Session 4

Multi-test, multi-population models

Matt Denwood

2021-06-29

Why stop at two tests?

In *traditional* diagnostic test evaluation, one test is assumed to be a gold standard from which all other tests are evaluated

- So it makes no difference if you assess one test at a time or do multiple tests at the same time

Why stop at two tests?

In *traditional* diagnostic test evaluation, one test is assumed to be a gold standard from which all other tests are evaluated

- So it makes no difference if you assess one test at a time or do multiple tests at the same time

Using a latent class model each new test adds new information - so we should analyse all available test results in the same model

Simulating data: simple example

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

```r
# Parameter values to simulate:
N <- 200
sensitivity <- c(0.8, 0.9, 0.95)
specificity <- c(0.95, 0.99, 0.95)

Populations <- 2
prevalence <- c(0.25,0.5)

data <- tibble(Population = sample(seq_len(Populations), N,
↪  replace=TRUE)) %>%
  mutate(Status = rbinom(N, 1, prevalence[Population])) %>%
  mutate(Test1 = rbinom(N, 1, sensitivity[1]*Status +
  ↪  (1-specificity[1])*(1-Status))) %>%
  mutate(Test2 = rbinom(N, 1, sensitivity[2]*Status +
  ↪  (1-specificity[2])*(1-Status))) %>%
  mutate(Test3 = rbinom(N, 1, sensitivity[3]*Status +
  ↪  (1-specificity[3])*(1-Status))) %>%
  select(-Status)
```

Model specification

Like for two tests, except it is now a 2x2x2 table

Model specification

Like for two tests, except it is now a 2x2x2 table

```
Tally[1:8,p] ~ dmulti(prob[1:8,p], TotalTests[p])

# Probability of observing Test1- Test2- Test3-
prob[1,p] <-  prev[p] * ((1-se[1])*(1-se[2])*(1-se[3]) +
              (1-prev[p]) * (sp[1]*sp[2]*sp[3])

# Probability of observing Test1+ Test2- Test3-
prob[2,p] <-  prev[p] * (se[1]*(1-se[2])*(1-se[3])) +
              (1-prev[p]) * ((1-sp[1])*sp[2]*sp[3])

## snip ##

# Probability of observing Test1+ Test2+ Test3+
prob[3,p] <-  prev[p] * (se[1]*se[2]*se[3]) +
              (1-prev[p]) * ((1-sp[1])*(1-sp[2])*(1-sp[3]))
```

Model specification

Like for two tests, except it is now a 2x2x2 table

```
Tally[1:8,p] ~ dmulti(prob[1:8,p], TotalTests[p])

# Probability of observing Test1- Test2- Test3-
prob[1,p] <-  prev[p] * ((1-se[1])*(1-se[2])*(1-se[3]) +
              (1-prev[p]) * (sp[1]*sp[2]*sp[3])

# Probability of observing Test1+ Test2- Test3-
prob[2,p] <-  prev[p] * (se[1]*(1-se[2])*(1-se[3])) +
              (1-prev[p]) * ((1-sp[1])*sp[2]*sp[3])

## snip ##

# Probability of observing Test1+ Test2+ Test3+
prob[3,p] <-  prev[p] * (se[1]*se[2]*se[3]) +
              (1-prev[p]) * ((1-sp[1])*(1-sp[2])*(1-sp[3]))
```

- We need to take **extreme** care with these equations, and the multinomial tabulation!!!

Are the tests conditionally independent?

- Example: we have one blood, one milk, and one faecal test
    - But the blood and milk test are basically the same test
    - Therefore they are more likely to give the same result

Are the tests conditionally independent?

- Example: we have one blood, one milk, and one faecal test
    - But the blood and milk test are basically the same test
    - Therefore they are more likely to give the same result
- Example: we test people for COVID using an antigen test on a nasal swab, a PCR test on a throat swab, and the same antigen test on the same throat swab
    - The virus may be present in the throat, nose, neither, or both
    - But we use the same antigen test twice
        - Might it cross-react with the same non-target virus?

Are the tests conditionally independent?

- Example: we have one blood, one milk, and one faecal test
  - But the blood and milk test are basically the same test
  - Therefore they are more likely to give the same result
- Example: we test people for COVID using an antigen test on a nasal swab, a PCR test on a throat swab, and the same antigen test on the same throat swab
  - The virus may be present in the throat, nose, neither, or both
  - But we use the same antigen test twice
    - Might it cross-react with the same non-target virus?
- In both situations we have pairwise correlation between some of the tests

Dealing with correlation: Covid example

It helps to consider the data simulation as a (simplified) biological process (where my parameters are not representative of real life!):

```
# The probability of infection with COVID in two populations:
prevalence <- c(0.01,0.05)
# The probability of shedding COVID in the nose conditional on
↪  infection:
nose_shedding <- 0.8
# The probability of shedding COVID in the throat conditional on
↪  infection:
throat_shedding <- 0.8
# The probability of detecting virus with the antigen test:
antigen_detection <- 0.75
# The probability of detecting virus with the PCR test:
pcr_detection <- 0.999
# The probability of random cross-reaction with the antigen test:
antigen_crossreact <- 0.05
# The probability of random cross-reaction with the PCR test:
pcr_crossreact <- 0.01
```

Dealing with correlation: Covid example

It helps to consider the data simulation as a (simplified) biological process (where my parameters are not representative of real life!):

```
# The probability of infection with COVID in two populations:
prevalence <- c(0.01,0.05)
# The probability of shedding COVID in the nose conditional on
↪  infection:
nose_shedding <- 0.8
# The probability of shedding COVID in the throat conditional on
↪  infection:
throat_shedding <- 0.8
# The probability of detecting virus with the antigen test:
antigen_detection <- 0.75
# The probability of detecting virus with the PCR test:
pcr_detection <- 0.999
# The probability of random cross-reaction with the antigen test:
antigen_crossreact <- 0.05
# The probability of random cross-reaction with the PCR test:
pcr_crossreact <- 0.01
```

Note: cross-reactions are assumed to be independent!

Simulating latent states:

```
N <- 20000
Populations <- length(prevalence)

covid_data <- tibble(Population = sample(seq_len(Populations), N,
↪  replace=TRUE)) %>%
  ## True infection status:
  mutate(Status = rbinom(N, 1, prevalence[Population])) %>%
  ## Nose shedding status:
  mutate(Nose = Status * rbinom(N, 1, nose_shedding)) %>%
  ## Throat shedding status:
  mutate(Throat = Status * rbinom(N, 1, throat_shedding))
```

Simulating test results:

```r
covid_data <- covid_data %>%
  ## The nose swab antigen test may be false or true positive:
  mutate(NoseAG = case_when(
    Nose == 1 ~ rbinom(N, 1, antigen_detection),
    Nose == 0 ~ rbinom(N, 1, antigen_crossreact)
  )) %>%
  ## The throat swab antigen test may be false or true positive:
  mutate(ThroatAG = case_when(
    Throat == 1 ~ rbinom(N, 1, antigen_detection),
    Throat == 0 ~ rbinom(N, 1, antigen_crossreact)
  )) %>%
  ## The PCR test may be false or true positive:
  mutate(ThroatPCR = case_when(
    Throat == 1 ~ rbinom(N, 1, pcr_detection),
    Throat == 0 ~ rbinom(N, 1, pcr_crossreact)
  ))
```

The overall sensitivity of the tests can be calculated as follows:

```
covid_sensitivity <- c(
  # Nose antigen:
  nose_shedding*antigen_detection +
↪  (1-nose_shedding)*antigen_crossreact,
  # Throat antigen:
  throat_shedding*antigen_detection +
↪  (1-throat_shedding)*antigen_crossreact,
  # Throat PCR:
  throat_shedding*pcr_detection + (1-throat_shedding)*pcr_crossreact
)
covid_sensitivity
## [1] 0.6100 0.6100 0.8012
```

The overall specificity of the tests is more straightforward:

```r
covid_specificity <- c(
  # Nose antigen:
  1 - antigen_crossreact,
  # Throat antigen:
  1 - antigen_crossreact,
  # Throat PCR:
  1 - pcr_crossreact
)
covid_specificity
## [1] 0.95 0.95 0.99
```

The overall specificity of the tests is more straightforward:

```r
covid_specificity <- c(
  # Nose antigen:
  1 - antigen_crossreact,
  # Throat antigen:
  1 - antigen_crossreact,
  # Throat PCR:
  1 - pcr_crossreact
)
covid_specificity
## [1] 0.95 0.95 0.99
```

However: this assumes that cross-reactions are independent!

Model specification

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

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

## snip ##

# 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] )
```

Model specification

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

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

## snip ##

# 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] )
```

It is quite easy to get the terms slightly wrong!

Template Hui-Walter

The model code and data format for an arbitrary number of populations (and tests) can be determined automatically using the template_huiwalter function from the runjags package:

```
template_huiwalter(
  covid_data %>% select(Population, NoseAG, ThroatAG, ThroatPCR),
  outfile = 'covidmodel.txt')
```

This generates self-contained model/data/initial values etc

```
model{

    ## Observation layer:

    # Complete observations (N=20000):
    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]
    }


    ## Observation probabilities:

    for(p in 1:Populations){

        # Probability of observing NoseAG- ThroatAG- ThroatPCR- 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 NoseAG- ThroatAG- ThroatPCR- from a
        ↪  true negative::
        sp_prob[1,p] <- (1-prev[p]) * (sp[1]*sp[2]*sp[3] +covsp12
        ↪  +covsp13 +covsp23)

        # Probability of observing NoseAG+ ThroatAG- ThroatPCR- from a
        ↪  true positive::
```

```
## Inits:
inits{
"se" <- c(0.5, 0.99, 0.5)
"sp" <- c(0.99, 0.75, 0.99)
"prev" <- c(0.05, 0.95)
# "covse12" <- 0
# "covse13" <- 0
# "covse23" <- 0
# "covsp12" <- 0
# "covsp13" <- 0
# "covsp23" <- 0
}
inits{
"se" <- c(0.99, 0.5, 0.99)
"sp" <- c(0.75, 0.99, 0.75)
"prev" <- c(0.95, 0.05)
# "covse12" <- 0
# "covse13" <- 0
# "covse23" <- 0
# "covsp12" <- 0
# "covsp13" <- 0
# "covsp23" <- 0
}

## Data:
data{
"Populations" <- 2
```

And can be run directly from R:

```r
results <- run.jags('covidmodel.txt')
## Loading required namespace: rjags
results
```

	Lower95	Median	Upper95	SSeff	psrf
se[1]	0.584	0.632	0.684	9111	1.000
se[2]	0.717	0.768	0.816	7191	1.000
se[3]	0.946	0.982	1.000	5839	1.001
sp[1]	0.944	0.947	0.951	11350	1.000
sp[2]	0.948	0.951	0.954	11986	1.001
sp[3]	0.987	0.989	0.991	7351	1.000
prev[1]	0.005	0.007	0.009	9324	1.000
prev[2]	0.037	0.042	0.046	7374	1.000
covse12	0.000	0.000	0.000	NA	NA
covsp12	0.000	0.000	0.000	NA	NA
covse13	0.000	0.000	0.000	NA	NA
covsp13	0.000	0.000	0.000	NA	NA
covse23	0.000	0.000	0.000	NA	NA
covsp23	0.000	0.000	0.000	NA	NA

Template Hui-Walter

- Modifying priors must still be done directly in the model file
  - Same for adding .RNG.seed and the deviance monitor
- 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

Template Hui-Walter

- Modifying priors must still be done directly in the model file
    - Same for adding .RNG.seed and the deviance monitor
- 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
- Covariance terms are all deactivated by default

Activating covariance terms

Find the lines for the covariances that we want to activate (i.e. the two Throat tests):

```
# Covariance in sensitivity between ThroatAG and ThroatPCR 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 ThroatAG and ThroatPCR 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 assumed to be
 ↪  independent
```

And edit so it looks like:

```
# Covariance in sensitivity between ThroatAG and ThroatPCR 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 ThroatAG and ThroatPCR 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 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('covidmodel.txt', sample=50000)
results
##
## JAGS model summary statistics from 100000 samples (chains = 2;
↪  adapt+burnin = 5000):
##
##               Lower95      Median     Upper95        Mean
## se[1]         0.58864     0.64874      0.7104      0.6494
## se[2]         0.57344     0.69754      0.7894     0.68985
## se[3]         0.77955      0.9236     0.99998     0.90911
## sp[1]         0.94453     0.94839     0.95256     0.94848
## sp[2]         0.94607     0.94945     0.95278     0.94943
## sp[3]         0.98577      0.9882     0.99025     0.98816
## prev[1]     0.0050779   0.0071614   0.0096176   0.0072409
## prev[2]      0.037501    0.043536    0.051425     0.04397
## covse12             0           0           0           0
## covsp12             0           0           0           0
## covse13             0           0           0           0
## covsp13             0           0           0           0
## covse23    -0.0063817     0.02604    0.076519    0.029692
## covsp23   -0.00017968 0.00036898 0.00090546 0.00036243
##
##                   SD  Mode       MCerr MC%ofSD SSeff     AC.10
## se[1]       0.031125    --  0.00028779     0.9 11697  0.055513
## se[2]       0.057466    --   0.0013049     2.3  1940   0.59787
## se[3]       0.067675    --   0.0016625     2.5  1657    0.7113
## sp[1]      0.0020385    -- 0.000032975     1.6  3822   0.27681
```

Adding “Bayesian p-values”

What on earth is that?

- “Bayesian p-values” are effectively the probability that a particular parameter has a positive (or negative) value, i.e. the probability that it is non-zero
- The interpretaiton is actually much cleaner than frequentist p-values, so it is probably a bad name for them...

Calculation in JAGS code:

```
bpv_cse23 <- covse23 <= 0
bpv_csp23 <- covsp23 <= 0

#monitor# bpv_cse23, bpv_csp23
```

The mean estimate for covse23_bpv and covsp23_bpv is the “Bayesian p-value”

```
results <- run.jags('covidmodel.txt', sample=50000)
results
##
## JAGS model summary statistics from 100000 samples (chains = 2;
↪  adapt+burnin = 5000):
##
##                 Lower95     Median    Upper95       Mean
## bpv_cse23             0          0          1    0.08095
## bpv_csp23             0          0          1    0.09775
## se[1]           0.58734    0.64879    0.70943    0.64951
## se[2]           0.57268      0.697    0.78811    0.69001
## se[3]           0.78031    0.92384          1    0.90939
## sp[1]           0.94457    0.94842    0.95256     0.9485
## sp[2]           0.94611    0.94949    0.95276    0.94945
## sp[3]           0.98592     0.9882    0.99032    0.98817
## prev[1]       0.0049427  0.0071529  0.0094816  0.0072274
## prev[2]        0.037448   0.043538   0.051544   0.043986
## covse12               0          0          0          0
## covsp12               0          0          0          0
## covse13               0          0          0          0
## covsp13               0          0          0          0
## covse23      -0.0061604   0.025714     0.0764   0.029492
## covsp23     -0.00021357 0.00037614 0.00088459 0.00036759
```

Alternative in R:

```
cov23 <- combine.mcmc(results, vars=c("covse23", "covsp23"))
str(cov23)
##  'mcmc' num [1:100000, 1:2] 0.0448 0.054 0.0389 0.064 0.0531 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:100000] "5001" "5002" "5003" "5004" ...
##   ..$ : chr [1:2] "covse23" "covsp23"
##  - attr(*, "mcpar")= num [1:3] 5001 105000 1
apply(cov23 <= 0, 2, mean)
## covse23 covsp23
## 0.08144 0.09772
```

Alternative in R:

```
cov23 <- combine.mcmc(results, vars=c("covse23", "covsp23"))
str(cov23)
##  'mcmc' num [1:100000, 1:2] 0.0448 0.054 0.0389 0.064 0.0531 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:100000] "5001" "5002" "5003" "5004" ...
##   ..$ : chr [1:2] "covse23" "covsp23"
##  - attr(*, "mcpar")= num [1:3] 5001 105000 1
apply(cov23 <= 0, 2, mean)
## covse23 covsp23
## 0.08144 0.09772
```

[Note: these numbers are not quite the same as the JAGS version, because the thinning used for calculating summary statistics is slightly different between the two]

Or even:

```
results <- add.summary(results, mutate = function(x){
  list(
    bpvR_covse23 = x[,"covse23"] <= 0,
    bpvR_covsp23 = x[,"covsp23"] <= 0
  )
})
summary(results, vars="bpv")
##              Lower95 Median Upper95     Mean        SD Mode
## bpv_cse23          0      0       1 0.080875 0.2726465    0
## bpv_csp23          0      0       1 0.099475 0.2993025    0
## bpvR_covse23       0      0       1 0.080875 0.2726465    0
## bpvR_covsp23       0      0       1 0.099475 0.2993025    0
##                    MCerr MC%ofSD SSeff       AC.10
## bpv_cse23    0.002731974     1.0  9960 0.090067306
## bpv_csp23    0.002062952     0.7 21050 0.003656881
## bpvR_covse23 0.002731974     1.0  9960 0.090067306
## bpvR_covsp23 0.002062952     0.7 21050 0.003656881
##                   psrf
## bpv_cse23    1.0002539
## bpv_csp23    0.9999852
## bpvR_covse23 1.0002539
## bpvR_covsp23 0.9999852
```

Or even:

```
results <- add.summary(results, mutate = function(x){
  list(
    bpvR_covse23 = x[,"covse23"] <= 0,
    bpvR_covsp23 = x[,"covsp23"] <= 0
  )
})
summary(results, vars="bpv")
##              Lower95 Median Upper95     Mean        SD Mode
## bpv_cse23          0      0       1 0.080875 0.2726465    0
## bpv_csp23          0      0       1 0.099475 0.2993025    0
## bpvR_covse23       0      0       1 0.080875 0.2726465    0
## bpvR_covsp23       0      0       1 0.099475 0.2993025    0
##                    MCerr MC%ofSD SSeff       AC.10
## bpv_cse23    0.002731974     1.0  9960 0.090067306
## bpv_csp23    0.002062952     0.7 21050 0.003656881
## bpvR_covse23 0.002731974     1.0  9960 0.090067306
## bpvR_covsp23 0.002062952     0.7 21050 0.003656881
##                   psrf
## bpv_cse23    1.0002539
## bpv_csp23    0.9999852
## bpvR_covse23 1.0002539
## bpvR_covsp23 0.9999852
```

See `?runjags::add.summary` for more information on mutate

Practical considerations

- Correlation terms add complexity to the model in terms of:
    - Opportunity to make a coding mistake
    - Reduced identifiability

Practical considerations

- Correlation terms add complexity to the model in terms of:
  - Opportunity to make a coding mistake
  - Reduced identifiability
- The template_huiwalter function helps us with coding mistakes
- Only careful consideration of covariance terms can help us with identifiability

Practical considerations

- Correlation terms add complexity to the model in terms of:
    - Opportunity to make a coding mistake
    - Reduced identifiability
- The template_huiwalter function helps us with coding mistakes
- Only careful consideration of covariance terms can help us with identifiability
- We will return to these themes tomorrow!

Practical session 4

Points to consider

1. How does including a third test impact the inference for the first two tests?
2. What happens if we include correlation between tests?
3. Can we include correlation if we only have 2 tests?

Summary

- Including multiple tests is technically easy
  - But philosophically more difficult!!!
- Complexity of adding correlation terms increases non-linearly with more tests
  - Probably best to stick to correlations with biological justification?
- Adding/removing test results may change the posterior for
  - Other test Se / Sp
  - Prevalence