Session 3

Multi-population Hui-Walter models

Matt Denwood

2022-06-09

Reminders

All of the material is on the GitHub repository

- We may tweak material as we go along
- Remember to pull changes at the start of each day!
- And click *refresh* in your browser!

Reminders

All of the material is on the GitHub repository

- We may tweak material as we go along
- Remember to pull changes at the start of each day!
- And click *refresh* in your browser!

R code tips:

- Watch out for errors (red text!) in RStuio output
- You probably need these at the top of every R code file:

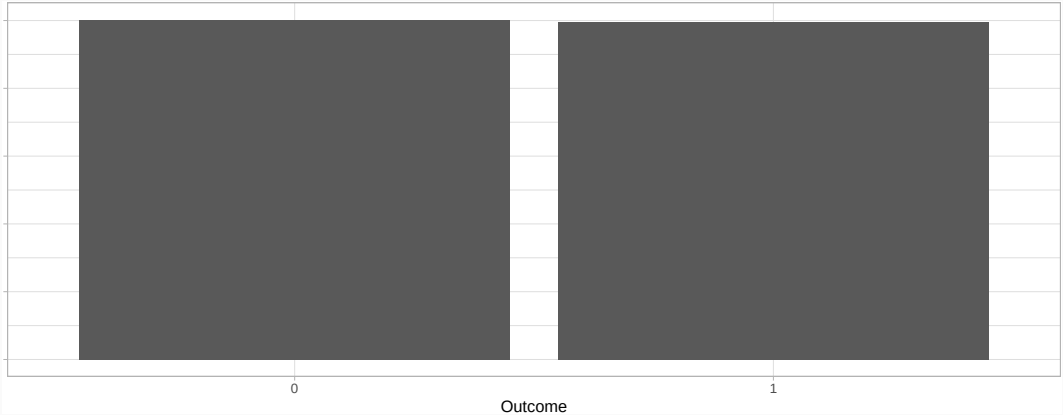
```
library("tidyverse")
library("runjags")
library("PriorGen")
library("TeachingDemos")
```

Recap

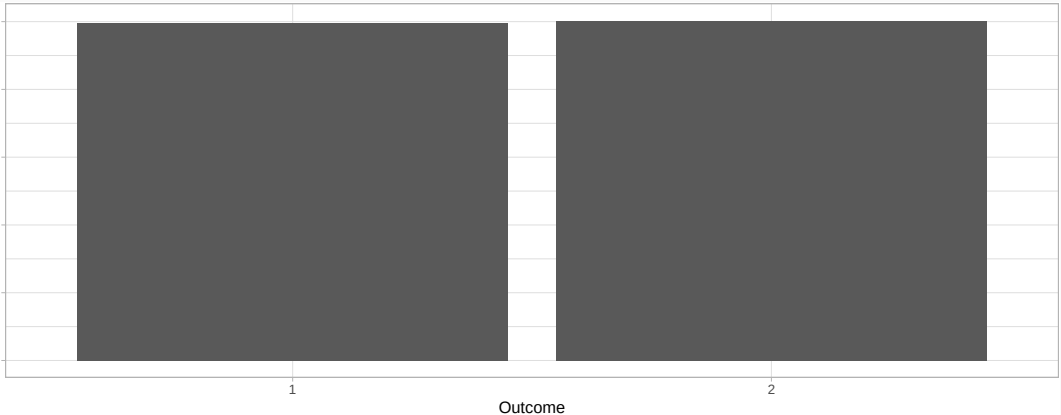
- Fitting models using MCMC is easy with JAGS / runjags
- But we must **never forget** to check convergence and effective sample size!
- More complex models become easy to implement
  - For example imperfect diagnostic tests, and Hui-Walter models
  - But remember to be realistic about what is possible with your data
  - Also carefully consider the influence of your priors

The multinomial distribution

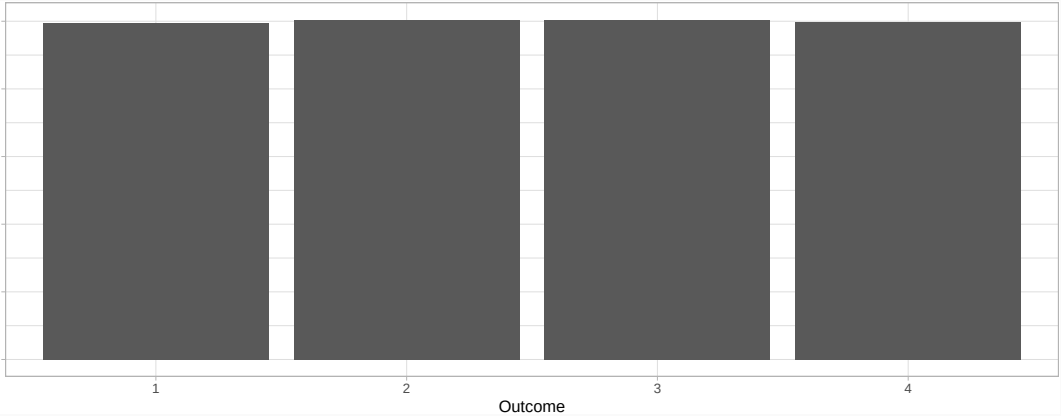
Binomial (always with two possible outcomes):



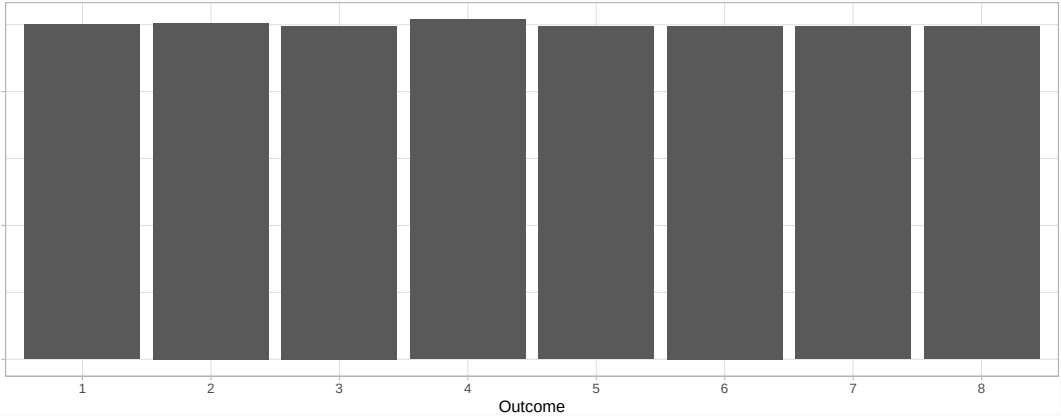
Multinomial with two possible outcomes:



Multinomial with four possible outcomes:



Multinomial with eight possible outcomes:



etc!

Diagnsotic test evaluation with one population

The two-test one-population model we looked at yesterday:

- Has five parameters to estimate:
  - Prevalence, 2 x sensitivity, 2 x specificity
- Has three pieces of information in the data:
  - The number of $+/+$, $+/-$ and $-/+$ pairs
  - Remember that N is fixed, so there are only 3 degrees of freedom!

Diagnsotic test evaluation with one population

The two-test one-population model we looked at yesterday:

- Has five parameters to estimate:
  - Prevalence, 2 x sensitivity, 2 x specificity
- Has three pieces of information in the data:
  - The number of $+/+$, $+/-$ and $-/+$ pairs
  - Remember that N is fixed, so there are only 3 degrees of freedom!
- Is therefore only identifiable with strong priors for one test

Diagnsotic test evaluation with one population

The two-test one-population model we looked at yesterday:

- Has five parameters to estimate:
  - Prevalence, 2 x sensitivity, 2 x specificity
- Has three pieces of information in the data:
  - The number of $+/+$, $+/-$ and $-/+$ pairs
  - Remember that N is fixed, so there are only 3 degrees of freedom!
- Is therefore only identifiable with strong priors for one test

How can we improve on this situation?

The Hui-Walter Paradigm

Hui-Walter models have multiple populations

- We can extend the one-population idea to multiple populations:
  - Each adds one new parameter (prevalence)
  - But adds three new pieces of information (test results)

Hui-Walter models have multiple populations

- We can extend the one-population idea to multiple populations:
  - Each adds one new parameter (prevalence)
  - But adds three new pieces of information (test results)

Borderline identifiable (parameters = degrees of freedom):

- Two tests, two populations
- Three tests, one population (see later!)

More easily identifiable (parameters < degrees of freedom):

- Two tests, three or more populations

Hui-Walter models have multiple populations

- We can extend the one-population idea to multiple populations:
  - Each adds one new parameter (prevalence)
  - But adds three new pieces of information (test results)

Borderline identifiable (parameters = degrees of freedom):

- Two tests, two populations
- Three tests, one population (see later!)

More easily identifiable (parameters < degrees of freedom):

- Two tests, three or more populations

Works best with multiple populations each with differing prevalence

- BUT be wary of assumptions regarding constant sensitivity/specificity!

Different prevalence in different populations

```
model{
  for(p in 1:Populations){
    Tally[1:4, p] ~ dmulti(prob[1:4, p], TotalTests[p])
    # Test1- Test2- Pop1
    prob[1, p] <- (prev[p] * ((1-se[1])*(1-se[2]))) + ((1-prev[p]) * ((sp[1])*(sp[2])))

    ## snip ##

    prev[p] ~ dbeta(1, 1)
  }

  se[1] ~ dbeta(1, 1)T(1-sp[1], )
  sp[1] ~ dbeta(1, 1)
  se[2] ~ dbeta(1, 1)T(1-sp[2], )
  sp[2] ~ dbeta(1, 1)

  #data# Tally, TotalTests, Populations
  #monitor# prev, prob, se, sp
  #inits# prev, se, sp
}
```

Multiple populations: assumptions

- We typically assume that the sensitivity and specificity *must* be consistent between populations
  - Do you have an endemic and epidemic population?
  - Or vaccinated and unvaccinated?
  - If so then the assumptions might not hold!

Multiple populations: assumptions

- We typically assume that the sensitivity and specificity *must* be consistent between populations
  - Do you have an endemic and epidemic population?
  - Or vaccinated and unvaccinated?
  - If so then the assumptions might not hold!
- The populations can be artificial (e.g. age groups) but must not be decided based on the diagnostic test results
  - It helps if the prevalence differs between the populations

Multiple populations: special cases

- A small disease-free group is extremely helpful
  - Contains strong data regarding specificity
  - As long as specificity can be assumed to be the same in the other populations

Multiple populations: special cases

- A small disease-free group is extremely helpful
  - Contains strong data regarding specificity
  - As long as specificity can be assumed to be the same in the other populations
- A small experimentally infected group MAY be helpful but it is often dangerous to assume that sensitivity is consistent!

Initial values

We have to be careful to make sure that the length of initial values for `prev` in each chain is equal to the number of populations

For example with 5 populations we need:

```
prev <- list(chain1=c(0.1, 0.1, 0.1, 0.9, 0.9), chain2=c(0.9, 0.9, 0.9, 0.1, 0.1))
```

Initial values

We have to be careful to make sure that the length of initial values for `prev` in each chain is equal to the number of populations

For example with 5 populations we need:

```
prev <- list(chain1=c(0.1, 0.1, 0.1, 0.9, 0.9), chain2=c(0.9, 0.9, 0.9, 0.1, 0.1))
```

The values you choose for different populations in the same chain can be the same - just make sure you pick different values for the same population between chains (i.e. *over-dispersed* initial values)

Note: specifying your own initial values is technically optional with JAGS, but it is always a good idea!!!

Incorporating populations with known prevalence

Up to now prevalence has been a parameter, but it can also be (partially) observed:

```
model{
  for(p in 1:Populations){
    Tally[1:4, p] ~ dmulti(prob[1:4, p], TotalTests[p])
    # Test1- Test2- Pop1
      prob[1, p] <- (prev[p] * ((1-sensitivity[1])*(1-sensitivity[2]))) + ((1-prev[p]) *
↪  ((sp[1])*(sp[2])))

      ## snip ##

    prev[p] ~ dbeta(1, 1)
  }

  ## snip ##

  #data# Tally, TotalTests, Populations, se_prior, sp_prior, prev
  #monitor# prev, prob, se, sp
  #inits# prev, se, sp
}
```

To fix the prevalence of population 1 we could do:

```
Populations <- 5
prev <- rep(NA, Populations)
prev[1] <- 0
prev
## [1]  0 NA NA NA NA
```

To fix the prevalence of population 1 we could do:

```
Populations <- 5
prev <- rep(NA, Populations)
prev[1] <- 0
prev
## [1]  0 NA NA NA NA
```

But you also need to account for this in the initial values:

```
prev <- list(chain1=c(NA, 0.1, 0.1, 0.9, 0.9), chain2=c(NA, 0.9, 0.9, 0.1, 0.1))
```

Note: we now have two definitions of `prev` in R!

Data and initial value lists

There are actually multiple ways to specify data and initial values to runjags, including via the `data` and `inits` arguments

We will use these to keep separate lists of data and initial values (these could also be data frames, or environments)

```
data <- list(
  Tally = Tally,
  TotalTests = apply(Tally, 2, sum),
  Populations = dim(Tally)[2],
  prev = rep(NA, Populations),
  se_prior = matrix(1, ncol=2, nrow=2),
  sp_prior = matrix(1, ncol=2, nrow=2)
)
data$prev[1] <- 0
```

```r
inits <- list(
  chain1 = list(
    prev = c(NA, 0.1, 0.1, 0.9, 0.9),
    se = c(0.5, 0.99),
    sp = c(0.5, 0.99)
  ),
  chain2 = list(
    prev = c(NA, 0.9, 0.9, 0.1, 0.1),
    se = c(0.99, 0.5),
    sp = c(0.99, 0.5)
  )
)

results <- run.jags(..., data = data, inits = inits)
```

```
inits <- list(
  chain1 = list(
    prev = c(NA, 0.1, 0.1, 0.9, 0.9),
    se = c(0.5, 0.99),
    sp = c(0.5, 0.99)
  ),
  chain2 = list(
    prev = c(NA, 0.9, 0.9, 0.1, 0.1),
    se = c(0.99, 0.5),
    sp = c(0.99, 0.5)
  )
)

results <- run.jags(..., data = data, inits = inits)
```

See the help file for `?run.jags` for more details

Other runjags options

There are a large number of other options to runjags. Some highlights:

- The method can be `parallel` or `background` or `bgparallel`
- You can use `extend.jags` to continue running an existing model (e.g. to increase the sample size)
- You can use `coda::as.mcmc.list` to extract the underlying MCMC chains
- Use the `summary()` method to extract summary statistics
  - See `?summary.runjags` and `?runjagsclass` for more information

Setting the RNG seed

If we want to get numerically replicable results we need to add `.RNG.name` and `.RNG.seed` to the initial values, and an additional `#modules#` lecuyer hook to our model definition:

```
#inits# .RNG.name, .RNG.seed
#modules# lecuyer
```

Setting the RNG seed

If we want to get numerically replicable results we need to add `.RNG.name` and `.RNG.seed` to the initial values, and an additional `#modules#` lecuyer hook to our model definition:

```
  #inits# .RNG.name, .RNG.seed
  #modules# lecuyer
```

Then we can propogate R's RNG to JAGS like so:

```
set.seed(2021-06-29)
.RNG.name <<- "lecuyer::RngStream"
.RNG.seed <<- list(chain1=sample.int(1e6, 1), chain2=sample.int(1e6, 1)))
results <- run.jags(model_string, n.chains=2)
```

Setting the RNG seed

If we want to get numerically replicable results we need to add `.RNG.name` and `.RNG.seed` to the initial values, and an additional `#modules#` lecuyer hook to our model definition:

```
  #inits# .RNG.name, .RNG.seed
  #modules# lecuyer
```

Then we can propogate R's RNG to JAGS like so:

```
set.seed(2021-06-29)
.RNG.name <<- "lecuyer::RngStream"
.RNG.seed <<- list(chain1=sample.int(1e6, 1), chain2=sample.int(1e6, 1)))
results <- run.jags(model_string, n.chains=2)
```

- Every time this model is run the results will now be identical

Practical session 3

Points to consider

1. What are the benefits of including multiple populations?
2. How can we define/obtain these populations?
3. What happens if our fundamental assumptions about consistent Se/Sp are broken?

Summary

- Multiple populations helps to estimate Se and Sp
  - Particularly if the prevalences differ
  - A minimum of two populations is generally needed
- Populations may be artificial
  - But cannot be based on the result of either test
- But if Se / Sp are inconsistent then we will get misleading results
  - In practice, groups with widely varying prevalence rarely have consistent Se / Sp
  - It is possible to allow Se / Sp to differ between populations, but then there is no benefit of combining the data