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

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 handle missing data in one or more test results

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

```
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[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

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
```

```
##
## JAGS model summary statistics from 20000 samples (chains = 2;
↪  adapt+burnin = 5000):
##
##                Lower95  Median Upper95    Mean         SD
## prev[1]        0.21422 0.30021 0.39187 0.30141  0.045606
## prev[2]        0.65574 0.75334 0.84256 0.75111  0.048021
## se[1]          0.75699 0.83684 0.91043 0.83492  0.039756
## se[2]          0.77069  0.8477 0.92018 0.84555  0.038603
```

	Lower95	Median	Upper95	SSeff	psrf
prev[1]	0.214	0.300	0.392	13448	1.000
prev[2]	0.656	0.753	0.843	11045	1.000
se[1]	0.757	0.837	0.910	7202	1.000
se[2]	0.771	0.848	0.920	6576	1.000
se[3]	0.874	0.933	0.986	5564	1.001
sp[1]	0.883	0.939	0.985	6686	1.000
sp[2]	0.923	0.971	1.000	3757	1.001
sp[3]	0.915	0.969	1.000	3057	1.000
truestatus[1]	0.000	0.000	0.000	20000	1.002
truestatus[2]	1.000	1.000	1.000	20000	1.106
truestatus[3]	1.000	1.000	1.000	10000	1.291
truestatus[4]	1.000	1.000	1.000	10000	1.291
truestatus[5]	1.000	1.000	1.000	20000	1.010

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

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

Modify the simulation code to introduce an antibody response step between the true status and the test results (see below in the HTML file for example R code).

Simulate data from three antibody tests including the antibody response step

Does the sensitivity / specificity estimated by the model recover the true prevalence parameter?

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

Branching of processes leading to test results

- Sometimes we have multiple tests that are detecting a similar thing
    - For example: two antibody tests and one antigen test
    - The antibody tests will be correlated

Branching of processes leading to test results

- Sometimes we have multiple tests that are detecting a similar thing
    - For example: two antibody tests and one antigen test
    - The antibody tests will be correlated
- Or even three antibody tests where two are primed to detect the same thing, and one has a different target!
    - In this case all three tests are correlated, but two are more strongly correlated

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])

# Tests 1 & 2 will be co-dependent on antibody response:
antibody <- rbinom(N*Populations, 1, abse*true +
↪  (1-absp)*(1-true))
# Simulate test 1 & 2 results based on this other latent state:
test1 <- rbinom(N*Populations, 1, se1*antibody +
↪  (1-sp1)*(1-antibody))
test2 <- rbinom(N*Populations, 1, se2*antibody +
↪  (1-sp2)*(1-antibody))

# Simulate test results for the independent test 3:
test3 <- rbinom(N*Populations, 1, se3*true + (1-sp3)*(1-true))

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

```r
# The overall sensitivity of the correlated tests is:
abse*se1 + (1-abse)*(1-sp1)
```

```
## [1] 0.65
```

```r
abse*se2 + (1-abse)*(1-sp2)
```

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

```r
# The overall specificity of the correlated tests is:
absp*sp1 + (1-absp)*(1-se1)
```

```
## [1] 0.8
```

```r
absp*sp2 + (1-absp)*(1-se2)
```

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

```r
# The overall sensitivity of the correlated tests is:
abse*se1 + (1-abse)*(1-sp1)
```

```
## [1] 0.65
```

```r
abse*se2 + (1-abse)*(1-sp2)
```

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

```r
# The overall specificity of the correlated tests is:
absp*sp1 + (1-absp)*(1-se1)
```

```
## [1] 0.8
```

```r
absp*sp2 + (1-absp)*(1-se2)
```

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

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

First use autohuiwalter to create a model file:

```r
auto_huiwalter(ind3tests, 'auto3tihw.bug')
```

Then find the lines for the covariances that we want to activate:

```
# 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
```

And edit so it looks like:

```
# 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
```

[i.e. swap the comments around]

You will also need to uncomment out the relevant initial values for BOTH chains (on lines 117-122 and 128-133):

```
# "covse12" <- 0
# "covse13" <- 0
# "covse23" <- 0
# "covsp12" <- 0
# "covsp13" <- 0
# "covsp23" <- 0
```

So that they look like:

```
"covse12" <- 0
# "covse13" <- 0
# "covse23" <- 0
"covsp12" <- 0
# "covsp13" <- 0
# "covsp23" <- 0
```

```
results <- run.jags('auto3tihw.bug')
```

Exercise

Simulate data with N=1000 and dependence between tests 1 and 2

Then fit a model assuming independence between all tests and compare the results to your simulation parameters

Now turn on covariance between tests 1 and 2 and refit the model. Are the results more reasonable?

Optional Exercise

Re-fit a model to this data using all three possible covse and covsp parameters

What do you notice about the results?

Session 3c: Model selection

Motivation

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

Motivation

- 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

Motivation

- 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)

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)

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

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?!?

Free practical time

- Explore the optional exercises (and solutions) and feel free to ask questions!
- Feedback very welcome!