SOC542 Statistical Methods in Sociology II

Count outcomes

Thomas Davidson

Rutgers University

April 4, 2022

Course updates

- Homework 4 will be released new week
  - Count outcomes
  - Categorical and ordered outcomes

Plan

- Count outcomes
- Poisson regression
- Overdispersion and negative-binomial regression
- Offsets
- Zero-inflated models

Count outcomes

- Count outcomes are variables defined as *non-negative integers*.
  - Values must be 0 or greater.
  - Numbers must not contain any fractional component.

Count outcomes

- In general, we obtain count variables by counting discrete events over space and time. Many social processes produce counts:
  - How many people live in a census tract?
  - How many siblings does someone have?
  - How many times has someone been arrested?

Count outcome

Modeling counts using OLS

- We could treat counts like continuous variables and model them using OLS.
- Such a strategy might be appropriate if a count variable is normally distributed.
  - This could occur if a continuous variable was rounded.
- But like the LPM, we might run into problems when making predictions:

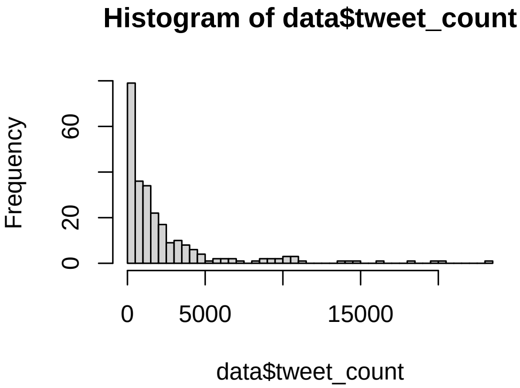
Data

Twitter and political parties in Europe

```
data <- read_csv("data/twitter_parties_2016.csv")#
data <- data %>%
    replace_na(list(left_right = 5,
                    seats_per = 0))
```

Data

Twitter and political parties in Europe



Count outcome

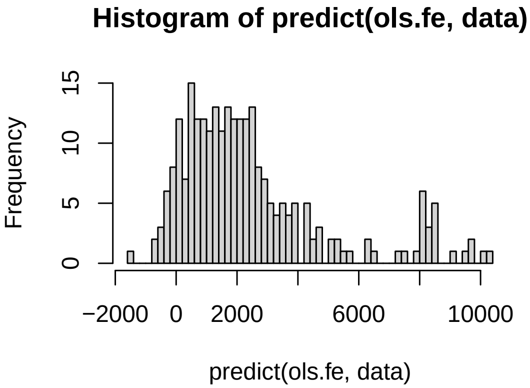
Modeling counts using OLS

	OLS	OLS FE	OLS FE (Log)
(Intercept)	2372.846***	1341.047	6.557***
	(555.477)	(1267.149)	(0.563)
populist	1457.289*	1184.876*	0.264
	(625.378)	(538.252)	(0.239)
left_right	-79.666	-76.613	-0.051
	(99.156)	(86.319)	(0.038)
seats_per	24.056	52.538**	0.022**
	(19.043)	(16.387)	(0.007)
Num.Obs.	255	255	255
R2	0.029	0.428	0.419
R2 Adj.	0.018	0.351	0.341
Log.Lik.	-2451.473	-2384.101	-415.757
F	2.520	5.580	5.375

Country FE omitted.

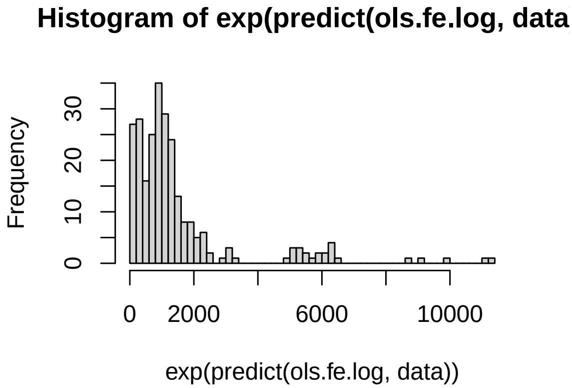
Count outcome

Making predictions with OLS



Count outcome

Making predictions with OLS



Poisson regression

Modeling counts as Poisson processes

- The **Poisson** distribution is a discrete probability distribution that indicates the number of events in a fixed time or space. These counts can be considered as rates of events per unit.[1].
- The *probability mass function* is defined by a single parameter λ, where the probability of observing k events is equal to

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

- For any Poisson distributed random variable, x

$$E(x) = \lambda = Var(x)$$

[1] The distribution gets its name from French mathematician Siméon Denis Poisson [1781-1840])

Poisson regression

Modeling counts as Poisson processes

- Let's say the average number of visits to the dentist in a single year is 1.6.
- We can model the probabilities of observing different numbers of visits given $\lambda = 1.6$:

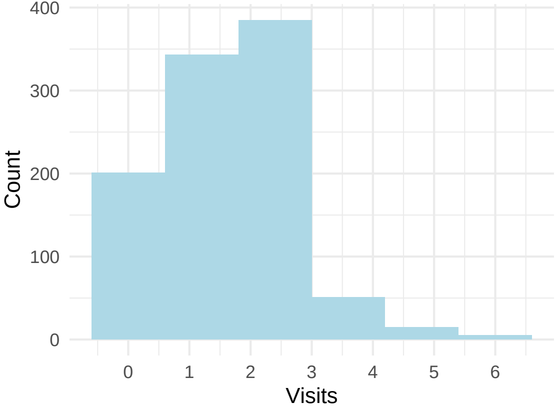
$$P(k \text{ visits a year}) = \frac{1.6^k e^{-1.6}}{k!}$$

$$P(0 \text{ visits a year}) = \frac{1.6^0 e^{-1.6}}{0!} = \frac{e^{1.6}}{1} \approx 0.2$$

$$P(1 \text{ visits a year}) = \frac{1.6^1 e^{-1.6}}{1!} = \frac{1.6e^{-1.6}}{1} \approx 0.4$$

Poisson regression

Poisson distributions



1000 draws from Poisson($\lambda = 1.6$)

Poisson regression

Poisson distributions, $E[x] = \lambda = Var(x)$

```
round(mean(x),2)
```

```
## [1] 1.57
```

```
round(var(x),2)
```

```
## [1] 1.54
```

Poisson regression

- The Poisson regression model assumes that the outcome is Poisson distributed, conditional on the observed predictions.

$$y \sim Poisson(\lambda)$$

- To ensure that our estimates are positive, we can use a logarithmic *link function*, thus

$$y = log(\lambda) = \beta_0 + \beta_1 x_1 + \beta_2 x_1 + ... + \beta_k x_k$$

- Like logistic regression, this equation can equivalently be expressed using the *inverse* of the logarithm function:

$$\lambda = e^{\beta_0 + \beta_1 x_1 + \beta_2 x_1 + ... + \beta_k x_k}$$

Poisson regression

Fitting a model

```
pois <- glm(tweet_count ~ populist + left_right +
                seats_per + country,
         data = data, family = poisson(link = "log"))
```

Poisson regression

	OLS FE (Log)	Poisson
(Intercept)	6.557***	7.233***
	(0.563)	(0.010)
populist	0.264	0.354***
	(0.239)	(0.003)
left_right	-0.051	-0.019***
	(0.038)	(0.001)
seats_per	0.022**	0.018***
	(0.007)	(0.000)
Num.Obs.	255	255
R2	0.419	
R2 Adj.	0.341	
Log.Lik.	-415.757	-211788.969

Country FE omitted.

+ $p < 0.1$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Poisson regression

Interpretation

- The intercept β_0 is the *logged* average value of the outcome when all other predictors are equal to zero.
- Each coefficient β_i indicates the effect of a unit change of x_i on the *logarithm* of the outcome.
  - e.g., $\beta_{populism} = 0.354$ implies that the expected log number of tweets for populist parties is higher than non-populists by 0.354.
- Coefficients can be interpreted as *multiplicative* changes after exponentiation
  - e.g., $e^{\beta_{populism}} = e^{0.354} \approx 1.425$. This implies that populist parties tweet 1.425 times as frequently or 42.5% more frequently than non-populists.
  - These coefficients are sometimes referred to as textbf{incident rate ratios (IRRs)}.

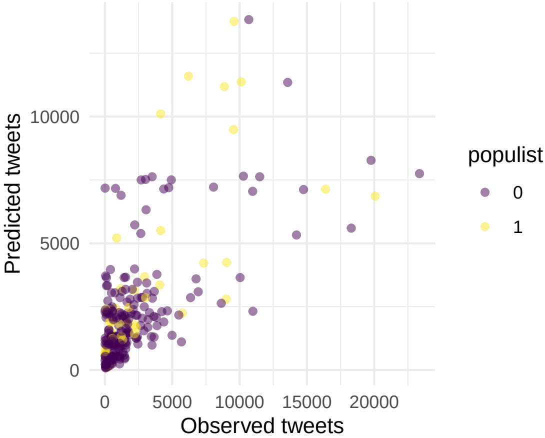
Poisson regression

	Poisson
(Intercept)	1384.634***
	(0.010)
populist	1.425***
	(0.003)
left_right	0.982***
	(0.001)
seats_per	1.018***
	(0.000)
Num.Obs.	255
Log.Lik.	-211788.969
F	14872.576

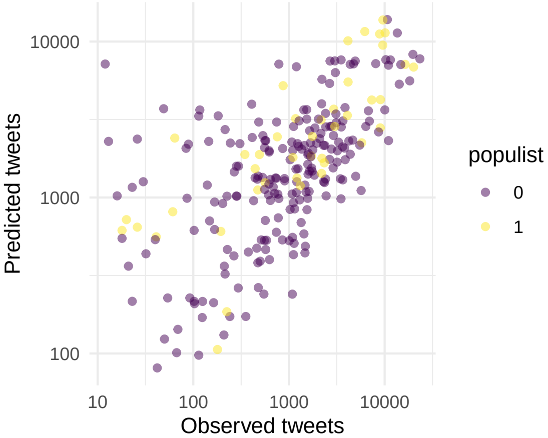
Country FE omitted.

+ $p < 0.1$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

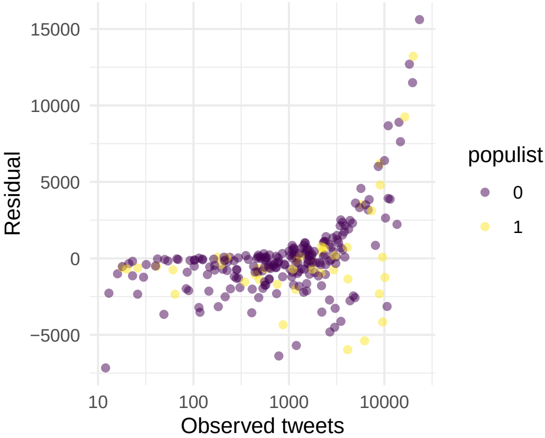
Poisson regression



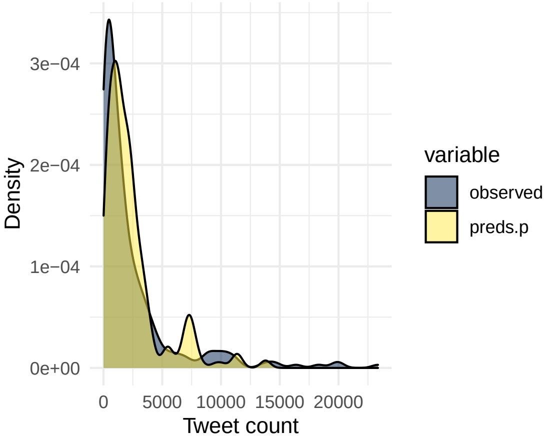
Poisson regression



Poisson regression



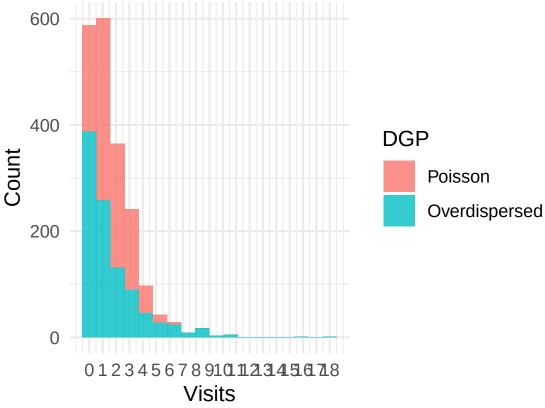
Poisson regression



Overdispersion

- A random variable is **overdispersed** if the observed variability is greater than the variability expected by the underlying probability model.
- In this case, we can see that the variance is far larger than the mean.
  - We could see this in the descriptive statistics, but the issue can only be properly diagnosed after fitting a model (note that the variance of the data is more than two times as large as the predicted values)
- **Underdispersion** occurs if the variability is lower than expected, but it is rarely an issue.

Overdispersion



Poisson($\lambda = 1.6$) and NegBin($\mu = 1.6$, $\theta = 1$)

Overdispersion

Negative binomial distribution and regression

- The **negative binomial** distribution (aka the gamma-Poisson distribution) includes an additional parameter θ to account for dispersion, referred to as a **scale parameter**.

$$y = NegativeBinomial(\lambda, \theta)$$

- In negative binomial regression, θ is estimated from the data. The value must be positive.
  - Lower values indicate greater overdispersion.
  - Negative binomial becomes Poisson as $\lim_{\theta \to \infty}$.

Overdispersion

Fitting a negative binomial regression

The procedure for estimating a negative binomial regression via Maximum Likelihood is not implemented in `glm`. Instead, we use the modified `glm.nb` function from the `MASS` package.

```
library(MASS)
nb <- glm.nb(tweet_count ~ populist + left_right +
                  seats_per + country,
         data = data)
```

Comparing Poisson and negative binomial regression

	Poisson	Negative binomial
(Intercept)	1384.634***	1614.471***
	(0.010)	(0.410)
populist	1.425***	1.320
	(0.003)	(0.174)
left_right	0.982***	0.950+
	(0.001)	(0.028)
seats_per	1.018***	1.023***
	(0.000)	(0.005)
Num.Obs.	255	255
Log.Lik.	-211788.969	-2120.171
F	14872.576	9.935

Country FE omitted. Exponentiated coefficients.

+ $p < 0.1$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Negative binomial regression

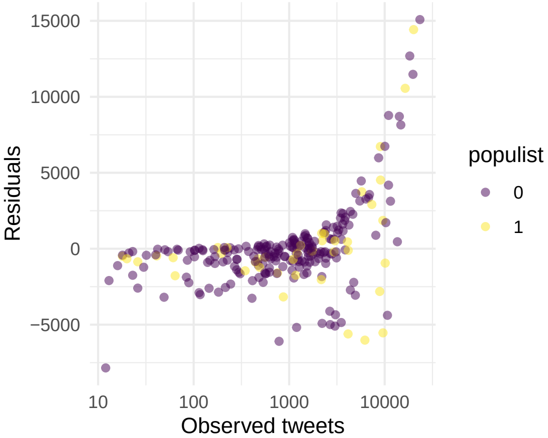
```r
nb$theta
```

```
## [1] 1.0883
```

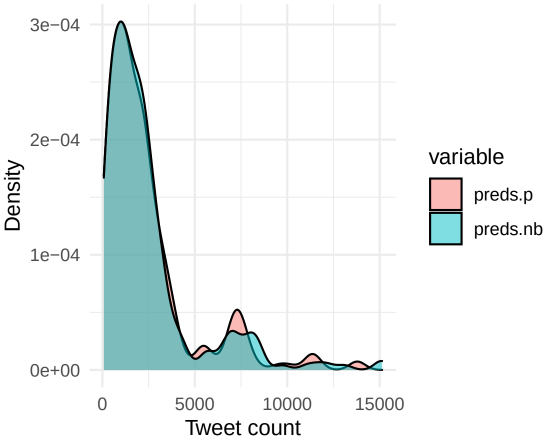
```r
nb$SE.theta
```

```
## [1] 0.08578371
```

Negative binomial regression



Negative binomial regression



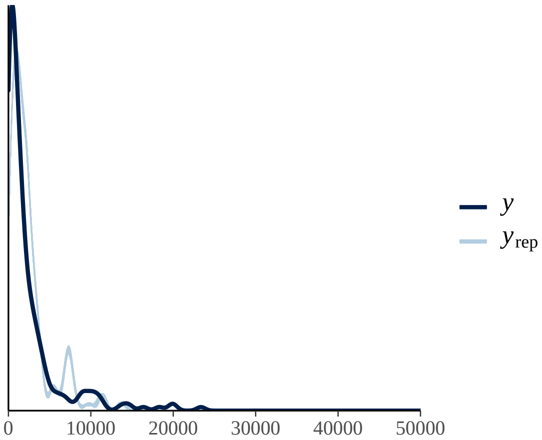
Negative binomial regression

Bayesian estimation

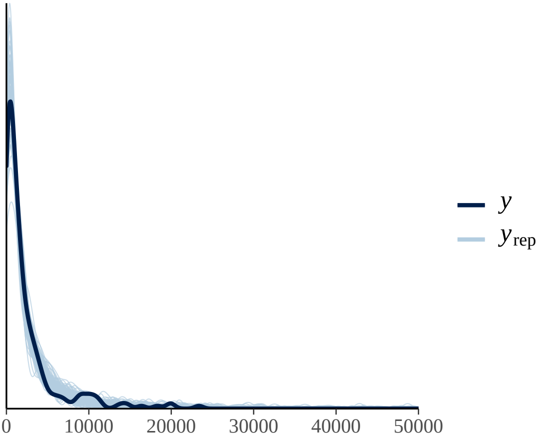
```
pois.b <- stan_glm(tweet_count ~ populist + left_right +
                           seats_per + country,
        data = data,
        family = poisson,
       seed = 08901, chains = 1, refresh = 0)

nb.b <- stan_glm(tweet_count ~ populist + left_right +
                           seats_per + country,
        data = data,
        family = neg_binomial_2(),
       seed = 08901, chains = 1, refresh = 0)
```

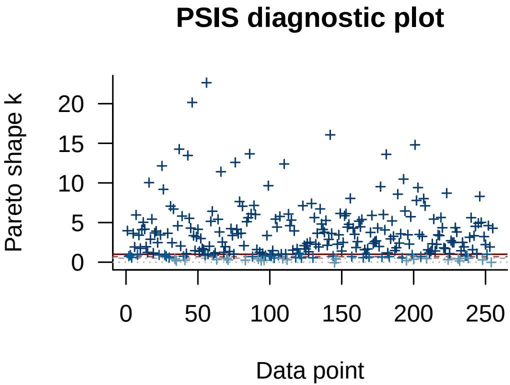
Poisson posterior predictive check



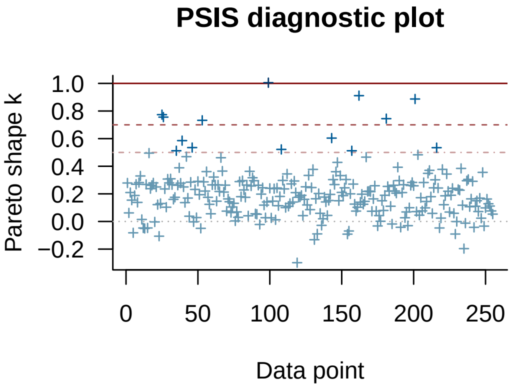
Negative binomial posterior predictive check



Poisson PSIS plot



Negative binomial PSIS plot



Negative binomial regression

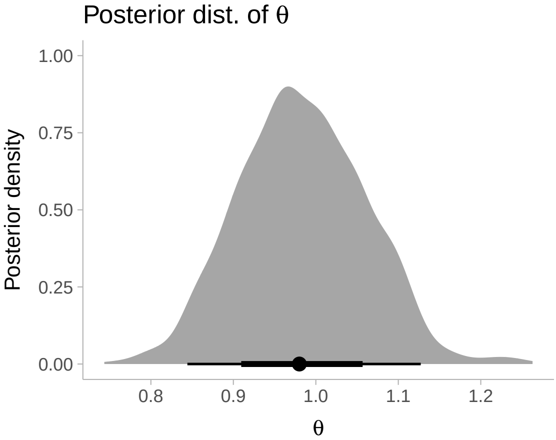
Comparing Poisson and negative binomial models

```
loo_compare(l.pois, l.nb)
```

```
##        elpd_diff se_diff
## nb.b        0.0       0.0
## pois.b -215832.9   24353.1
```

Negative binomial regression

Bayesian estimate of θ



Offsets

Intuition

- Assume a count outcome y is measured over varying time intervals t. The level of y will vary both as a function of the underlying count process and the length of **exposure**.[2]
- We can add an **offset** to our model to account for varying exposures.
- The outcome of a model with an offset is now $\frac{y}{t}$.

[2] The same logic would apply if we measured quantities over varying spatial units, e.g. counting people in blocks versus census tracts.

Offsets

Explanation

- The mean of a Poisson process, λ is implicitly $\lambda = \frac{\mu}{\tau}$, the expected number of events, μ, over the duration τ.
- Assume a Poisson process where λ_i is the expected number of events for the i^{th} observation. We can write the link function as

$$y = Poisson(\lambda)$$

$$log(\lambda) = log(\frac{\mu}{\tau}) = \beta_0 + \beta_1 x$$

- This can be re-written as

$$= log(\mu) - log(\tau) = \beta_0 + \beta_1 x$$

Offsets

Explanation

- We can think of τ as the number of **exposures** for each observation. Thus, we can write out a new model for μ:

$$y \sim Poisson(\mu)$$

$$log(\mu) = log(\tau) + \beta_0 + \beta_1 x$$

Offsets

Simulated example

```
N <- 1000
tweets <- sample(c(1:100), N, replace = TRUE)
ideology <- rbinom(N,1,0.4)
likes <- c()
for (i in 1:N) {
    y <- sum(rpois(tweets[i], exp(1 + 1*ideology + rnorm(1))))
    likes[i] <- y
}
sims <- as_tibble(cbind(tweets, likes, ideology))
```

Offsets

Simulated example

```
head(sims)
```

```
## # A tibble: 6 x 3
##   tweets likes ideology
##    <int> <int>    <int>
## 1     15   231        0
## 2     15    26        0
## 3      9    30        1
## 4     48   109        1
## 5      6    46        0
## 6     47    93        1
```

Offsets

Specification and interpretation

- The model is specified by adding the logarithm of exposures (e.g. $log(\tau)$) as an **offset** using the `offset` function.
  - The coefficient for the logarithm of exposures is fixed to $\beta_{offset} = 1$.
- The model is now interpreted as predicting a *rate* rather than a count.
- We could also directly include the logarithm of exposures as a predictor and let the model determine the coefficient.

Offsets

Simulated example

```r
m1 <- glm(likes ~ 1 + ideology,
          data = sims, family = poisson(link = "log"))
m2 <- glm(likes ~ ideology + log(tweets),
          data = sims, family = poisson(link = "log"))
m3 <- glm(likes ~ ideology + offset(log(tweets)),
          data = sims, family = poisson(link = "log"))
```

Offsets

	Poisson	Poisson (Log exposure)	Poisson (Offset)
(Intercept)	5.888***	2.073***	1.989***
	(0.002)	(0.014)	(0.002)
ideology	-0.150***	-0.175***	-0.176***
	(0.004)	(0.004)	(0.004)
log(tweets)		0.980***	
		(0.003)	
Num.Obs.	1000	1000	1000
Log.Lik.	-231359.705	-165445.532	-165464.738
F	1738.806	45275.265	2388.381

$+$ $p < 0.1$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Offsets

	Poisson	Poisson (Log exposure)	Poisson (Offset)
(Intercept)	360.718***	7.947***	7.309***
	(0.002)	(0.014)	(0.002)
ideology	0.861***	0.839***	0.839***
	(0.004)	(0.004)	(0.004)
log(tweets)		2.663***	
		(0.003)	
Num.Obs.	1000	1000	1000
Log.Lik.	-231359.705	-165445.532	-165464.738
F	1738.806	45275.265	2388.381

Exponentiated coefficients.

$+$ $p < 0.1$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Offsets

Example: Predicting retweet rates

- Three models of yearly retweets
  - No offset
  - Log(tweets) included as predictor
  - Log(tweets) included as offset

Offsets

Example: Predicting retweet rates

```
nb.rt <- glm.nb(retweet_total ~
                    populist + left_right + seats_per + country,
                    data = data)
nb.rt.e <- glm.nb(retweet_total ~ log(tweet_count) +
                      populist + left_right + seats_per + country,
                      data = data)
nb.rt.o <- glm.nb(retweet_total ~ offset(log(tweet_count)) +
                      populist + left_right + seats_per + country,
                      data = data)
```

Offsets

	NB	NB (Log exposure)	NB (Offset)
(Intercept)	9.795***	0.336	2.570***
	(0.599)	(0.561)	(0.456)
populist	0.375	0.068	0.163
	(0.255)	(0.189)	(0.194)
left_right	-0.047	0.000	-0.017
	(0.041)	(0.030)	(0.031)
seats_per	0.039***	0.020***	0.026***
	(0.008)	(0.006)	(0.006)
log(tweet_count)		1.324***	
		(0.053)	
Num.Obs.	255	255	255
Log.Lik.	-2858.070	-2752.504	-2762.354
F	25.315	56.445	13.570

+ $p < 0.1$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

Offsets

Using offsets

- Include an offset if there are differences in measurement intervals across observations.
- Offsets allow models to be interpreted as rates rather than counts.
- The logarithm of exposures can also be directly modeled, but interpretation is less intuitive.

Zero-inflated models

Intuition

- Some count outcomes have high rates of zeros. What if the outcomes with a value of zero are generated by a different kind of process?
- **Zero-inflated models** allow us to separately model the process determining whether counts are non-zero and the expected count for each observations.

Zero-inflated models

Specification

- The zero-inflated Poisson model consists of a mixture of two linear models, a logistic regression predicting the probability of a zero and a Poisson model predicting the count outcome.

$$y_i = ZIPoisson(p, \lambda)$$

$$logit(p) = \beta_{0p} + \beta_{1p}x$$

$$log(\lambda) = \beta_{0\lambda} + \beta_{1\lambda}x$$

- Each model has its own parameters. These can be specified to model each process.

Zero-inflated models

Example: Books borrowed from the library

```
N <- 100
prob_lib <- 0.6
lib <- rbinom(N, 1, prob_lib)
sum(lib)/N
```

```
## [1] 0.47
```

Zero-inflated models

Example: Books borrowed from the library

```
x <- rnorm(N)

books <- c()
for (i in 1:N) {
    if (lib[i] == 1) {
        b <- rpois(1, lambda = exp(1 + 0.3*x[i] + rnorm(1)))
        books[i] <- b
    }
    else {books[i] <- 0}
}
mean(books)
```

```
## [1] 2.37
```

```
max(books)
```

```
## [1] 62
```

Zero-inflated models

Two kinds of zeros

```
sum(books == 0)
```

```
## [1] 65
```

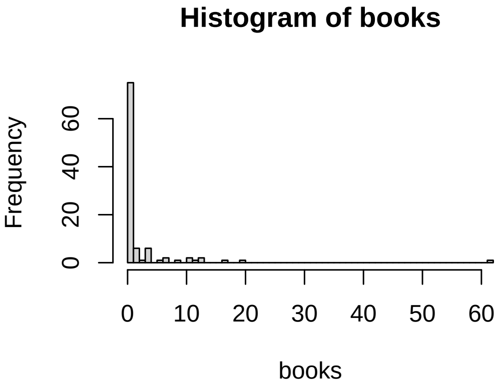
```
sum(books == 0 & lib == 1)
```

```
## [1] 12
```

```
sum(books == 0 & lib == 0)
```

```
## [1] 53
```

Zero-inflated models

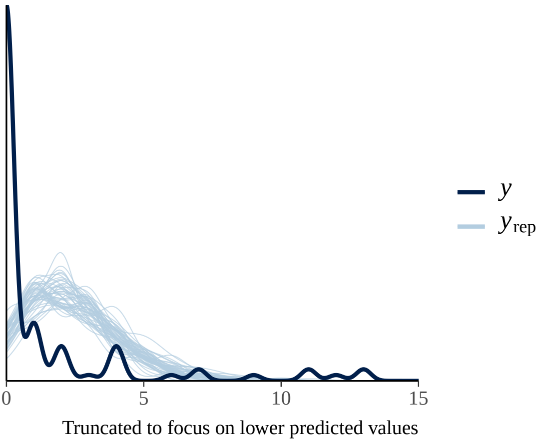


Zero-inflated models

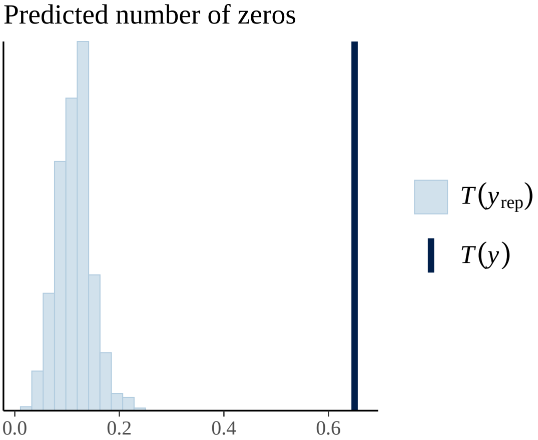
Estimating a Poisson model

```
book.data <- as.data.frame(cbind(books, x))
pois.m <- stan_glm(books ~ x, data = book.data, family = poisson(),
       seed = 08901, chains = 1, refresh = 0)
```

Zero-inflated models



Zero-inflated models



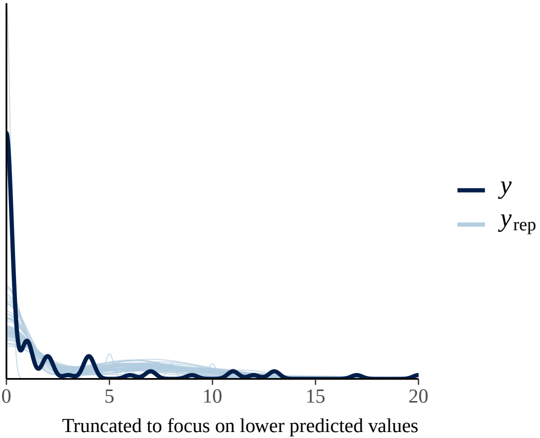
Zero-inflated models

Estimating a zero-inflated Poisson model

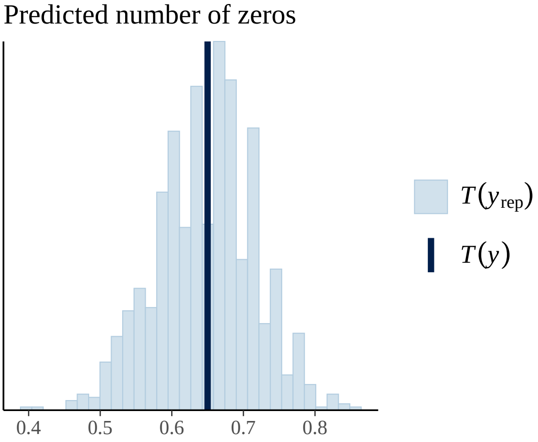
We must use the brms library to implement Bayesian zero-inflated Poisson regression.

```
library(brms)
zip <- brm(books ~ x,
                  data = book.data,
                  family = zero_inflated_poisson(link = "log",
                                                 link_zi = "logit"),
                  seed = 08901, refresh = 0, chains = 1)
```

Zero-inflated models



Zero-inflated models



Zero-inflated models

Comparing standard and zero-inflated models

```
##        elpd_diff se_diff
## zip       0.0       0.0
## pois.m -175.8      75.2
```

Summary

- Standard linear models are generally unsuitable for count data
- Poisson regression can be used for most count outcomes
- Overdispersion occurs when variation higher than expected under Poisson model
  - Negative binomial regression includes a scale parameter
- Zero-inflated models are used to decompose processes generating zeros and counts

Next week

- Categorical outcomes
  - Multinomial and ordered logistic regression