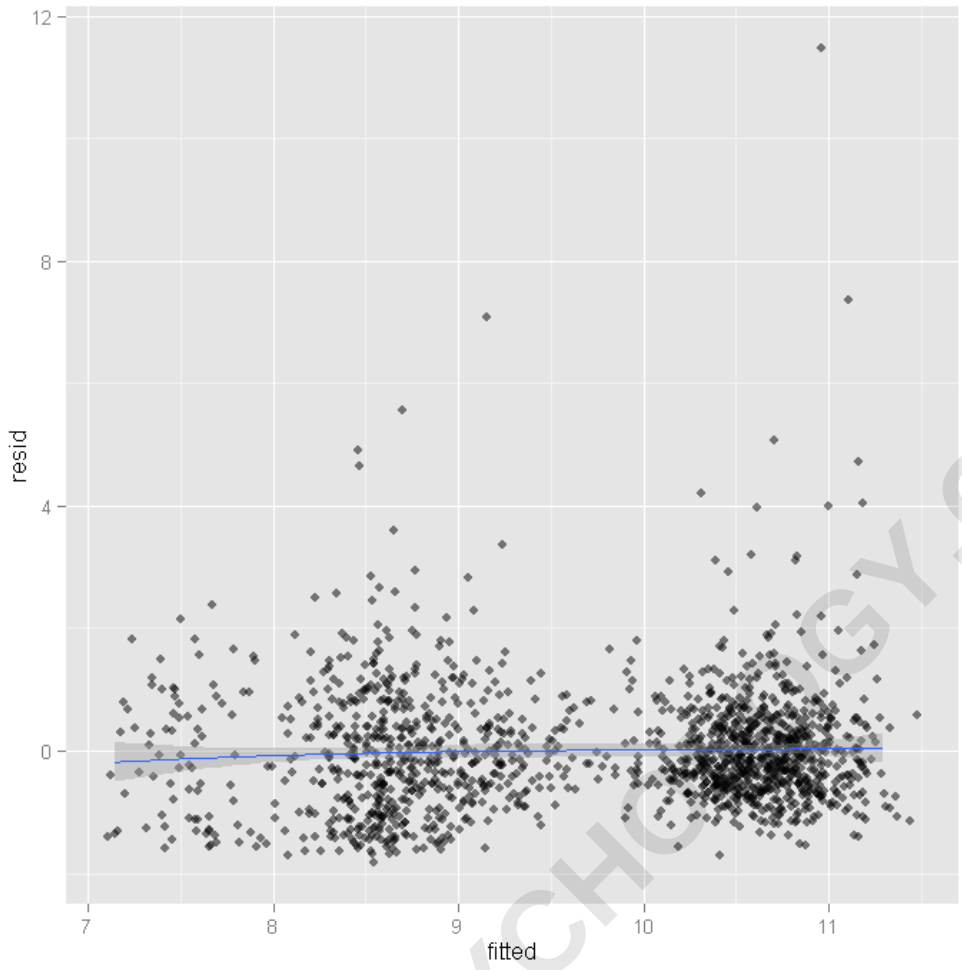
```
##
## Call:
## vglm(formula = stay ~ age + hmo + died, family =
posnegbinomial(),
## data = dat)
##
## Pearson Residuals:
## Min 1Q Median 3Q Max
## log(munb) -1.4 -0.70 -0.23 0.45 9.8
## log(size) -14.1 -0.27 0.45 0.76 1.0
##
## Coefficients:
## Estimate Std. Error z value
## (Intercept):1 2.408 0.072 33.6
## (Intercept):2 0.569 0.055 10.4
## age -0.016 0.013 -1.2
## hmo1 -0.147 0.059 -2.5
## died1 -0.218 0.046 -4.7
##
## Number of linear predictors: 2
##
## Names of linear predictors: log(munb), log(size)
##
## Dispersion Parameter for posnegbinomial family: 1
```

```
##
## Log-likelihood: -4755 on 2981 degrees of freedom
##
## Number of iterations: 4
```

The output looks very much like the output from an OLS regression:

Now let's look at a plot of the residuals versus fitted values. We add random horizontal
noise as well as 50 percent transparency to alleviate over plotting and better see where
most residuals fall. Note that these residuals are for the mean prediction.

```
output <- data.frame(resid = resid(m1), fitted = fitted(m1))
ggplot(output, aes(fitted, resid)) + geom_jitter(position = position_jitter(width = 0.25),
alpha = 0.5) + stat_smooth(method = "loess")
```

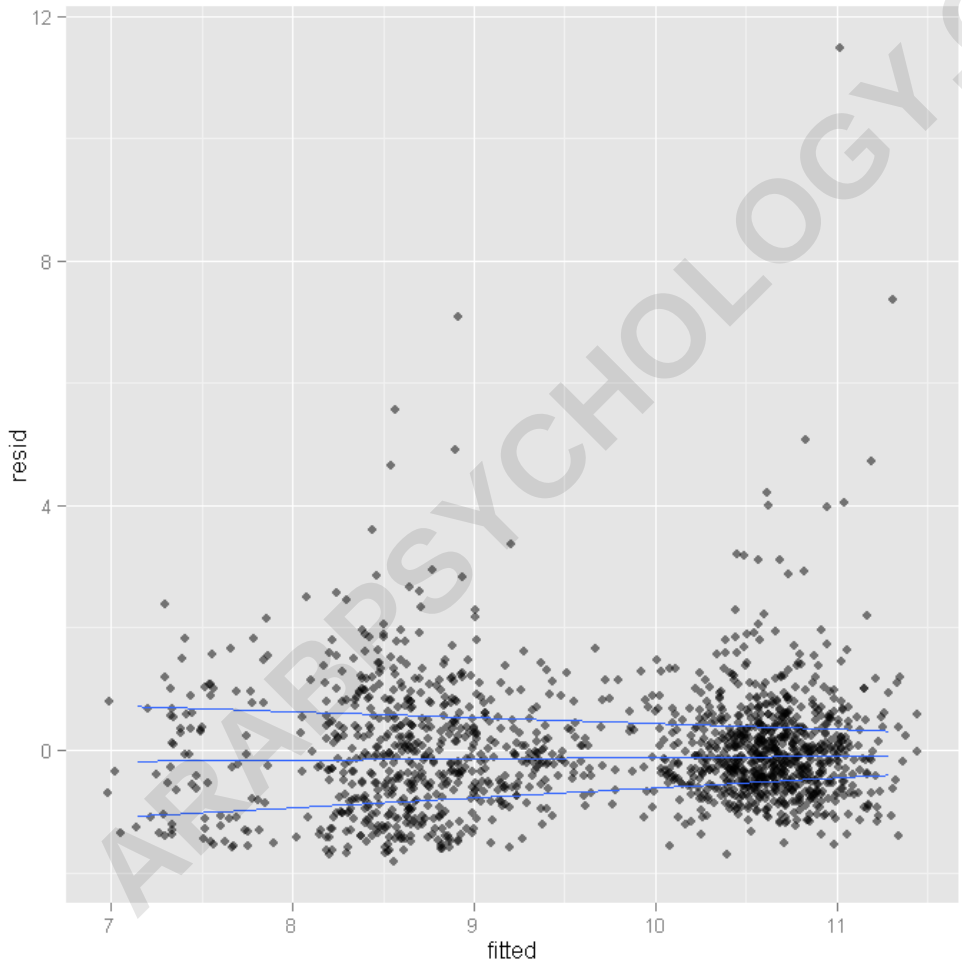


The mean is around zero across all the fitted levels it looks like. However,
there are some values that look rather extreme. To see if these have much influence,
we can fit lines using quantile regression, these lines represent the 75th, 50th, and 25th
percentiles.

ggplot(output, aes(fitted, resid)) +

```
geom_jitter(position=position_jitter(width=.25),
alpha=.5) +
stat_quantile(method="rq")

## Smoothing formula not specified. Using: y ~ x
```

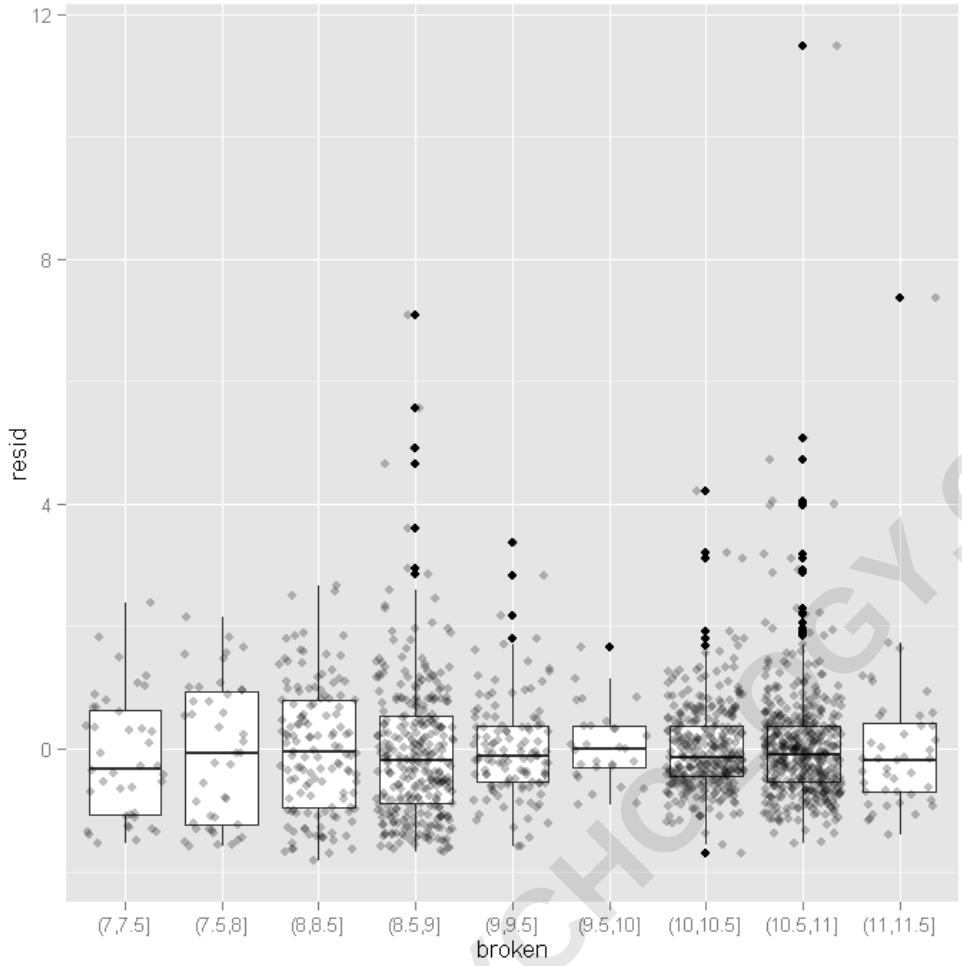


Here we see the spread narrowing at higher levels. Let's cut the data into intervals and check box plots for each. We will get

```
the breaks
from the algorithm for a histogram.

output <- within(output, {
broken <- cut(fitted, hist(fitted, plot=FALSE)$breaks)
})

ggplot(output, aes(broken, resid)) +
geom_boxplot() +
geom_jitter(alpha=.25)
```



The variance seems to decrease slightly at higher fitted values, except for the
very last category (this shown by the hinges of the boxplots).

To test whether we need to estimate over dispersion, we could fit a zero-truncated
Poisson model and compare the two.

m2 <- vglm(formula = stay ~ age + hmo + died, family =

```
pospoisson(), data = dat)

## change in deviance
(dLL <- 2 * (logLik(m1) - logLik(m2)))

## 4307

## p-value, 1 df---the overdispersion parameterpchisq(dLL, df = 1, lower.tail = FALSE)

## 0
```

Based on this, we would conclude that the negative binomial model is
a better fit to the data.

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.
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),3))

## structure(c(2.408, 0.569, -0.016, -0.147, -0.218),
.Names = c("(Intercept):1",
## "(Intercept):2", "age", "hmo1", "died1"))

f <- function(data, i, newdata) {
require(VGAM)
m <- vglm(formula = stay ~ age + hmo + died, family =
posnegbinomial(),
data = data, coefstart = c(2.408, 0.569, -0.016, -0.147,
-0.218))
```

```
mparams <- as.vector(t(coef(summary(m))))
yhat <- predict(m, newdata, type = "response")
return(c(mparams, yhat))
}

## newdata for prediction
newdata <- expand.grid(age = 1:9, hmo = factor(0:1),
died = factor(0:1))
newdata$yhat <- predict(m1, newdata, type =
"response")

set.seed(10)
res <- boot(dat, f, R = 1200, newdata = newdata, parallel
= "snow", ncpus = 4)

## "head(w)"
##
## 1
## 1
## 1
## 1
## 1
## 1
## "head(y)"
```

```
##
## 1 4
## 2 9
## 3 3
## 4 9
## 5 1
## 6 4
```

The results are alternating parameter estimates and standard
errors for the parameters (the first 10).
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.

Now we can get the confidence intervals for all the parameters.
We start on the original scale with percentile and basic bootstrap CIs.

```
## basic parameter estimates with percentile and bias
```

```
adjusted CIs
parms <- t(sapply(c(1, 3, 5, 7, 9), function(i) {
out <- boot.ci(res, index = c(i, i + 1), type = c("perc",
"basic"))
with(out, c(Est = t0, pLL = percent, pUL = percent,
basicLL = basic, basicLL = basic))
}))

## add row namesrow.names(parms) <-
names(coef(m1))
## print results
parms

## Est pLL pUL basicLL basicLL
## (Intercept):1 2.40833 2.26288 2.55285 2.26381 2.55377
## (Intercept):2 0.56864 0.43812 0.70414 0.43314 0.69916
## age -0.01569 -0.04233 0.01089 -0.04228 0.01095
## hmo1 -0.14706 -0.26276 -0.03931 -0.25481 -0.03135
## died1 -0.21777 -0.32846 -0.11476 -0.32078 -0.10708
```

The bootstrapped confidence intervals are wider than would be expected using a
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.
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 CIs
expparms <- t(sapply(c(1, 3, 5, 7, 9), function(i) {
out <- boot.ci(res, index = c(i, i + 1), type = c("perc", "basic"), h = exp)
with(out, c(Est = t0, pLL = percent, pUL = percent, basicLL = basic, basicLL = basic))
}))

## add row namesrow.names(expparms) <- names(coef(m1))
## print results
expparms

##                Est     pLL     pUL basicLL basicLL
## (Intercept):1 11.1154 9.6107 12.8436 9.3871 12.6200
```

(Intercept):2 1.7659 1.5498 2.0221 1.5096 1.9819
age 0.9844 0.9585 1.0110 0.9579 1.0103
hmo1 0.8632 0.7689 0.9615 0.7650 0.9576
died1 0.8043 0.7200 0.8916 0.7170 0.8886

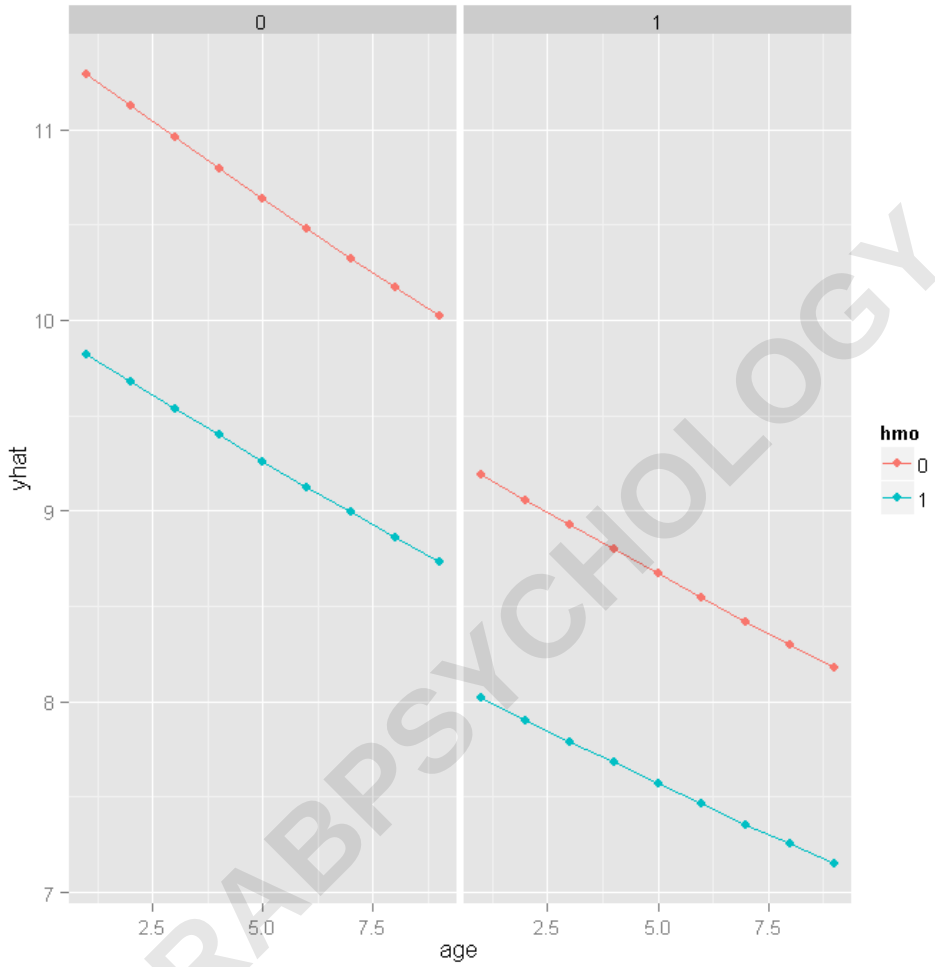
The results are consistent with what we initially viewed graphically,
`age` does not have a significant effect, but `hmo` and `died` both do.
In order to better understand our results and model, let's plot some predicted values.
Because all of our predictors were categorical (`hmo` and `died`)
or had a small number of unique values (`age`) we will get predicted values for
all possible combinations. This was actually done earlier
when we bootstrapped the parameter estimates by creating a new data set
using the `expand.grid` function, then estimating the predicted values
using the `predict` function. Now we can plot that data.

```
ggplot(newdata, aes(x = age, y = yhat, colour = hmo)) +
```

geom_point() +

geom_line() +

facet_wrap(~ died)



If we really wanted to compare the predicted values, we could bootstrap

confidence intervals around the predicted estimates. These confidence

intervals are not for the predicted value themselves, but

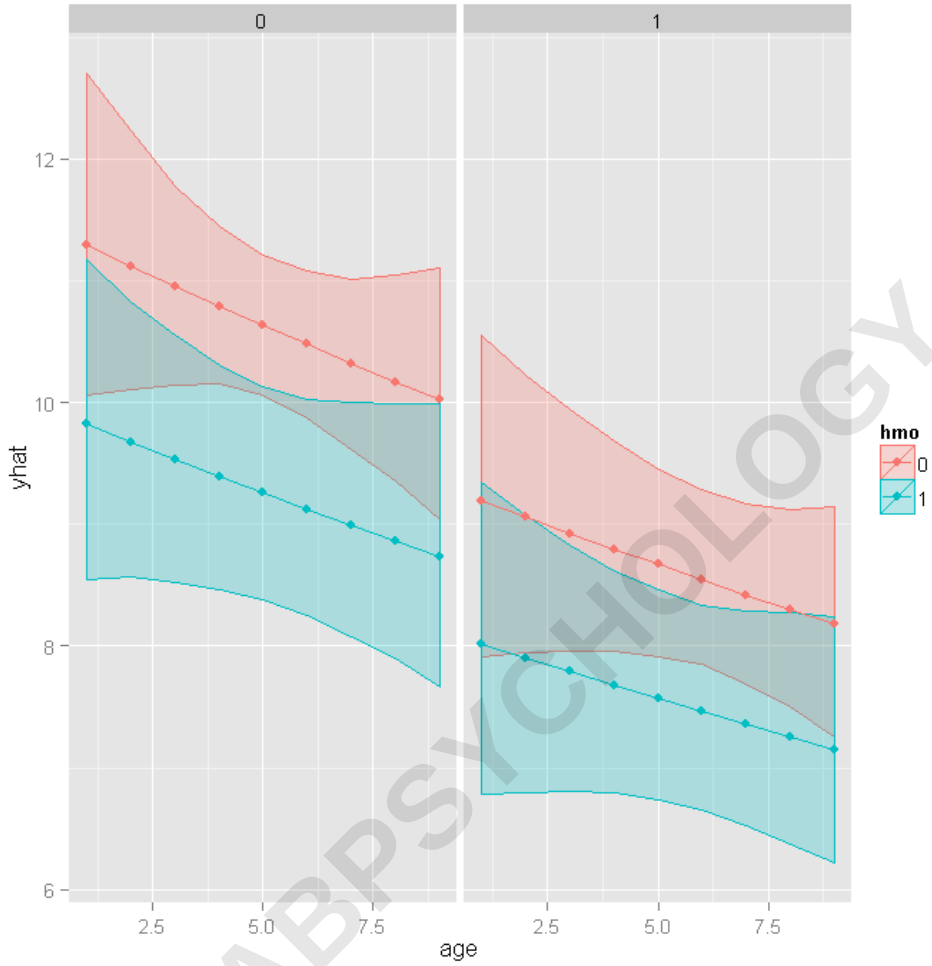
that that is the
mean predicted value (i.e., for the estimate, not a new individual).
Because fitting these models is slow, we included the predicted values
earlier when we bootstrapped the model parameters. We will go back to
the bootstrap output now and get the confidence intervals for the
predicted values.

```
## get the bootstrapped percentile CIs
yhat <- t(sapply(10 + (1:nrow(newdata)), function(i) {
out <- boot.ci(res, index = i, type = c("perc"))
with(out, c(Est = t0, pLL = percent, pUL = percent))
}))

## merge CIs with predicted values
newdata <- cbind(newdata, yhat)

## graph with CIsggplot(newdata, aes(x = age, y = yhat,
colour = hmo, fill = hmo)) +
geom_ribbon(aes(ymin = pLL, ymax = pUL), alpha = .25)
+
geom_point() +
```

geom_line() + facet_wrap(~ died)



Things to consider

See Also

References