Hands-on training session 2

Hui-Walter models for diagnostic test evaluation

Matt Denwood Giles Innocent

2020-02-18

Introduction

Overview

Date/time:

- 19th February 2020
- 16.00 - 17.00

Teachers:

- Matt Denwood (presenter)
- Giles Innocent

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?

Session 2a: Hui-Walter models for 2 tests and 1 population

Hui-Walter Model

TODO

Background (not necessarily Bayesian)

Rabbits and hats

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])))
```

```
1
2     prev ~ dbeta(1, 1)
3     se[1] ~ dbeta(1, 1)
4     sp[1] ~ dbeta(1, 1)
5     se[2] ~ dbeta(1, 1)
6     sp[2] ~ dbeta(1, 1)
7
8     #data# Tally, TotalTests
9     #monitor# prev, prob, se, sp
10    #inits# prev, se, sp
11  }
```

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

```
##      [,1] [,2]
## [1,]   48    4
## [2,]   12   36
```

```r
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)
```

```
## Warning: You should update the rjags package to version 5.x -
↪  if the version on CRAN is not currently up to date, try
↪  downloading from
↪  https://sourceforge.net/projects/mcmc-jags/files/rjags/
↪  instead
```

```
results
```

	Lower95	Median	Upper95	SSeff	psrf
prev	0.306	0.442	0.574	3860	1.000
prob[1]	0.366	0.461	0.558	13086	1.000
prob[2]	0.072	0.133	0.201	14066	1.000
prob[3]	0.019	0.055	0.104	9199	1.000
prob[4]	0.253	0.344	0.438	12555	1.000
se[1]	0.823	0.931	1.000	5625	1.000
se[2]	0.689	0.847	1.000	3293	1.000
sp[1]	0.745	0.877	1.000	3204	1.000
sp[2]	0.862	0.948	1.000	5311	1.001

- Note the wide confidence intervals!

Practicalities

TODO

Care with order of combinations in dmultinom

Lots of data needed

- And/or strong priors for one of the tests

Convergence can be tricky

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

Simulating data

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

Some simultion code:

```
se1 <- 0.9
sp1 <- 0.95
sp2 <- 0.99
se2 <- 0.8
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)
twoXtwo
```

Exercise

Modify JAGS code to force tests to be better than useless

Simulate data and recover parameters for:

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

Optional Exercise

Use priors for test1 taken from session 1 and compare the results

Solution

Model definition:

```
1  model{
2    Tally ~ dmulti(prob, TotalTests)
3
4    # Test1- Test2-
5      prob[1] <- (prev * ((1-se[1])*(1-se[2]))) + ((1-prev) *
       ↪  ((sp[1])*(sp[2])))
6
7    # Test1+ Test2-
8      prob[2] <- (prev * ((se[1])*(1-se[2]))) + ((1-prev) *
       ↪  ((1-sp[1])*(sp[2])))
9
10   # Test1- Test2+
11     prob[3] <- (prev * ((1-se[1])*(se[2]))) + ((1-prev) *
       ↪  ((sp[1])*(1-sp[2])))
12
13   # Test1+ Test2+
14     prob[4] <- (prev * ((se[1])*(se[2]))) + ((1-prev) *
       ↪  ((1-sp[1])*(1-sp[2])))
15
```

Optional Solution

```
HPSe[1,] <- c(148.43, 16.49)
HPSp[1,] <- c(240.03, 12.63)

HPSe
```

```
##         [,1]  [,2]
## [1,] 148.43 16.49
## [2,]   1.00  1.00
```

```
HPSp
```

```
##         [,1]  [,2]
## [1,] 240.03 12.63
## [2,]   1.00  1.00
```

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

```
## Finished running the simulation
```



```
results
```

Session 2b: Hui-Walter models for 2 tests and N populations

Independent intercepts for 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])))

    ## etc ##

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

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

  #data# Tally, TotalTests, Populations, HPSe, HPSp
  #monitor# prev, prob, se, sp
```

Auto Hui-Walter

We would usually start with individual-level data in a dataframe

e.g.:

```
se1 <- 0.9
sp1 <- 0.95
sp2 <- 0.99
se2 <- 0.8
prevalences <- c(0.1, 0.5, 0.9)
N <- 100

simdata <- data.frame(Population = sample(seq_along(prevalences),
↪  N, replace=TRUE))
simdata$probability <- prevalences[simdata$Population]
simdata$truestatus <- rbinom(N, 1, simdata$probability)
simdata$Test1 <- rbinom(N, 1, (simdata$truestatus * se1) +
↪  ((1-simdata$truestatus) * (1-sp1)))
simdata$Test2 <- rbinom(N, 1, (simdata$truestatus * se2) +
↪  ((1-simdata$truestatus) * (1-sp2)))

head(simdata)
```

The model code and data format for an arbitrary number of populations (and tests) can be determined automatically

There is a function (soon to be included in the runjags package, but for now provided in the GitHub repo) that can do this for us:

```
simdata$Population <- factor(simdata$Population,
↪  levels=seq_along(prevalences), labels=paste0('Pop_',
↪  seq_along(prevalences)))

source("autohuiwalter.R")
auto_huiwalter(simdata[,c('Population','Test1','Test2')],
↪  outfile='autohw.bug')
```

```
## The model and data have been written to autohw.bug in the
↪  current working directory
## You should check and alter priors before running the model
```

This generates self-contained model/data/initial values etc (ignore covse and covsp for now):

```
## ## Auto-generated Hui-Walter model created by script version 0.1 on 2020-02-18
##
## model{
##
##  ## Observation layer:
##
##  # Complete observations (N=100):
##  for(p in 1:Populations){
##      Tally_RR[1:4,p] ~ dmulti(prob_RR[1:4,p], N_RR[p])
##
##      prob_RR[1:4,p] <- se_prob[1:4,p] + sp_prob[1:4,p]
##  }
##
##
##  ## Observation probabilities:
##
##  for(p in 1:Populations){
##
##      # Probability of observing Test1- Test2- from a true
```

And can be run directly from R:

```
results <- run.jags('autohw.bug')
```

```
## Note: The monitored variables 'covse12' and 'covsp12'
## appear to be non-stochastic; they will not be
## included in the convergence diagnostic
## Finished running the simulation
```

```
results
```

```
##
## JAGS model summary statistics from 20000 samples (chains = 2;
↪  adapt+burnin = 5000):
##
##           Lower95   Median Upper95     Mean       SD
## se[1]     0.87189  0.96816       1  0.95577 0.042197
## se[2]     0.60525  0.76164 0.89516  0.75667  0.07517
## sp[1]     0.92611   0.9804       1  0.97362  0.02416
## sp[2]     0.88409  0.94591 0.99308   0.9417 0.029668
## prev[1] 0.0069498 0.069215 0.16152 0.076369 0.043371
## prev[2]   0.18076  0.32083 0.46915  0.32322 0.075138
## prev[3]    0.7327  0.88449 0.99128  0.87414 0.072603
## covse12         0        0       0        0        0
## covsp12         0        0       0        0        0
##
##              Mode      MCerr MC%ofSD SSeff       AC.10
## se[1]     0.98579 0.00066739     1.6  3998    0.015552
## se[2]     0.76962 0.00071055     0.9 11192    0.014574
## sp[1]     0.99125 0.00035858     1.5  4540   0.0054374
## sp[2]      0.9504  0.0003137     1.1  8944  -0.0064593
## prev[1] 0.057135 0.00049806     1.1  7583 -0.00044204
```

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){
```

Just like in session 1, the main difference is the prior for prevalence (this time 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):

```
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)
```

```
## Note: The monitored variable 'population_effect[1]'
## appears to be non-stochastic; it will not be included
## in the convergence diagnostic
```

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. . .

Practicalities

Need to be very careful with tabulating the data, or use automatically generated code

Works best when populations have very different prevalences

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