

Diagnostic test evaluation with imperfect reference test

Paolo Eusebi

16/09/2021

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
- So how do we extend these models to multiple diagnostic tests?

Hui-Walter models for 2 tests and 1 population

Hui-Walter Model

- A particular model formulation that was originally designed for evaluating diagnostic tests in the absence of a gold standard
- Not necessarily (or originally) Bayesian but often implemented using Bayesian MCMC
- But evaluating an imperfect test against another imperfect test is a bit like pulling a rabbit out of a hat
- If we don't know the true disease status, how can we estimate sensitivity or specificity for either test?

Model Specification

```
model{  
  y ~ dmulti(prob, TotalTests)  
  
  # Test1- Test2-  
  prob[1] <- (p*((1-se[1])*(1-se[2]))) + ((1-p)*((sp[1])*(1-sp[2])))  
  # Test1+ Test2-  
  prob[2] <- (p*((se[1])*(1-se[2]))) + ((1-p)*((1-sp[1])*(1-sp[2])))  
  # Test1- Test2+  
  prob[3] <- (p*((1-se[1])*(se[2]))) + ((1-p)*((sp[1])*(se[2])))  
  # Test1+ Test2+  
  prob[4] <- (p*((se[1])*(se[2]))) + ((1-p)*((1-sp[1])*(se[2])))
```

Model Specification

```
p ~ dbeta(1, 1)
se[1] ~ dbeta(1, 1)
sp[1] ~ dbeta(1, 1)
se[2] ~ dbeta(1, 1)
sp[2] ~ dbeta(1, 1)
#data# y, TotalTests
#monitor# p, prob, se, sp
#inits# p, se, sp
}
```

Data, initial values, run

	T2+	T2-
T1+	48	4
T1-	12	36

```
y <- c(48, 12, 4, 36)
TotalTests <- sum(y)
p <- 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(hw_2t_1p, n.chains=2)
```

[Remember to check convergence and effective sample size!]

results

	Lower95	Median	Upper95	SSeff	psrf
p	0.324	0.498	0.665	4538	2.289
prob[1]	0.369	0.462	0.561	13451	1.000
prob[2]	0.074	0.133	0.202	13747	1.000
prob[3]	0.018	0.055	0.104	9522	1.000
prob[4]	0.252	0.343	0.435	13154	1.000
se[1]	0.026	0.535	1.000	4673	15.458
se[2]	0.000	0.392	0.966	4765	13.472
sp[1]	0.000	0.475	0.972	4796	15.167
sp[2]	0.036	0.627	1.000	4389	13.506

- Note the wide credible intervals!

- These models need A LOT of data
 - And/or strong priors for one of the tests
- Convergence is more problematic than usual
- Be **very** careful with the order of combinations in `dmultinom!`
- Check your results carefully to ensure they make sense!

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 $se+sp \geq 1$:

```
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.

```
se1 <- 0.9; sp1 <- 0.95;
se2 <- 0.8; sp2 <- 0.99
prevalence <- 0.5; N <- 100
truestatus <- rbinom(N, 1, prevalence)
Test1 <- rbinom(N, 1, (truestatus * se1) + ((1-truestatus)
Test2 <- rbinom(N, 1, (truestatus * se2) + ((1-truestatus)
t <- table(Test1, Test2)
y <- as.numeric(t)
```

Can we recover these parameter values?

Exercise

Modify the code in the Hui Walter model to force tests to be no worse than useless

Simulate data and recover parameters for:

- $N=10$
- $N=100$
- $N=1000$

Solution

Model definition:

```
model{
  y ~ dmulti(prob, TotalTests)

  # Test1- Test2-
  prob[1] <- (p * ((1-se[1])*(1-se[2]))) + ((1-p) * ((sp[1]
  # Test1+ Test2-
  prob[2] <- (p * ((se[1])*(1-se[2]))) + ((1-p) * ((1-sp[1]
  # Test1- Test2+
  prob[3] <- (p * ((1-se[1])*(se[2]))) + ((1-p) * ((sp[1]
  # Test1+ Test2+
  prob[4] <- (p * ((se[1])*(se[2]))) + ((1-p) * ((1-sp[1]

  p ~ dbeta(1, 1)
  se[1] ~ dbeta(HPSo[1,1], HPSo[1,2])T(1-sp[1], )
  se[2] ~ dbeta(HPSo[2,1], HPSo[2,2])T(1-sp[2], )
  sp[1] ~ dbeta(HPSo[1,3], HPSo[1,4])T(1-se[1], )
  sp[2] ~ dbeta(HPSo[2,3], HPSo[2,4])T(1-se[2], )
}
```

Optional Exercise

Compare results with the following priors for test 1:

- Sensitivity = 0.9 (95% CI: 0.85 - 0.95)
- Specificity = 0.95 (95%CI: 0.92-0.97)

[These are the same as in session 1]

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(hw_2t_1p_priors, n.chains=2)
```

Hui-Walter models for 2 tests and N populations

Hui-Walter models with multiple populations

- Basically an extension of the single-population model
- Works best with multiple populations each with differing prevalence
- Including an unexposed population works well BUT be wary of assumptions regarding constant sensitivity/specificity across populations with very different types of infections

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-p  
    ## 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  
  #inits# prev, se, sp
```

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

```
se1 <- 0.9; se2 <- 0.8; sp1 <- 0.95; sp2 <- 0.99
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$truestatus * se1))
simdata$Test2 <- rbinom(N, 1, (simdata$truestatus * se2) + (1 - simdata$truestatus * se2))
```

```
head(simdata)
```

##	Population	probability	truestatus	Test1	Test2
## 1	3	0.9	1	1	1
## 2	3	0.9	0	0	0
## 3	2	0.5	0	0	0
## 4	3	0.9	1	1	1
## 5	1	0.1	0	0	0
## 6	2	0.5	1	1	1

[Except that probability and truestatus would not normally be known!]

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

There is a function (provided in the GitHub repo) that can do this for us:

```
simdata$Population <- factor(simdata$Population, levels=source("autohuiwalter.R"))
auto_huiwalter(simdata[,c('Population','Test1','Test2')], c
```

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

```
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){
```

And can be run directly from R:

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

	Lower95	Median	Upper95	SSeff	psrf
se[1]	0.721	0.848	0.954	7729	1
se[2]	0.792	0.901	0.999	6345	1
sp[1]	0.772	0.892	0.994	6706	1
sp[2]	0.819	0.932	1.000	5957	1
prev[1]	0.002	0.101	0.240	7111	1
prev[2]	0.269	0.478	0.692	8654	1
prev[3]	0.697	0.838	0.960	7248	1
covse12	0.000	0.000	0.000	NA	NA
covsp12	0.000	0.000	0.000	NA	NA

- Modifying priors must still be done directly in the model file
- The model needs to be re-generated if the data changes
- But remember that your modified priors will be reset
- There must be a single column for the population (as a factor), and all of the other columns (either factor, logical or numeric) are interpreted as being test results
- The function will soon be included in the runjags package