Session 2

Basic Hui-Walter models

Matt Denwood

2021-06-28

Session 2: Basic Hui-Walter models

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
  - But remember to be realistic about what is possible with your data

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
    - But remember to be realistic about what is possible with your data
- So how do we extend these models to multiple diagnostic tests?

Hui-Walter Model

- A particular model formulation that was originally designed for evaluating diagnostic tests in the absence of a gold standard
- Not necessarily (or originally) Bayesian but often implemented using Bayesian MCMC
- But evaluating an imperfect test against another imperfect test is a bit like pulling a rabbit out of a hat
    - If we don't know the true disease status, how can we estimate sensitivity or specificity for either test?

Model Specification

```
model{
  Tally ~ dmulti(prob, TotalTests)

  # Test1- Test2-
    prob[1] <- (prev * ((1-se[1])*(1-se[2]))) + ((1-prev) *
    ↪  ((sp[1])*(sp[2])))

  # Test1+ Test2-
    prob[2] <- (prev * ((se[1])*(1-se[2]))) + ((1-prev) *
    ↪  ((1-sp[1])*(sp[2])))

  # Test1- Test2+
    prob[3] <- (prev * ((1-se[1])*(se[2]))) + ((1-prev) *
    ↪  ((sp[1])*(1-sp[2])))
```

```
  # Test1+ Test2+
    prob[4] <- (prev * ((se[1])*(se[2]))) + ((1-prev) *
    ↪  ((1-sp[1])*(1-sp[2])))

  prev ~ dbeta(1, 1)
  se[1] ~ dbeta(1, 1)
  sp[1] ~ dbeta(1, 1)
  se[2] ~ dbeta(1, 1)
  sp[2] ~ dbeta(1, 1)

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

```r
twoXtwo <- matrix(c(48, 12, 4, 36), ncol=2, nrow=2)
twoXtwo
##      [,1] [,2]
## [1,]   48    4
## [2,]   12   36

library('runjags')

Tally <- as.numeric(twoXtwo)
TotalTests <- sum(Tally)

prev <- list(chain1=0.05, chain2=0.95)
se <- list(chain1=c(0.5,0.99), chain2=c(0.99,0.5))
sp <- list(chain1=c(0.5,0.99), chain2=c(0.99,0.5))

results <- run.jags('basic_hw.bug', n.chains=2)
```

[Remember to check convergence and effective sample size!]

```
results
```

	Lower95	Median	Upper95	SSeff	psrf
prev	0.308	0.438	0.574	4088	1
prob[1]	0.367	0.462	0.557	14093	1
prob[2]	0.073	0.133	0.202	13744	1
prob[3]	0.017	0.055	0.104	9633	1
prob[4]	0.251	0.343	0.433	13634	1
se[1]	0.821	0.932	1.000	5809	1
se[2]	0.691	0.848	1.000	3278	1
sp[1]	0.744	0.875	1.000	3383	1
sp[2]	0.861	0.948	1.000	5741	1

- Note the wide confidence intervals!

Practicalities

- These models need A LOT of data, and/or strong priors for one of the tests
- Convergence is more problematic than usual
- Be **very** vareful with the order of combinations in dmultinom!
- Check your results carefully to ensure they make sense!

Label Switching

How to interpret a test with Se=0% and Sp=0%?

Label Switching

How to interpret a test with Se=0% and Sp=0%?

- The test is perfect - we are just holding it upside down...

Label Switching

How to interpret a test with Se=0% and Sp=0%?

- The test is perfect - we are just holding it upside down...

We can force se+sp >= 1:

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

Or:

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

But not both!

This allows the test to be useless, but not worse than useless.

Alternatively we can have the weakly informative priors:

```
se[1] ~ dbeta(2, 1)
```

Simulating data

Analysing simulated data is useful to check that we can recover parameter values.

```
se1 <- 0.9; sp1 <- 0.95;
se2 <- 0.8; sp2 <- 0.99
prevalence <- 0.5; N <- 100

truestatus <- rbinom(N, 1, prevalence)
Test1 <- rbinom(N, 1, (truestatus * se1) + ((1-truestatus) * (1-sp1)))
Test2 <- rbinom(N, 1, (truestatus * se2) + ((1-truestatus) * (1-sp2)))

twoXtwo <- table(Test1, Test2)
Tally <- as.numeric(twoXtwo)
```

Can we recover these parameter values?

Exercise

Modify the code in the Hui Walter model to force tests to be no worse than useless

Simulate data and recover parameters for:

- N=10
- N=100
- N=1000

Optional Exercise

Compare results with the following priors for test 1:

- Sensitivity = 0.9 (95% CI: 0.85 - 0.95)
- Specificity = 0.95 (95%CI: 0.92-0.97)

[These are the same as in session 1]

Practical Session 2

Session 1b: Working with basic models (apparent prevalence)

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

Using embedded character strings

- For simple models we might not want to bother with an external text file. Then we can do:

```
mt <- "
model{
  Positives ~ dbinom(prevalence, TotalTests)
  prevalence ~ dbeta(2, 2)

  #data# Positives, TotalTests
  #monitor# prevalence
  #inits# prevalence
}
"

results <- run.jags(mt, n.chains=2)
```

- But I would advise that you stick to using a separate text file!

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 basicjags.bug file:

```
model{
  Positives ~ dbinom(prevalence, TotalTests)
  prevalence ~ dbeta(2, 2)

  #data# Positives, TotalTests
  #monitor# prevalence
  #inits# prevalence, .RNG.name, .RNG.seed
  #modules# lecuyer
}
```

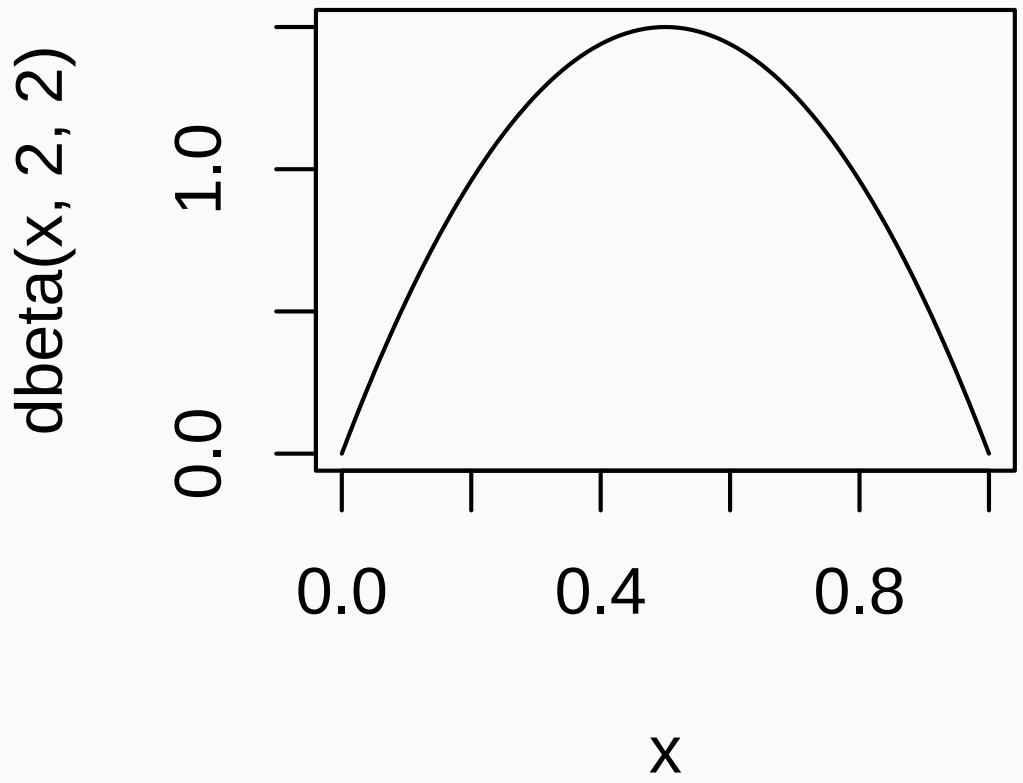
```
.RNG.name <- "lecuyer::RngStream"
.RNG.seed <- list(chain1=1, chain2=2)
results <- run.jags('basicjags.bug', n.chains=2)
```

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

A different prior

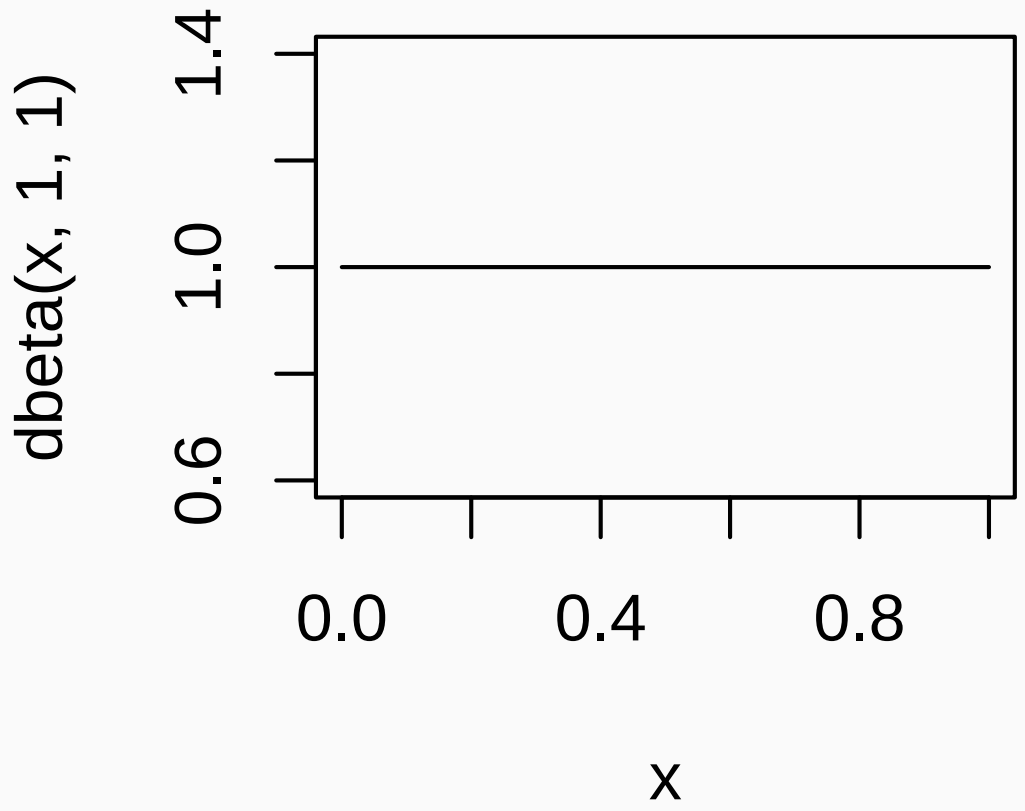
- A quick way to see the distribution of a prior:

```
curve(dbeta(x, 2, 2), from=0, to=1)
```



- A minimally informative prior might be:

```
curve(dbeta(x, 1, 1), from=0, to=1)
```



- Let's change the prior we are using to `dbeta(1,1)`:

```
model{
  Positives ~ dbinom(prevalence, TotalTests)
  prevalence ~ dbeta(1, 1)

  # Hooks for automatic integration with R:
  #data# Positives, TotalTests
  #monitor# prevalence
  #inits# prevalence
}
```

An Equivalent Model

- We could equivalently specify an observation-level model:

```
model{
  # Likelihood part:
  for(i in 1:TotalTests){
    Status[i] ~ dbern(prevalence)
  }

  # Prior part:
  prevalence ~ dbeta(1, 1)

  # Hooks for automatic integration with R:
  #data# Status, TotalTests
  #monitor# prevalence
  #inits# prevalence
}
```

- But we need the data in a different format: a vector of 0/1 rather than total positives!

```
Positives <- 70
TotalTests <- 100
Status <- c(rep(0, TotalTests-Positives), rep(1, Positives))
```

A GLM Model

```
model{
  # Likelihood part:
  for(i in 1:TotalTests){
    Status[i] ~ dbern(predicted[i])
    logit(predicted[i]) <- intercept
  }

  # Prior part:
  intercept ~ dnorm(0, 10^-6)

  # Derived parameter:
  prevalence <- ilogit(intercept)

  # Hooks for automatic integration with R:
  #data# Status, TotalTests
  #monitor# intercept, prevalence
  #inits# intercept
}
```

- This is the start of a generalised linear model, where we could add covariates at individual animal level.
- We introduce a new distribution `dnorm()` - notice this is mean and precision, not mean and sd!
- For a complete list of the distributions available see:
  - https://sourceforge.net/projects/mcmc-jags/files/Manuals/4.x/
  - This document is also provided on the GitHub repository
- However, notice that the prior is specified differently...

Exercise

- Run the original version, the observation-level version, and the GLM version of the model and compare results with the same data
- Now try a larger sample size: e.g. 70 positives out of 100 tests - are the posteriors from the two models more or less similar than before?
- Now try running the GLM model with a prior of `dnorm(0, 0.33)` (and the original data) - does this make a difference?

Optional Exercise

Another way of comparing different priors is to run different models with no data - as there is no influence of a likelihood, the posterior will then be identical to the priors (and the model will run faster).

One way to do this is to make all of the response data (i.e. either Positives or Status) missing. Try doing this for the following three models, and compare the priors for prevalence:

- The original model with prior `prevalence ~ dbeta(1,1)`
- The GLM model with prior `intercept ~ dnorm(0, 10^-6)`
- The GLM model with prior `intercept ~ dnorm(0, 0.33)`

Session 1c: Basics of latent-class models (imperfect test)

Imperfect tests

- Up to now we have ignored issues of diagnostic test sensitivity and specificity
- Usually, however, we do not have a perfect test, so we do not know how many are truly positive or truly negative, rather than just testing positive or negative.
- But we know that:

$$Prev_{obs} = (Prev_{true} \times Se) + ((1 - Prev_{true}) \times (1 - Sp))$$

$$\implies Prev_{true} = \frac{Prev_{obs} - (1 - Sp)}{Se - (1 - Sp)}$$

Model Specification

- We can incorporate the imperfect sensitivity and specicifity into our model:

```
model{
  Positives ~ dbinom(obsprev, TotalTests)
  obsprev <- (prevalence * se) + ((1-prevalence) * (1-sp))

  prevalence ~ dbeta(1, 1)
  se ~ dbeta(1, 1)
  sp ~ dbeta(1, 1)

  #data# Positives, TotalTests
  #monitor# prevalence, obsprev, se, sp
  #inits# prevalence, se, sp
}
```

- And run it:

```
prevalence <- list(chain1=0.05, chain2=0.95)
se <- list(chain1=0.5, chain2=0.99)
sp <- list(chain1=0.5, chain2=0.99)
Positives <- 70
TotalTests <- 100
results <- run.jags('basicimperfect.bug', n.chains=2)
## Compiling rjags model...
## Calling the simulation using the rjags method...
## Adapting the model for 1000 iterations...
## Burning in the model for 4000 iterations...
## Running the model for 10000 iterations...
## Simulation complete
## Calculating summary statistics...
## Calculating the Gelman-Rubin statistic for 4
## variables....
## Finished running the simulation
```

[Remember to check convergence and effective sample size!]

What do these results tell us?

	Lower95	Median	Upper95	SSeff	psrf
prevalence	0.003	0.502	0.941	1930	1.000
obsprev	0.605	0.694	0.780	20613	1.000
se	0.150	0.698	1.000	1449	1.000
sp	0.000	0.317	0.852	1446	1.001

What do these results tell us?

	Lower95	Median	Upper95	SSeff	psrf
prevalence	0.003	0.502	0.941	1930	1.000
obsprev	0.605	0.694	0.780	20613	1.000
se	0.150	0.698	1.000	1449	1.000
sp	0.000	0.317	0.852	1446	1.001

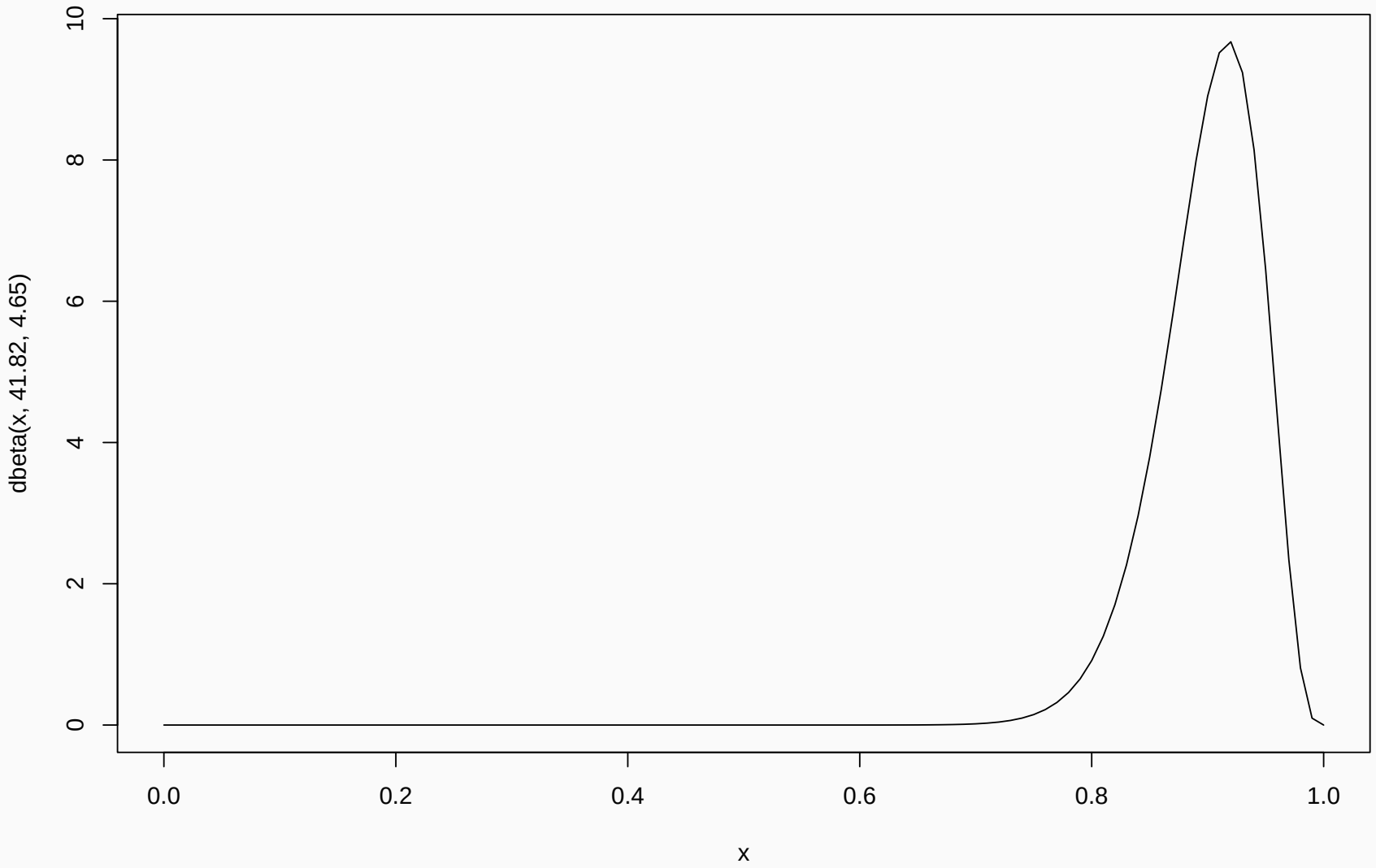
- We can estimate the observed prevalence quite well
- But not the prevalence, se or sp!
  - The model is unidentifiable.

Priors

- We cannot estimate `se`, `sp` and `prevalence` simultaneously
  - We need strong priors for se and sp
- We can use the PriorGen package to generate Beta priors based on published results, for example:

```
PriorGen::findbeta(themean=0.9, percentile = 0.975, percentile.value =
↪  0.8)
## [1] "The desired Beta distribution that satisfies the specified
↪  conditions is: Beta( 41.82 4.65 )"
## [1] "Here is a plot of the specified distribution."
## [1] "Descriptive statistics for this distribution are:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.6559  0.8753  0.9072  0.9008  0.9328  0.9939
## [1] "Verification: The percentile value 0.8 corresponds to the 0.025
↪  th percentile"
```

```r
qbeta(c(0.025, 0.5, 0.975), 41.82, 4.65)
## [1] 0.7999144 0.9056630 0.9677283
curve(dbeta(x, 41.82, 4.65), from=0, to=1)
```



Exercise

- Find beta distribution priors for:
  - Sensitivity = 0.9 (95% CI: 0.85 - 0.95)
  - Specificity = 0.95 (95%CI: 0.92-0.97)
- Look at these distributions using curve and qbeta
- Modify the imperfect test model using these priors and re-estimate prevalence

Optional Exercise

- Run the same model with se and sp fixed to the mean estimate
    - How does this affect CI for prevalence?
- Run the same model with se and sp fixed to 1
    - How does this affect estimates and CI for prevalence?

Summary

- Using JAGS / runjags allows us to work with MCMC more easily, safely and efficiently than writing our own sampling algorithms
- But we must *never forget* to check convergence and effective sample size!
- More complex models become easy to implement
    - For example imperfect diagnostic tests
- But just because a model can be defined does not mean that it will be useful for our data
    - We need to be realistic about the information available in the data, what parameters are feasible to estimate, and where we will need to use strong priors