Session 7

Incorporating imperfect sensitivity and specificity into more complex models

Matt Denwood

2021-07-01

Recap

Outline

- Logistic regression models can include Se and Sp
- We can do this either at population level or individual level
- Sometimes it matters, particularly if we have two tests, or if the tests differ between populations
- Usually it doesn't make any difference except to the prevalence
- Flexibility of MCMC means we are free to do many different things

Incorporating imperfect sensitivity and specificity into more complex models

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
}
```

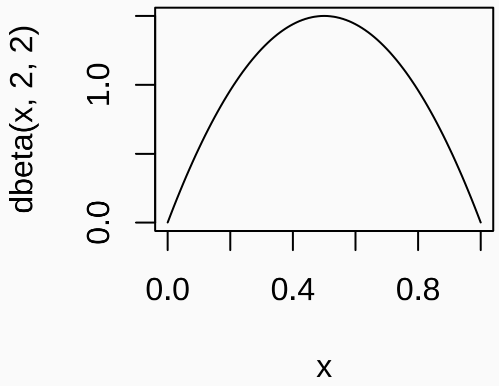
```
.RNG.name <- "lecuyer::RngStream"
.RNG.seed <- list(chain1=1, chain2=2)
results <- run.jags('basicjags.bug', n.chains=2)
```

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

A different prior

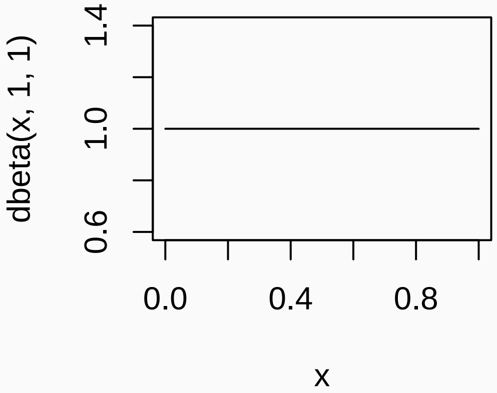
- A quick way to see the distribution of a prior:

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



- A minimally informative prior might be:

```r
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!

```
TotalTests <- 100
Positives <- 70
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?

Population-level model specification

We might want to explicitly model the latent state:

```
model{

  for(i in 1:N){
    truestatus[i] ~ dbern(prev[Population[i]])

    Status[i] ~ dcat(prob[1:8, i])
    prob[1:8,i] <- se_prob[1:8,i] + sp_prob[1:8,i]

        se_prob[1,i] <- truestatus[i] * ((1-se[1])*(1-se[2])*(1-se[3]))
        sp_prob[1,i] <- (1-truestatus[i]) * (sp[1]*sp[2]*sp[3])

        se_prob[2,i] <- truestatus[i] * (se[1]*(1-se[2])*(1-se[3]))
        sp_prob[2,i] <- (1-truestatus[i]) * ((1-sp[1])*sp[2]*sp[3])
```

```
se_prob[3,i] <- truestatus[i] * ((1-se[1])*se[2]*(1-se[3]))
sp_prob[3,i] <- (1-truestatus[i]) * (sp[1]*(1-sp[2])*sp[3])

se_prob[4,i] <- truestatus[i] * (se[1]*se[2]*(1-se[3]))
sp_prob[4,i] <- (1-truestatus[i]) * ((1-sp[1])*(1-sp[2])*sp[3])

se_prob[5,i] <- truestatus[i] * ((1-se[1])*(1-se[2])*se[3])
sp_prob[5,i] <- (1-truestatus[i]) * (sp[1]*sp[2]*(1-sp[3]))

se_prob[6,i] <- truestatus[i] * (se[1]*(1-se[2])*se[3])
sp_prob[6,i] <- (1-truestatus[i]) * ((1-sp[1])*sp[2]*(1-sp[3]))

se_prob[7,i] <- truestatus[i] * ((1-se[1])*se[2]*se[3])
sp_prob[7,i] <- (1-truestatus[i]) * (sp[1]*(1-sp[2])*(1-sp[3]))
```

```
      se_prob[8,i] <- truestatus[i] * (se[1]*se[2]*se[3])
      sp_prob[8,i] <- (1-truestatus[i]) *
      ↪  ((1-sp[1])*(1-sp[2])*(1-sp[3]))
  }

    prev[1] ~ dbeta(1,1)
    prev[2] ~ 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)
  se[3] ~ dbeta(1, 1)T(1-sp[3], )
  sp[3] ~ dbeta(1, 1)

  #data# Status, N, Population
  #monitor# prev, se, sp, truestatus[1:5]
  #inits# prev, se, sp
}
```

```
Population <- simdata$Population
Status <- with(simdata, factor(interaction(Test1, Test2, Test3),
↪  levels=c('0.0.0','1.0.0','0.1.0','0.0.1','1.1.0','1.0.1','0.1.1','1.1.1')))

prev <- list(chain1=c(0.05,0.95), chain2=c(0.95,0.05))
se <- list(chain1=c(0.5,0.75,0.99), chain2=c(0.99,0.5,0.75))
sp <- list(chain1=c(0.5,0.75,0.99), chain2=c(0.99,0.5,0.75))
```

```
results <- run.jags('glm_hw3t.bug', n.chains=2)
```

```
results
```

But this is inefficient

- Time taken is 1.6 minutes rather than a few seconds
- And the barely stochastic nature of some truestatus estimates triggers false convergence warnings
- And there is no way to distinguish individuals within the same boxes anyway, as they have the same data!

But this is inefficient

- Time taken is 1.6 minutes rather than a few seconds
- And the barely stochastic nature of some truestatus estimates triggers false convergence warnings
- And there is no way to distinguish individuals within the same boxes anyway, as they have the same data!

It is much better to use the estimated se/sp/prev to post-calculate these truestatus probabilities

- This can be useful for post-hoc ROC

Model selection

- Choosing between candidate models
    - DIC
    - Bayes Factors
    - BIC
    - WAIC
    - Effect size spans zero?

Model selection

- Choosing between candidate models
    - DIC
    - Bayes Factors
    - BIC
    - WAIC
    - Effect size spans zero?
- Assessing model adequacy:
    - Verify using a simulation study
    - Posterior predictive p-values
    - Comparison of results from different models eg:
        - Independence vs covariance
        - Different priors

Model selection

- Choosing between candidate models
  - DIC
  - Bayes Factors
  - BIC
  - WAIC
  - Effect size spans zero?
- Assessing model adequacy:
  - Verify using a simulation study
  - Posterior predictive p-values
  - Comparison of results from different models eg:
    - Independence vs covariance
    - Different priors

Others?

DIC and WAIC

- DIC
    - Works well for hierarchical normal models
    - To calculate:
        - Add dic and ped to the monitors in runjags
        - But be cautious with these types of models

DIC and WAIC

- DIC
  - Works well for hierarchical normal models
  - To calculate:
    - Add dic and ped to the monitors in runjags
    - But be cautious with these types of models
- WAIC
  - Approximation to LOO
  - Needs independent likelihoods
    - Could work for individual-level models?
  - Currently a pain to calculate
    - See WAIC.R in the GitHub directory
    - And/or wait for updates to runjags (and particularly JAGS 5)

Some advice

- Always start by simulating data and verifying that you can recover the parameters
  - The simulation can be more complex than the model!
  - See the autorun.jags function
- If you have different candidate models then compare the posteriors between models

Some advice

- Always start by simulating data and verifying that you can recover the parameters
    - The simulation can be more complex than the model!
    - See the autorun.jags function
- If you have different candidate models then compare the posteriors between models
- A particular issue is test dependence
    - Is there biological justification for the correlation?
    - Are the test sensitivity/specificity estimates consistent?
    - Do the covse / covsp estimates overlap zero?

Some advice

- Always start by simulating data and verifying that you can recover the parameters
    - The simulation can be more complex than the model!
    - See the autorun.jags function
- If you have different candidate models then compare the posteriors between models
- A particular issue is test dependence
    - Is there biological justification for the correlation?
    - Are the test sensitivity/specificity estimates consistent?
    - Do the covse / covsp estimates overlap zero?
- Any other good advice?!?

Observation-level model specification

```
model{

  for(i in 1:N){
    Status[i] ~ dcat(prob[i, ])

      prob[i,1] <- (prev[i] * ((1-se[1])*(1-se[2]))) +
                  ((1-prev[i]) * ((sp[1])*(sp[2])))
      prob[i,2] <- (prev[i] * ((se[1])*(1-se[2]))) +
                  ((1-prev[i]) * ((1-sp[1])*(sp[2])))
      prob[i,3] <- (prev[i] * ((1-se[1])*(se[2]))) +
                  ((1-prev[i]) * ((sp[1])*(1-sp[2])))
      prob[i,4] <- (prev[i] * ((se[1])*(se[2]))) +
                  ((1-prev[i]) * ((1-sp[1])*(1-sp[2])))

      logit(prev[i]) <- intercept + population_effect[Population[i]]
  }
  intercept ~ dnorm(0, 0.33)
  population_effect[1] <- 0
  for(p in 2:Pops){
    population_effect[p] ~ dnorm(0, 0.1)
  }
  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# Status, N, Population, Pops
  #monitor# intercept, population_effect, se, sp
  #inits# intercept, population_effect, se, sp
}
```

- The main difference is the prior for prevalence in each population
- We also need to give initial values for `intercept` and `population_effect` rather than `prev`, and tell `run.jags` the data frame from which to extract the data (except `N` and `Pops`):

```r
intercept <- list(chain1=-1, chain2=1)
population_effect <- list(chain1=c(NA, 1, -1), chain2=c(NA, -1, 1))
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))

simdata$Status <- with(simdata, factor(interaction(Test1, Test2),
↪  levels=c('0.0','1.0','0.1','1.1')))
N <- nrow(simdata)
Pops <- length(levels(simdata$Population))
glm_results <- run.jags('glm_hw.bug', n.chains=2, data=simdata)
```

Also like in session 1, the estimates for se/sp should be similar, although this model runs more slowly.

Note: this model could be used as the basis for adding covariates

For a handy way to generate a GLM model see runjags::template.jags

- Look out for integration with autohuiwalter in the near (ish) future…

Exercise

Play around with the autohuiwalter function

Notice the model and data and initial values are in a self contained file

Ignore the covse and covsp for now

[There is no particular solution to this exercise!]

Summary

- Estimating sensitivity and specificity is like pulling a rabbit out of a hat
- Multiple populations helps **a lot**
- Strong priors for one of the tests helps even more!
- Make sure you tabulate the data correctly . . . or use the automated model generator!

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)`

Practical session 7

Points to consider

1. When is there a benefit of adding imperfect test information to a LR, and when is there no benefit?
2. How does specifying priors on logistic scale vs prevalence scale impact estimates?
3. What other models could you adapt using similar methods?

Summary

- Imperfect diagnostic test models are not limited to the Hui-Walter paradigm
  - Direct equivalence in Hidden Markov Models (e.g. SIR models with imperfect detection)
- Bayesian methods (particularly MCMC) give almost unlimited flexibility to adapt other models
  - Most obviously those involving some form of logsitic regression
- But bear in mind that there is a balance between complexity and parsimony!