

## Class 12: Partial pooling, zero-inflation, and multinomial models

Andrew Parnell  
andrew.parnell@mu.ie



### Learning outcomes:

- ▶ Be able to describe the advantages of partial pooling
- ▶ Be able to fit some basic zero inflation and hurdle models
- ▶ Be able to understand and fit some multinomial modelling examples

1 / 36

2 / 36

### A false dichotomy: fixed vs random effects

- ▶ We've been fitting a model with varying intercepts and slopes to the earnings data:

$$y_i \sim N(\alpha_{\text{eth}_i} + \beta_{\text{eth}_i} x_i, \sigma^2)$$

where:

$$\alpha_j \sim N(\mu_\alpha, \sigma_\alpha^2) \text{ and } \beta_j \sim N(\mu_\beta, \sigma_\beta^2)$$

- ▶ In traditional parlance this is a random effects model
- ▶ When we fit our model we are learning about the values of the slopes and intercepts, and also the values of their means and standard deviations

### The extremes of varying vs fixed parameters

- ▶ Now consider what happens when  $\sigma_\alpha$  and  $\sigma_\beta$  get smaller and smaller. What will happen to the values of the slopes and the intercepts?
- ▶ Alternatively, consider what happens as  $\sigma_\alpha$  and  $\sigma_\beta$  get larger and larger?
- ▶ Are these still random effects models?

3 / 36

4 / 36

## The advantages of borrowing strength

- ▶ The process of  $\sigma_\alpha$  and  $\sigma_\beta$  getting smaller or larger will control the degree to which the slopes and intercepts are similar to each other
- ▶ If they are similar to each other we say they are *borrowing strength* as data in the other groups is influencing the intercept/slope. This is a powerful idea
- ▶ Mathematically you can write out the estimated mean of the parameters as a weighted average of the group mean and the overall mean where the weights are dependent on the group and overall variance and sample sizes.
- ▶ Because of the weighted nature of the estimate this is often called *partial pooling*

5 / 36

## Zero-inflation and hurdle models

- ▶ Let's introduce some new data. This is data from an experiment on whiteflies:

```
wf = read.csv('../data/whitefly.csv')
head(wf)
```

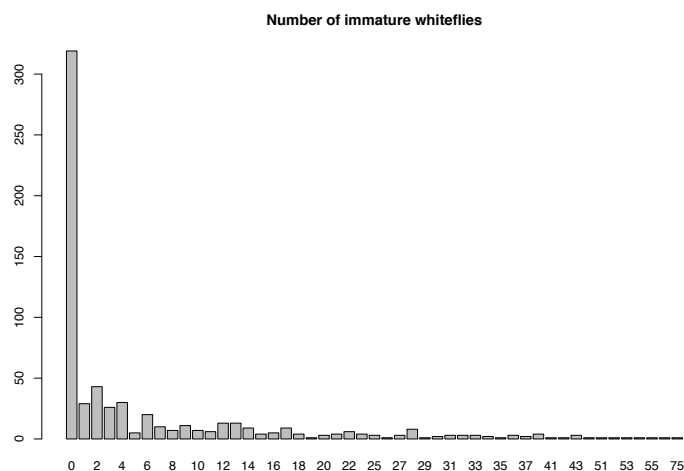
```
##   imm week block trt   n live plantid
## 1  15    1     3   5 12   11       1
## 2  16    2     3   5  8    6       1
## 3  28    3     3   5 10   10       1
## 4  17    4     3   5 10    8       1
## 5   9    5     3   5 10   10       1
## 6  28    6     3   5 10   10       1
```

The response variable here is the count `imm` of immature whiteflies, and the explanatory variables are `block` (plant number), `week`, and treatment `treat`.

6 / 36

## Look at those zeros!

```
barplot(table(wf$imm),
          main = 'Number of immature whiteflies')
```



7 / 36

## A first model

- ▶ These are count data so a Poisson distribution is a good start
- ▶ Let's consider a basic Poisson distribution model for  $Y_i$ ,  $i = 1, \dots, N$  observations:

$$Y_i \sim \text{Po}(\lambda_i)$$

$$\log(\lambda_i) = \beta_{\text{trt}_i}$$

- ▶ We'll only consider the treatment effect but we could run much more complicated models with e.g. other covariates and interactions

8 / 36

## Fitting the model in Stan

```
stan_code = '  
data {  
  int<lower=0> N;  
  int<lower=0> N_trt;  
  int<lower=0> y[N];  
  int trt[N];  
}  
parameters {  
  real beta_trt[N_trt];  
  real trt_mean;  
  real<lower=0> trt_sd;  
}  
model {  
  for (i in 1:N)  
    y[i] ~ poisson_log(beta_trt[trt[i]]);  
  
  // Priors on coefficients  
  for (j in 1:N_trt)  
    beta_trt[j] ~ normal(trt_mean, trt_sd);  
  
  trt_mean ~ normal(0, 10);  
  trt_sd ~ cauchy(0, 5);  
}  
'
```

9 / 36

## Running the model

```
stan_run = stan(data = list(N = nrow(wf),  
  N_trt = length(unique(wf$trt)),  
  y = wf$imm,  
  trt = wf$trt),  
  model_code = stan_code)
```

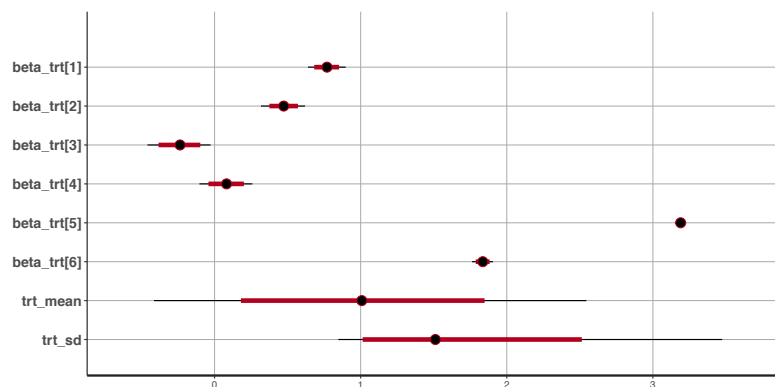
10 / 36

## Results

```
plot(stan_run)
```

```
## ci_level: 0.8 (80% intervals)
```

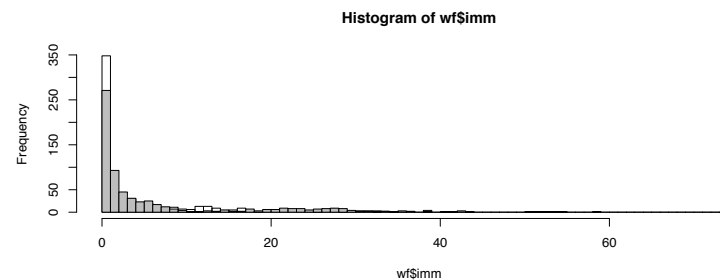
```
## outer_level: 0.95 (95% intervals)
```



11 / 36

## Did the model actually fit well?

```
pars = extract(stan_run, pars = 'beta_trt')$beta_trt  
beta_means = apply(pars, 2, 'mean')  
y_sim_mean = exp(beta_means[wf$trt])  
y_sim = rpois(nrow(wf), y_sim_mean)  
hist(wf$imm, breaks = seq(0, max(wf$imm)))  
hist(y_sim, breaks = seq(0, max(wf$imm)),  
  add = TRUE, col = 'gray')
```



12 / 36

## What about the zeros?

- ▶ One way of broadening the distribution is through over-dispersion which we have already met:

$$\log(\lambda_i) \sim N(\beta_{\text{trt}_i}, \sigma^2)$$

- ▶ However this doesn't really solve the problem of excess zeros
- ▶ Instead there are a specific class of models called *zero-inflation* models which use a specific probability distribution. The zero-inflated Poisson (ZIP) with ZI parameter  $q_0$  is written as:

$$p(y|\lambda) = \begin{cases} q_0 + (1 - q_0) \times \text{Poisson}(0, \lambda) & \text{if } y = 0 \\ (1 - q_0) \times \text{Poisson}(y, \lambda) & \text{if } y \neq 0 \end{cases}$$

13 / 36

## Fitting models with custom probability distributions

- ▶ The Zero-inflated Poisson distribution is not included in Stan by default. We have to create it
- ▶ It's pretty easy to create new probability distributions in Stan
- ▶ In Stan all of you have to do is give it a way of computing the likelihood score, which it keeps track of via a variable called `target`

14 / 36

## Fitting the ZIP in Stan

```
stan_code = '
data {
  int<lower=0> N;
  int<lower=0> N_trt;
  int<lower=0> y[N];
  int trt[N];
}
parameters {
  real<lower=0, upper=1> q_0;
  real beta_trt[N_trt];
  real trt_mean;
  real<lower=0> trt_sd;
}
model {
  for(j in 1:N_trt)
    beta_trt[j] ~ normal(trt_mean, trt_sd);
  trt_mean ~ normal(0, 10);
  trt_sd ~ cauchy(0, 5);

  for (i in 1:N) {
    if (y[i] == 0)
      target += log_sum_exp(bernoulli_lpmf(1 | q_0),
                           bernoulli_lpmf(0 | q_0)
                           + poisson_log_lpmf(y[i] | beta_trt[trt[i]]));
    else
      target += bernoulli_lpmf(0 | q_0) + poisson_log_lpmf(y[i] | beta_trt[trt[i]]);
  }
}
```

15 / 36

## Running the model

```
stan_run = stan(data = list(N = nrow(wf),
                           N_trt = length(unique(wf$trt)),
                           y = wf$imm,
                           trt = wf$trt),
               model_code = stan_code)
```

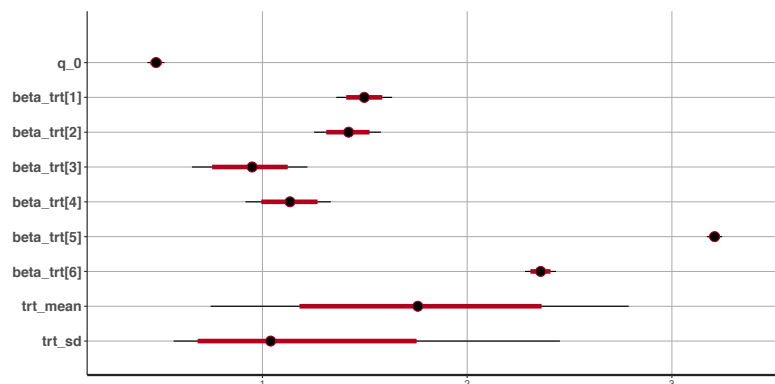
16 / 36

## Results

```
plot(stan_run)
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```



17 / 36

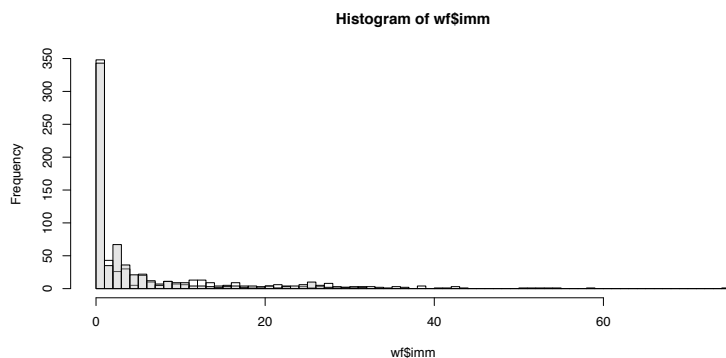
## Did it work any better? - code

```
beta_means = apply(extract(stan_run,
                           pars = 'beta_trt')$beta_trt,
                   2, 'mean')
q_0_mean = mean(extract(stan_run, pars = 'q_0')$q_0)
y_sim_mean = exp(beta_means[wf$trt])
rZIP = function(mean, q_0) {
  pois = rpois(length(mean), mean)
  pois[runif(length(mean)) < q_0] = 0
  return(pois)
}
y_sim = rZIP(y_sim_mean, q_0_mean)
```

18 / 36

## Did it work any better? - picture

```
hist(wf$imm, breaks = seq(0,max(wf$imm)))
hist(y_sim, breaks = seq(0,max(wf$imm)),
     add = TRUE, col = rgb(0.75,0.75,0.75,0.4))
```



19 / 36

## Some more notes on Zero-inflated Poisson

- ▶ This model seems to over-predict the number of zeros! It would be interesting to perhaps try having a different probability of zeros ( $q_0$ ) for different treatments
- ▶ It might be that the other covariates explain some of the zero behaviour
- ▶ We could further add in both zero-inflation and over-dispersion

20 / 36

## An alternative: hurdle models

- ▶ ZI models work by having a parameter (here  $q_0$ ) which is the probability of getting a zero, and so the probability of getting a Poisson value (which could also be a zero) is 1 minus this value
- ▶ An alternative (which is slightly more complicated) is a hurdle model where  $q_0$  represents the probability of the *only* way of getting a zero. With probability  $(1-q_0)$  we end up with a special Poisson random variable which has to take values 1 or more
- ▶ In some ways this is richer than a ZI model since zeros can be deflated or inflated

21 / 36

## A hurdle-Poisson model in Stan

```
stan_code = '  
data {  
  int<lower=0> N;  
  int<lower=0> N_trt;  
  int<lower=0> y[N];  
  int trt[N];  
}  
parameters {  
  real<lower=0, upper=1> q_0;  
  real beta_trt[N_trt];  
  real trt_mean;  
  real<lower=0> trt_sd;  
}  
model {  
  for(j in 1:N_trt)  
    beta_trt[j] ~ normal(trt_mean, trt_sd);  
  trt_mean ~ normal(0, 10);  
  trt_sd ~ cauchy(0, 5);  
  
  for(i in 1:N) {  
    if(y[i] == 0)  
      target += log(q_0);  
    else  
      target += logim(q_0) + poisson_log_lpmf(y[i] | beta_trt[trt[i]])  
        - poisson_lccdf(0 | exp(beta_trt[trt[i]]));  
  }  
}
```

22 / 36

## Running the model

```
stan_run = stan(data = list(N = nrow(wf),  
  N_trt = length(unique(wf$trt)),  
  y = wf$imm,  
  trt = wf$trt),  
  model_code = stan_code)
```

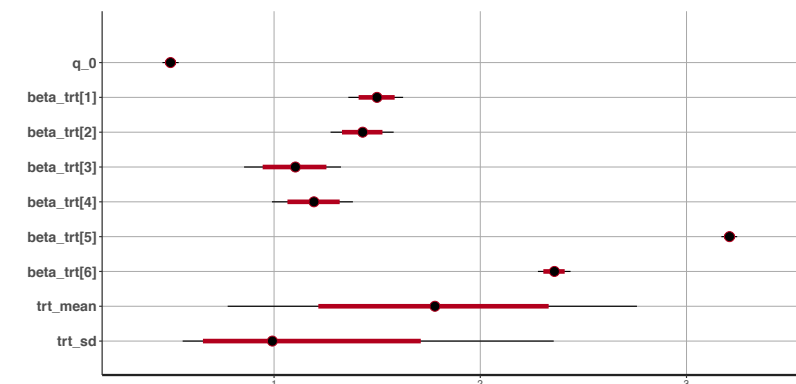
23 / 36

## Results

```
plot(stan_run)
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```



24 / 36

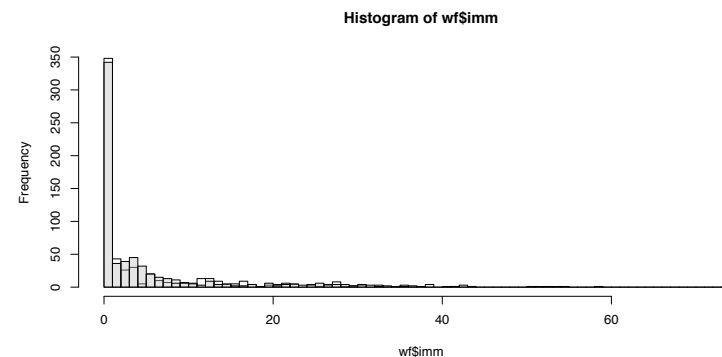
## Did it work any better? - code

```
beta_means = apply(extract(stan_run,
                           pars = 'beta_trt')$beta_trt,
                   2, 'mean')
q_0_mean = mean(extract(stan_run, pars = 'q_0')$q_0)
y_sim_mean = exp(beta_means[wf$trt])
rZIP = function(mean, q_0) {
  pois = rpois(length(mean), mean)
  pois[runif(length(mean)) < q_0] = 0
  return(pois)
}
y_sim = rZIP(y_sim_mean, q_0_mean)
```

25 / 36

## Did it work any better? - picture

```
hist(wf$imm, breaks = seq(0, max(wf$imm)))
hist(y_sim, breaks = seq(0, max(wf$imm)),
     add = TRUE, col = rgb(0.75, 0.75, 0.75, 0.4))
```



26 / 36

## The multinomial distribution

- Multinomial data can be thought of as multivariate discrete data
- It's usually used in two different scenarios:
  1. For classification, when you have an observation falling into a single one of  $K$  possible categories
  2. For multinomial regression, where you have a set of counts which sum to a known value  $N$
- We will just consider the multinomial regression case, whereby we have observations  $y_i = [y_{i1}, \dots, y_{iK}]$  where the sum  $\sum_{k=1}^K y_{ik} = N_i$  is fixed
- The classification version is a simplification of the regression version

27 / 36

## Some new data! - pollen

```
pollen = read.csv('../data/pollen.csv')
head(pollen)
```

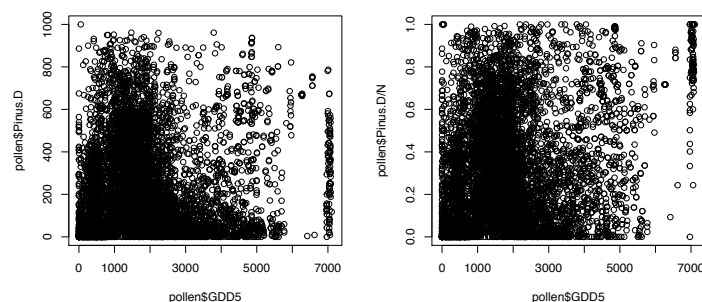
##	GDD5	MTCU	Abies	Alnus	Betula	Picea	Pinus.D	Quercus.D	Gramineae
## 1	1874	-7.9	0	50	158	7	721	22	0
## 2	1623	-5.5	0	38	28	302	537	19	0
## 3	1475	-4.7	0	276	183	110	136	0	0
## 4	1360	-8.8	0	111	354	141	364	0	0
## 5	1295	-6.9	0	91	50	151	708	0	0
## 6	1539	-7.8	0	51	194	82	673	0	0

These data are pollen counts of 7 varieties of pollen from modern samples with two covariates

28 / 36

## Some plots

- ▶ The two covariates represent the length of the growing season (GDD5) and harshness of the winter (MTCO)
- ▶ The task is to find which climate regimes each pollen variety favours



29 / 36

## A multinomial model

- ▶ The multinomial distribution is often written as:

$$[y_{i1}, \dots, y_{iK}] \sim \text{Mult}(N_i, \{p_{i1}, \dots, p_{iK}\})$$

or, for short:

$$y_i \sim \text{Mult}(N_i, p_i)$$

- ▶ The key parameters here are the probability vectors  $p_i$ . It's these we want to use a link function on to include the covariates
- ▶ We need to be careful as each must sum to one:  $\sum_{k=1}^K p_{ik} = 1$ . Any link function must satisfy this constraint

30 / 36

## Prior distributions on probability vectors

- ▶ When  $K = 2$  we're back the binomial-logit we met in the first day, and we can use the logit link function
- ▶ When  $K > 2$  a common function to use is the *soft-max* function:

$$p_{ik} = \frac{\exp(\theta_{ik})}{\sum_{j=1}^K \exp(\theta_{ij})}$$

- ▶ This is a generalisation of the logit function
- ▶ The next layer of our model sets, e.g.:

$$\theta_{ik} = \alpha_k + \beta_k \text{GDD5}_i + \gamma_k \text{MTCO}_i$$

31 / 36

## Stan code part 1

```
stan_code = '
data {
  int<lower=1> n;
  int<lower=1> K;
  int<lower=0> y[n,K];
  real x1[n];
  real x2[n];
}
parameters {
  vector[K] alpha;
  vector[K] beta;
  vector[K] gamma;
  real alpha_mean;
  real beta_mean;
  real gamma_mean;
  real<lower=0> sigma_alpha;
  real<lower=0> sigma_beta;
  real<lower=0> sigma_gamma;
}
transformed parameters {
  vector[K] theta[n];
  simplex[K] p[n];

  for(i in 1:n){
    theta[i] = alpha + beta*x1[i] + gamma*x2[i];
  }
  for(i in 1:n){
    p[i] = softmax(theta[i]);
  }
}
...

```

32 / 36



## Stan code part 2

```
stan_code = '
...
model {
  for(k in 1:K) {
    alpha[k] ~ normal(alpha_mean, sigma_alpha);
    beta[k] ~ normal(beta_mean, sigma_beta);
    gamma[k] ~ normal(gamma_mean, sigma_gamma);
  }
  alpha_mean ~ normal(0, 10);
  beta_mean ~ normal(0, 10);
  gamma_mean ~ normal(0, 10);
  sigma_alpha ~ cauchy(0, 5);
  sigma_beta ~ cauchy(0, 5);
  sigma_gamma ~ cauchy(0, 5);

  for(i in 1:n)
    y[i] ~ multinomial(p[i]);
}
```

33 / 36

## Notes about this model

- ▶ This model is not going to fit very well, since it is unlikely that a linear relationship between the covariates and the pollen counts will match the data
- ▶ It might be better to use e.g. a spline model (see later class)
- ▶ Similarly we might need some complex interactions between the covariates as they are strongly linked
- ▶ We have constrained the parameters here so that the slopes and intercepts borrow strength across species. Does this make sense? What else could we do?

34 / 36

## Some final notes about multinomial models

- ▶ These models can be a pain to deal with as there are tricky constraints on the  $\theta$  parameters to make them all sum to 1.
- ▶ The softmax function is one choice but there are lots of others (logistic ratios, the Dirichlet distribution, ...)
- ▶ Whilst the classification version of this model just has binary  $y_i$  (with just a single 1 in it) most packages (including Stan) have a special distribution for this situation

35 / 36

## Summary

- ▶ We have seen how partial pooling is a balance between a model of complete independence and complete dependence between groups
- ▶ We have fitted some zero inflated and hurdle Poisson models
- ▶ We have seen some (poorly fitting) multinomial regression models

36 / 36