Bayesian count data regression models: optional practical exercises

Christopher Jackson

1 Background: Poisson regression

Count data are common in epidemiology, e.g. where the outcome y_i is the number of disease cases by area i, or the number of hospital admissions per month i. The **Poisson** distribution $y_i \sim Pois(\mu_i)$: forms the basis for a wide range of models for count data. For example:

- Simple Poisson regression $\log(\mu_i) = \alpha + \beta x_i$ on a covariate x_i
- **Negative Binomial** mean μ_i, dispersion parameter r
  Used when data are *overdispersed*, so that the variance is greater than the mean.
  Tends to Poisson(μ_i) as $r \to \infty$.
- **Zero-inflated** and similar *mixture* models.
  - An uncertain proportion p of individuals have *zero risk* of events.
  - The remaining proportion $1 - p$ of individuals follow, e.g $Pois(\mu)$
  - The parameters p and μ are jointly estimated.
  - Regression models could also be defined to describe either p and μ in terms of predictors.

Hierarchical models (see later): can represent more general sources of variation.

The Poisson model and its extensions can be easily implemented in JAGS, using code such as the following: Note the use of the `log` link function, in a similar way to the `logit` link in logistic regression.

```
for (i in 1:n) {
  y[i]         ~ dpois(mu[i])
  log(mu[i]) <- alpha + beta*x[i]
}
```

2 Poisson regression - exercises

Key learning points

- Case study of Bayesian count data modelling
- Demo of Poisson, negative binomial and zero-inflated Poisson model in JAGS
- Interpreting the parameters of fitted count data models
- Comparing models by examining posterior distributions of parameters

In the dataset available in the file `lip_cancer.csv`, `Y` gives the number of cases of lip cancer in each county of Scotland during the 6 years from 1975 to 1980.

The variable `E` is the number of cases of lip cancer *expected* in each county due to the age distribution of people in that county, calculated from the overall age-specific cancer rates in Scotland.

The covariate `X` is the percentage of the population in outdoor occupations (defined as agriculture, fishing or forestry) in the county, who are thought to face a higher risk of lip cancer due to exposure to sunlight.

```
lipsdf <- read.csv("lip_cancer.csv")
plot(lipsdf$Y, lipsdf$X)
```

We will use count data regression models to estimate the relative rate of lip cancer associated with a increase of 10% in the percentage of people in outdoor occupations. The JAGS code for a basic Poisson regression model is shown below. This includes `log(E)` as an "offset" term in the regression (i.e. a covariate with a known coefficient of 1 - since a unit change in `E` will give a unit change in `mu`), a linear association with `X`, and vague prior distributions.

```
lips_model <- "
model {
  for (i in 1:n) {
    Y[i]        ~ dpois(mu[i])
    log(mu[i]) <- log(E[i]) + alpha + beta*X[i]/10
    RR[i]      <- exp(alpha + beta*X[i]/10)
  }
  alpha        ~ dnorm(0, 0.00001)
  beta         ~ dnorm(0, 0.00001)
  rrX         <- exp(beta)
}
"
```

(a) Fit the Poisson model.

(b) Compare the relative risks with those from a negative binomial model, which is more flexible than the Poisson.

The JAGS code should be adapted to replace the Poisson regression by a negative binomial regression. The negative binomial model in JAGS is parameterised in terms of a probability parameter `p` and a dispersion parameter `r`. The probability parameter can be transformed into a mean parameter `mu` to allow a regression model to be constructed, as follows.

```
Y[i] ~ dnegbin(p[i], r)
p[i] <- r / (mu[i] + r)
```

The dispersion parameter `r` is restricted to be a positive integer in JAGS. The following prior can be used, based on a latent continuous variable `1/sqrt(invr)` that is rounded to the nearest integer above. This prior has a mode around `invr=0` (the special case of the Poisson model) and places lower prior mass on increasingly greater amounts of overdispersion. [1]

```
r <- trunc(1/sqrt(invr)) + 1
invr ~ dexp(1)
```

(c) Implement a zero-inflated Poisson model, and compare with the previous results.

The zero-inflated Poisson model defines the Poisson mean to be zero for a subgroup of individuals, hence the outcome `Y` is fixed at zero for these individuals. The remaining individuals follow a Poisson regression model with mean `mu[i]`.

This is implemented by defining a binary variable `group[i]`, that is 0 if individual `i` is in the first group (with fixed outcome of `Y=0`), and `group[i]` is 1 if individual `i` is in the second group (following the standard Poisson regression model). People with observed `Y{i] > 0`, are known to be in the second group. However it is unknown which group people with `Y[i]=0` belong to - note the standard Poisson can take a value of 0 by chance.

The prior probability that any individual (whose data we have not observed) belongs to group 2 is `p`.

Hence this model can be implemented in JAGS by substituting the following code in the standard Poisson model.

```
Y[i] ~  dpois(group[i]*mu[i])
group[i] ~ dbern(p)
```

[1] https://github.com/stan-dev/rstanarm/issues/275

A prior distribution for `p` should also be supplied. Examine the posterior and check that it makes sense. Why might we have known in advance that this model was probably not very useful in this example?

Initial values for the `group` variable will need to be provided as well - these must be 1 for people with observed $Y > 1$, but can be 0 or 1 for people with $Y = 0$. The following is sufficient for this exercise:

```
ini <- list(list(alpha=0, beta=0, p=0.1, group=ifelse(dat$Y==0, 1, 1)),
            list(alpha=1, beta=1, p=0.2, group=ifelse(dat$Y==0, 0, 1)))
```

Solutions: Poisson regression

We will implement all three models, (a), (b) and (c), and compare their results at the end.

2.0.1 (a) Standard Poisson regression

```
library(rjags)
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.1
```

```
## Loaded modules: basemod,bugs
```

```
library(bayesplot)
```

```
## This is bayesplot version 1.10.0
```

```
## - Online documentation and vignettes at mc-stan.org/bayesplot
```

```
## - bayesplot theme set to bayesplot::theme_default()
```

```
##    * Does _not_ affect other ggplot2 plots
```

```
##    * See ?bayesplot_theme_set for details on theme setting
```

```
library(posterior)
```

```
## This is posterior version 1.4.1
```

```
##
## Attaching package: 'posterior'
```

```
## The following object is masked from 'package:bayesplot':
##
##     rhat
```

```
## The following objects are masked from 'package:stats':
##
##     mad, sd, var
```

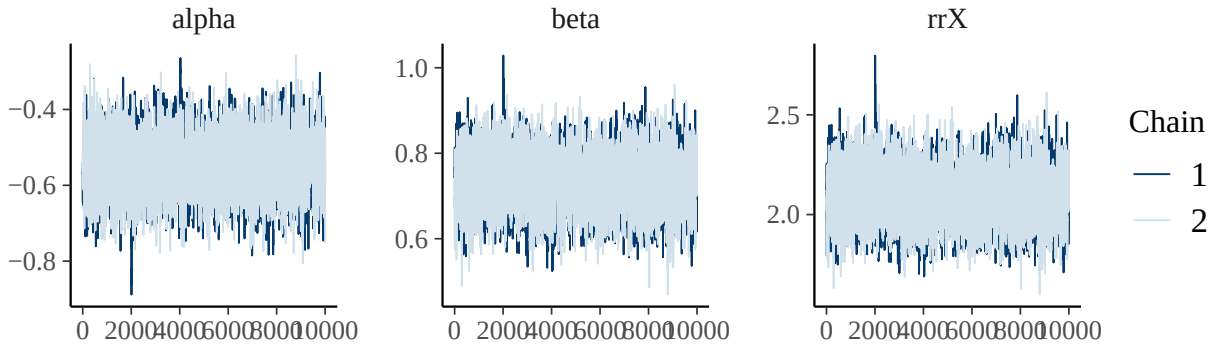
```
## The following objects are masked from 'package:base':
##
##     %in%, match
```

```
lipsdf <- read.csv("lip_cancer.csv")
dat <- c(as.list(lipsdf), list(n=length(lipsdf$X)))
lips_model <- "
model {
  for (i in 1:n) {
    Y[i]        ~ dpois(mu[i])
    log(mu[i]) <- log(E[i]) + alpha + beta*X[i]/10
    RR[i]      <- exp(alpha + beta*X[i]/10)
  }
  alpha        ~ dnorm(0, 0.00001)
  beta         ~ dnorm(0, 0.00001)
  rrX         <- exp(beta)
}
"
ini <- list(list(alpha=0, beta=0),
            list(alpha=1, beta=1))
```

```
lips.jag <- jags.model(textConnection(lips_model), data=dat, inits=ini, n.chains=2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 56
##    Unobserved stochastic nodes: 2
##    Total graph size: 358
##
## Initializing model
```

```
update(lips.jag, 1000)
sam <- coda.samples(lips.jag, c("alpha","beta", "rrX"), n.iter=10000)
mcmc_trace(sam)
```




```
summary(sam)
```

```
##
## Iterations = 2001:12000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD  Naive SE Time-series SE
## alpha -0.5431 0.07021 0.0004965       0.001272
## beta   0.7365 0.05980 0.0004229       0.001063
## rrX    2.0924 0.12498 0.0008837       0.002223
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%     75%   97.5%
## alpha -0.6813 -0.5904 -0.5423 -0.4950 -0.4081
## beta   0.6176  0.6965  0.7372  0.7774  0.8515
## rrX    1.8545  2.0068  2.0901  2.1758  2.3431
```

2.0.2 (b) Negative Binomial regression.

Remember to supply initial values for the new parameter r (actually placed on the parameter `invr` in the JAGS code).

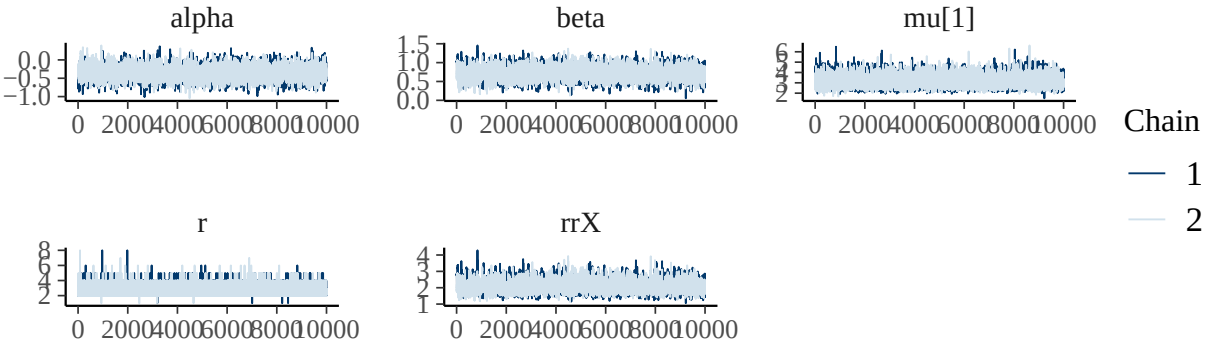
```
lips_nb_model <- "
model {
  for (i in 1:n) {
    Y[i]       ~ dnegbin(p[i], r)
    p[i]       <- r / (mu[i] + r)
    log(mu[i]) <- log(E[i]) + alpha + beta*X[i]/10
    RR[i]      <- exp(alpha + beta*X[i]/10)
  }
  alpha        ~ dnorm(0, 0.00001)
  beta         ~ dnorm(0, 0.00001)
  rrX         <- exp(beta)

  r <- trunc(1/sqrt(invr)) + 1
  invr ~ dexp(1)
}
"

ini <- list(list(alpha=0, beta=0, invr=1),
            list(alpha=1, beta=1, invr=0.1))
lipsnb.jag <- jags.model(textConnection(lips_nb_model), data=dat, inits=ini, n.chains=2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 56
##    Unobserved stochastic nodes: 3
##    Total graph size: 474
##
## Initializing model
```

```
update(lipsnb.jag, 1000)
samnb <- coda.samples(lipsnb.jag, c("alpha","beta", "rrX", "r", "mu[1]"), n.iter=10000)
mcmc_trace(samnb)
```




```
summary(samnb)
```

```
##
## Iterations = 2001:12000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
```

```
##    plus standard error of the mean:
##
##          Mean     SD Naive SE Time-series SE
## alpha -0.3522 0.1774 0.001254       0.003515
## beta   0.7263 0.1643 0.001162       0.003303
## mu[1]  3.1863 0.5108 0.003612       0.006232
## r      2.5078 0.6594 0.004663       0.007683
## rrX    2.0957 0.3487 0.002466       0.006953
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%     75%   97.5%
## alpha -0.6937 -0.4721 -0.3541 -0.2349 0.00249
## beta   0.4031  0.6172  0.7229  0.8360 1.05383
## mu[1]  2.3385  2.8285  3.1315  3.4871 4.33241
## r      2.0000  2.0000  2.0000  3.0000 4.00000
## rrX    1.4965  1.8537  2.0604  2.3072 2.86861
```

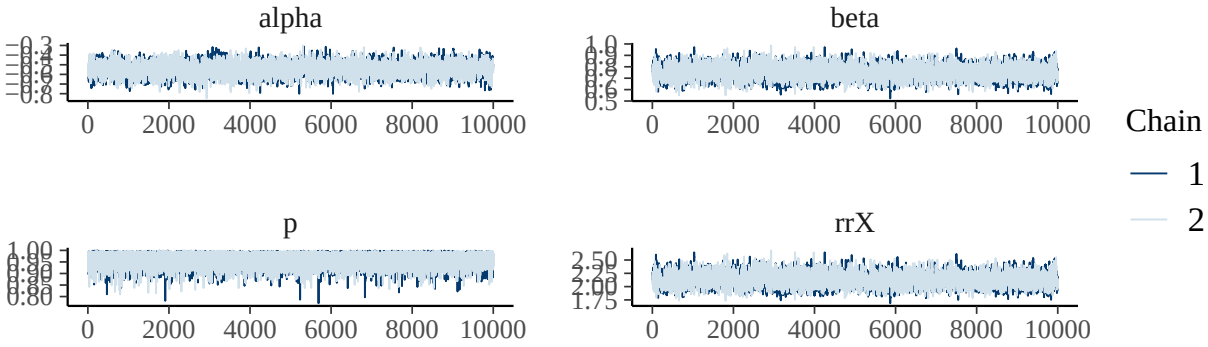
2.0.3 (c) Zero-inflated Poisson model

```
lips_zip_model <- "
model {
  for (i in 1:n) {
    Y[i] ~  dpois(group[i]*mu[i])
    group[i] ~ dbern(p)
    log(mu[i]) <- log(E[i]) + alpha + beta*X[i]/10
    RR[i]      <- exp(alpha + beta*X[i]/10)
  }
  alpha        ~ dnorm(0, 0.00001)
  beta         ~ dnorm(0, 0.00001)
  p ~ dunif(0,1)
  rrX         <- exp(beta)
}
"

ini <- list(list(alpha=0, beta=0, p=0.1, group=ifelse(dat$Y==0, 1, 1)),
            list(alpha=1, beta=1, p=0.2, group=ifelse(dat$Y==0, 0, 1)))
lipszip.jag <- jags.model(textConnection(lips_zip_model),
                          data=dat, inits=ini, n.chains=2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 56
##    Unobserved stochastic nodes: 59
##    Total graph size: 472
##
## Initializing model
```

```
update(lipszip.jag, 1000)
samzip <- coda.samples(lipszip.jag, c("alpha","beta", "rrX", "p"), n.iter=10000)
mcmc_trace(samzip)
```



Comparing median and 90% credible intervals for all relative rate estimates side by side, using tidyverse-style R code.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
rr_pois <- summary(as_draws(sam)) %>%
  filter(variable=="rrX") %>%
  mutate(model="Poisson")
rr_nb  <-  summary(as_draws(samnb)) %>%
  filter(variable=="rrX") %>%
  mutate(model="Negative binomial")
rr_zip <- summary(as_draws(samzip)) %>%
  filter(variable=="rrX") %>%
  mutate(model="Zero-inflated Poisson")
rbind(rr_pois, rr_nb, rr_zip) %>%
  select(model, median, q5, q95)
```

```
## # A tibble: 3 x 4
##   model                median    q5   q95
##   <chr>                 <num> <num> <num>
## 1 Poisson                2.09  1.89  2.30
## 2 Negative binomial      2.06  1.58  2.72
## 3 Zero-inflated Poisson  2.13  1.93  2.35
```

The estimates are very similar under the Poisson and zero-inflated Poisson model, which would be expected, as there are only two zeroes in the data. In the zero-inflated model, the posterior distribution for p is concentrated around 1, hence the posterior for the proportion of excess zeroes $1 - p$ is concentrated around 0.

The main difference between the three estimates of the relative rate comes from the wider credible interval under the negative Binomial model.

You might now ask whether the negative Binomial model is "better" than the Poisson, that is, is there enough detectable overdispersion in the data that we would prefer to present this wider credible interval?

Here we could examine the dispersion parameter r. Values of around 2-4 are low, indicating overdispersion

(remember the negative binomial approaches the Poisson as $r \to \infty$).

We might also judge the amount of overdispersion by comparing the variance to the mean, since an overdispersed distribution is one where the variance is greater than the mean. Note that in this model, both the mean and variance depend on covariates. So we could pick an arbitrary covariate value, e.g. those for area `i=1`, add the following JAGS code to compute the variance,

```
var1        <-  (1 - p[1])*r/p[1]^2
```

compare the posterior for this quantity to the posterior of `mu[1]`, and observe the extent of difference.

Session 4 of the course will discuss ways to compare models more generally.