Hands-on training session 2

Hui-Walter models for diagnostic test evaluation

Matt Denwood Giles Innocent

2020-02-12

Introduction

Overview

Date/time:

- 19th February 2020
- 16.00 - 17.00

Teachers:

- Matt Denwood (presenter)
- Giles Innocent

Recap

Important points from session 1

TODO

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

Hui-Walter Model

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])))

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

- And run it:

```
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)
```

```
## Loading required namespace: rjags
```

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

```
results
```

```
##
## JAGS model summary statistics from 20000 samples (chains = 2;
↪  adapt+burnin = 5000):
##
##               Lower95   Median Upper95     Mean       SD
## prev          0.32325  0.50098 0.66389  0.50026 0.089659
## prob[1]        0.3685   0.4616  0.5589  0.46192 0.048563
## prob[2]       0.07425  0.13211 0.20313  0.13476 0.033506
## prob[3]      0.017845 0.055363 0.10459 0.058274 0.023254
## prob[4]       0.25536  0.34443 0.43803  0.34505 0.046808
## se[1]         0.02658  0.51864 0.99989  0.52427  0.40551
## se[2]     0.000056421  0.40028 0.96364  0.45055   0.3968
## sp[1]      0.00003794  0.48834 0.97274  0.47625  0.40561
## sp[2]        0.037854  0.60532       1  0.54893  0.39667
##
##               Mode      MCerr MC%ofSD SSeff      AC.10
## prev       0.50647  0.0013864     1.5  4182   0.040909
## prob[1]    0.46235 0.00040627     0.8 14288  -0.006429
## prob[2]    0.12804 0.00028699     0.9 13630   0.010191
## prob[3]   0.050455 0.00024013       1  9377 -0.0059064
## prob[4]    0.34212   0.000403     0.9 13491 -0.0020015
```

	Lower95	Median	Upper95	SSeff	psrf
prev	0.323	0.501	0.664	4182	2.248
prob[1]	0.369	0.462	0.559	14288	1.000
prob[2]	0.074	0.132	0.203	13630	1.000
prob[3]	0.018	0.055	0.105	9377	1.000
prob[4]	0.255	0.344	0.438	13491	1.000
se[1]	0.027	0.519	1.000	4706	15.197
se[2]	0.000	0.400	0.964	4341	13.471
sp[1]	0.000	0.488	0.973	4302	15.219
sp[2]	0.038	0.605	1.000	4159	13.633

- Note wide confidence intervals

Practicalities

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%?

...

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)
```

Simulating data

How to simulate data for this and checking we can recover parameter values

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

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

twoXtwo <- table(Test1, Test2)
twoXtwo
```

```
##       Test2
## Test1  0  1
##      0 45  5
```

Exercise

Modify code to force tests to be better than useless

Simulate data and recover parameters

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

Optional Exercise

Use priors for test1 taken from session 1 and look again at the results

Solution

Model definition:

```
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])))

```

Optional Solution

```r
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
```

```r
HPSp
```

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

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

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



```r
results
```

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

Model specification 1

Independent intercepts for populations (standard)

Model specification 2

GLM-style with fixed effects of populations

Or random effects of populations

- Or covariates

Note it runs slower

Practicalities

Need to be very careful with tabulating the data

Works best when populations have very different prevalences

Auto Hui-Walter

Show autohuiwalter.R

Disable correlations by default

Modify so it allows Se/Sp priors to be defined as matrices?

And correlations on/off as matrices?

Will be in runjags at some point

Add force tests to be better than useless

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

simdata <- data.frame(
  Population = rep(seq_along(prevalences), each=N)
```

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

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

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

What would be useful to add to the function?

Summary