Hands-on training session 3

Hui-Walter models with more than two diagnostic tests

Matt Denwood Giles Innocent Sonja Hartnack

2020-02-19

Introduction

Overview

Date/time:

- 20th February 2020
- 14.00 - 15.30

Teachers:

- Matt Denwood (presenter)
- Giles Innocent
- Sonja Hartnack

Recap

- JAGS / runjags is the easy way to work with complex models
    - But we *still have to* check convergence and effective sample size!
- 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!

Recap

- JAGS / runjags is the easy way to work with complex models
  - But we *still have to* check convergence and effective sample size!
- 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!
- But what if the tests are not independent of each other?

Session 3a: Hui-Walter models for multiple conditionally independent tests

What exactly is our latent class?

- What do we mean by “conditionally independent?”

What exactly is our latent class?

- What do we mean by “conditionally independent?”
- Example: we have three antibody tests
  - The latent status is actually ‘producing antibodies’ not ‘diseased’

What exactly is our latent class?

- What do we mean by “conditionally independent?”
- Example: we have three antibody tests
  - The latent status is actually ‘producing antibodies’ not ‘diseased’
- Example: antibody vs egg count tests for liver fluke
  - Does the latent state include migrating juvenile fluke?

What exactly is our latent class?

- What do we mean by “conditionally independent?”
- Example: we have three antibody tests
  - The latent status is actually ‘producing antibodies’ not ‘diseased’
- Example: antibody vs egg count tests for liver fluke
  - Does the latent state include migrating juvenile fluke?
- We’re actually pulling **something** out of a hat, and deciding to call it a rabbit

Simulating data

Simulating data using an arbitrary number of independent tests is quite straightforward.

```r
# Parameter values to simulate:
N <- 200
se1 <- 0.8
sp1 <- 0.95
se2 <- 0.9
sp2 <- 0.99
se3 <- 0.95
sp3 <- 0.95

Populations <- 2
prevalence <- c(0.25,0.75)
Group <- sample(1:Populations, N, replace=TRUE)
```

```r
# Ensure replicable data:
set.seed(2020-02-18)

# Simulate the true latent state (which is unobserved in real
↪  life):
true <- rbinom(N, 1, prevalence[Group])
# Simulate test results for test 1:
test1 <- rbinom(N, 1, se1*true + (1-sp1)*(1-true))
# Simulate test results for test 2:
test2 <- rbinom(N, 1, se2*true + (1-sp2)*(1-true))
# Simulate test results for test 3:
test3 <- rbinom(N, 1, se3*true + (1-sp3)*(1-true))

simdata <- data.frame(Population=factor(Group), Test1=test1,
↪  Test2=test2, Test3=test3)
```

Model specification

- Like for two tests, except it is now a 2x2x2 table
  - If calculating this manually, take **extreme** care with multinomial tabulation
- Or use autohuiwalter
  - This will also deal gracefully with missing data in one or more test results

```r
source("autohuiwalter.R")
auto_huiwalter(simdata[,c('Population','Test1','Test2','Test3')],
↪  outfile='auto3thw.bug')
```

```
for(p in 1:Populations){
Tally_RRR[1:8,p] ~ dmulti(prob_RRR[1:8,p], N_RRR[p])
prob_RRR[1:8,p] <- se_prob[1:8,p] + sp_prob[1:8,p]

# Probability of observing Test1- Test2- Test3- from a true positive::
se_prob[1,p] <- prev[p] * ((1-se[1])*(1-se[2])*(1-se[3]) +covse12 +covse13 +covse23)
# Probability of observing Test1- Test2- Test3- from a true negative::
sp_prob[1,p] <- (1-prev[p]) * (sp[1]*sp[2]*sp[3] +covsp12 +covsp13 +covsp23)

. . .

# Probability of observing Test1+ Test2+ Test3+ from a true positive::
se_prob[8,p] <- prev[p] * (se[1]*se[2]*se[3] +covse12 +covse13 +covse23)
# Probability of observing Test1+ Test2+ Test3+ from a true negative::
sp_prob[8,p] <- (1-prev[p]) * ((1-sp[1])*(1-sp[2])*(1-sp[3]) +covsp12 +covsp13 +covsp23)
}
```

Alternative 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[i, ])
    prob[1:8,i] <- se_prob[1:8,i] + sp_prob[1:8,i]
```

```

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

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

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

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

        se_prob[5,p] <- truestatus[i] *
        ↪  ((1-se[1])*(1-se[2])*se[3])
        sp_prob[5,p] <- (1-truestatus[i]) *
        ↪  (sp[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
  #inits# prev, se, sp
}
```

But this is inefficient

There is also no way to distinguish individuals within the same boxes

We could also use the estimated se/sp/prev to post-calculate these status probabilities

This is useful for post-hoc ROC

Exercise

Simulate data from 3 tests and analyse using the autohuiwalter function

Do the estimates of Se/Sp correspond to the simulation parameters?

Make some data missing for one or more tests and re-generate the model

- Can you see what has changed in the code?

Optional Exercise

Simulate data from 3 antibody tests with ab positive step [give code in solution]

Does the se/sp estimated by a model recover the parameters?

Why not?

Session 3b: Hui-Walter models for multiple tests with conditional depdendence

Branching of processes leading to test results

Example: two antibody tests and one antigen test

Or three antibody tests where one has a different target to the other two

Simulating data

It helps to consider the data simulation as a biological process.

```r
# Parameter values to simulate:
N <- 200
se1 <- 0.8; sp1 <- 0.95
se2 <- 0.9; sp2 <- 0.99
se3 <- 0.95; sp3 <- 0.95

Populations <- 2
prevalence <- c(0.25,0.75)
Group <- rep(1:Populations, each=N)

# Ensure replicable data:
set.seed(2017-11-21)

# The probability of an antibody response given disease:
abse <- 0.8
# The probability of no antibody response given no disease:
absp <- 1 - 0.2
```

```r
# Simulate the true latent state:
true <- rbinom(N*Populations, 1, prevalence[Group])
# Simulate test results for test 1:
test1 <- rbinom(N*Populations, 1, se1*true + (1-sp1)*(1-true))
# Tests 2 & 3 will be co-dependent on antibody response:
antibody <- rbinom(N*Populations, 1, abse*true +
↪  (1-absp)*(1-true))
# Simulate test 2 & 3 results based on this other latent state:
test2 <- rbinom(N*Populations, 1, se2*antibody +
↪  (1-sp2)*(1-antibody))
test3 <- rbinom(N*Populations, 1, se3*antibody +
↪  (1-sp3)*(1-antibody))

ind3tests <- data.frame(Population=Group, Test1=test1,
↪  Test2=test2, Test3=test3)
```

```r
# The overall sensitivity of the correlated tests is:
abse*se2 + (1-abse)*(1-sp2)
```

```
## [1] 0.722
```

```r
abse*se3 + (1-abse)*(1-sp3)
```

```
## [1] 0.77
```

```r
# The overall specificity of the correlated tests is:
absp*sp2 + (1-absp)*(1-se2)
```

```
## [1] 0.812
```

```r
absp*sp3 + (1-absp)*(1-se3)
```

```
## [1] 0.77
```

```r
# The overall sensitivity of the correlated tests is:
abse*se2 + (1-abse)*(1-sp2)
```

```
## [1] 0.722
```

```r
abse*se3 + (1-abse)*(1-sp3)
```

```
## [1] 0.77
```

```r
# The overall specificity of the correlated tests is:
absp*sp2 + (1-absp)*(1-se2)
```

```
## [1] 0.812
```

```r
absp*sp3 + (1-absp)*(1-se3)
```

```
## [1] 0.77
```

We need to think carefully about what we are conditioning on when interpreting sensitivity and specificity!

Model specification

```
    se_prob[1,p] <- prev[p] * ((1-se[1])*(1-se[2])*(1-se[3])
↪   +covse12 +covse13 +covse23)
    sp_prob[1,p] <- (1-prev[p]) * (sp[1]*sp[2]*sp[3] +covsp12
↪   +covsp13 +covsp23)

    se_prob[2,p] <- prev[p] * (se[1]*(1-se[2])*(1-se[3]) -covse12
↪   -covse13 +covse23)
    sp_prob[2,p] <- (1-prev[p]) * ((1-sp[1])*sp[2]*sp[3] -covsp12
↪   -covsp13 +covsp23)

    ...

    # Covariance in sensitivity between tests 1 and 2:
    covse12 ~ dunif( (se[1]-1)*(1-se[2]) , min(se[1],se[2]) -
↪   se[1]*se[2] )
    # Covariance in specificity between tests 1 and 2:
    covsp12 ~ dunif( (sp[1]-1)*(1-sp[2]) , min(sp[1],sp[2]) -
↪   sp[1]*sp[2] )

    ...
```

Generating the model

Use autohuiwalter with argument covon=TRUE

```r
source('autohuiwalter.R')
auto_huiwalter(ind3tests, 'auto3tihw.bug', covon=TRUE)
```

```
# Covariance in sensitivity between Test1 and Test2 tests:
covse12 ~ dunif( (se[1]-1)*(1-se[2]) , min(se[1],se[2]) -
↪  se[1]*se[2] )  ## if the sensitivity of these tests may be
↪  correlated
#  covse12 <- 0  ## if the sensitivity of these tests can be
↪  assumed to be independent
# Covariance in specificity between Test1 and Test2 tests:
covsp12 ~ dunif( (sp[1]-1)*(1-sp[2]) , min(sp[1],sp[2]) -
↪  sp[1]*sp[2] )  ## if the specificity of these tests may be
↪  correlated
#  covsp12 <- 0  ## if the specificity of these tests can be
↪  assumed to be independent
```

```
# Covariance in sensitivity between Test1 and Test3 tests:
covse13 ~ dunif( (se[1]-1)*(1-se[3]) , min(se[1],se[3]) - se[1]*se[3] )  ## if the sensitivity of these tests may be correlated
#  covse13 <- 0  ## if the sensitivity of these tests can be assumed to be independent
# Covariance in specificity between Test1 and Test3 tests:
covsp13 ~ dunif( (sp[1]-1)*(1-sp[3]) , min(sp[1],sp[3]) - sp[1]*sp[3] )  ## if the specificity of these tests may be correlated
#  covsp13 <- 0  ## if the specificity of these tests can be assumed to be independent

# Covariance in sensitivity between Test2 and Test3 tests:
covse23 ~ dunif( (se[2]-1)*(1-se[3]) , min(se[2],se[3]) - se[2]*se[3] )  ## if the sensitivity of these tests may be correlated
#  covse23 <- 0  ## if the sensitivity of these tests can be assumed to be independent
# Covariance in specificity between Test2 and Test3 tests:
covsp23 ~ dunif( (sp[2]-1)*(1-sp[3]) , min(sp[2],sp[3]) - sp[2]*sp[3] )  ## if the specificity of these tests may be correlated
#  covsp23 <- 0  ## if the specificity of these tests can be
```

Exercise

Simulate data with a dependence between 2 tests

Model assuming conditional independence biases the estimates

Turn on covariance between the two tests

Model with conditional depdendence has bigger CI but unbiased

Optional Exercise

Activate covariance between all 3 tests

Session 3c: Model selection

Motivation

- Choosing between candidate models
  - DIC
  - Bayes Factors
  - BIC
  - WAIC

Motivation

- Choosing between candidate models
  - DIC
  - Bayes Factors
  - BIC
  - WAIC

- Assessing model adequacy:
  - Verify using a simulation study
  - Posterior predictive p-values
  - Comparison of results from different models eg:
    - Independence vs covariance
    - Different priors

Motivation

- Choosing between candidate models
  - DIC
  - Bayes Factors
  - BIC
  - WAIC
- 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? Discussion!

DIC and WAIC

- DIC
  - Works well for hierarchical normal models
  - To calculate:
    - Add dic and ped to the monitors in runjags
    - But don't trust the results for 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)

DIC and WAIC

- DIC
    - Works well for hierarchical normal models
    - To calculate:
        - Add dic and ped to the monitors in runjags
        - But don't trust the results for 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)

```
install.packages('runjags',
↪  repos=c("https://ku-awdc.github.io/drat/",
↪  "https://cran.rstudio.com/"))
```

Discussion and free practical time

Any questions?