Hui-Walter models - Adjusting for conditional dependencies between tests

CA18208 HARMONY Zurich Training School - https://harmony-net.eu/

Sonja Hartnack Valerie Hungerbühler Eleftherios Meletis

2022-07-14

Recap Hui-Walter

- Model assumptions
  - The population is divided into two or more populations with different prevalences in which two or more tests are evaluated
  - *Se* and *Sp* are the same in all populations
  - The tests are *conditionally independent* given the disease status.

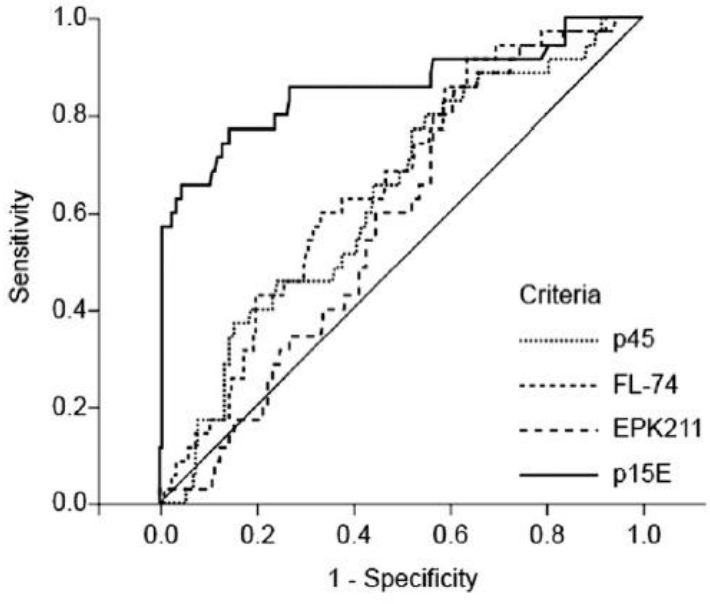
Criticism of the Hui-Walter paradigm assumption (1)

- The assumption (1) of distinct prevalences is necessary for the Hui-Walter model because otherwise, the data can be collapsed into a single 2x2 table with only three degrees of freedom for estimation.
- The smaller the difference between disease prevalences, the larger are the posterior credible intervals, indicating a loss in precision.
- The smallest difference in prevalence assessed by simulation was 10%. In case of rare diseases, it might be difficult to find populations with prevalences higher than 10%.

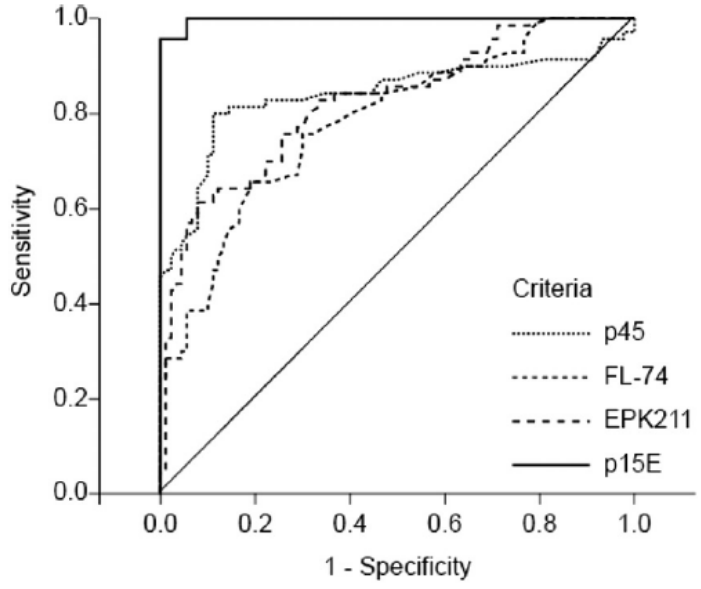
Criticism of the Hui-Walter paradigm assumption (2)

- In reality, the assumption of constant *Se* and *Sp* is questionable as already illustrated in the figure. Here the same diagnostic tests have been applied in the same laboratory to experimentally and naturally infected cats.

DOI: https://doi.org/10.1128/JCM.02584-13



Naturally infected cats



Experimentally infected cats

Criticism of the Hui-Walter paradigm assumption (2)

- If assumption (2) is not satisfied, the accuracies would differ between two populations, this would add four additional parameters to be estimated, while there are only three additional degrees of freedom.
- *Se* and *Sp* are assumed to vary with external factors.
- *Se*, for example, especially when detecting an infectious agent, may depend on the prevalence and the stage of disease.
- The occurrence of a so-called *spectrum bias* contradicts this assumption.

Criticism of the Hui-Walter paradigm assumption (3)

- Assumption (3) demanding conditional independence was the first to be questioned by Vacek (1985).
- Not accounting for potential conditional dependence may lead to misleading, biased estimates with a positive correlation leading to an over-estimation and a negative correlation to an under-estimation of the test Se and Sp.

Condtional independencies

- Test are considered conditionally independent if the probability of getting a given test result on one test does not depend on the result from the other test, given the disease status of the individual.
- Conditional independence implies that given that an animal is diseased (or not) the probability P of positive (or negative) outcomes for T_1, the test results of the first test, is the same - regardless of the known outcome for the second test, T_2.

Conditional independence

$$P(T_1{}^+, T_2{}^+|D^+) = P(T_1{}^+|D^+) \times P(T_2{}^+|D^+)]$$

Conditional dependence

$$P(T_1{}^+, T_2{}^+|D^+) \neq P(T_1{}^+|D^+) \times P(T_2{}^+|D^+)$$

Conditional dependencies

- Conditional dependence, in contrast, implies that

$$P({T_1}^+|{T_2}^+) \neq P({T_1}^+|{T_2}^-)$$

and / or

$$P({T_1}^-|{T_2}^-) \neq P({T_1}^-|{T_2}^+)$$

Conditional (in)dependencies Interpretation

- Seen from a biological perspective, conditional dependency between two diagnostic tests could occur if both tests are based on the same biological principle.
- For example, the *Sp*s of two ELISAs might be conditionally dependent because they are both affected by the same cross-reacting agent. Another example would be two PCRs utilising fecal material which might contain substances potentially inhibiting the PCR reaction.

Conditional dependencies

- Obviously, conditional dependencies or covariances are additional parameters to be estimated, which in the frequentist situation (without any constraints put on the parameters) would lead to a non-identifiable problem (over-parameterisation).
- Whereas under the assumption of conditional independence at least three tests per sample allowing to estimate seven parameters are needed, under the assumption of conditional dependence 15 parameters need to estimated thus leading to non-identifiability.
- It is of course vital, that the parameters of a latent class model are identifiable to obtain meaningful estimates.

Conditional dependencies

TABLE 2. Maximum Number of Estimable Parameters and Number of Parameters to Be Estimated in the Absence of Conditional Independence and Under Conditional Independence as a Function of the Number of Tests per Subject

Number of Tests	Maximum Number of Estimable Parameters	Parameters to be Estimated Under Conditional Dependence	Parameters to Be Estimated Under Conditional Independence
1	1	3	3
2	3	7	5
3	7	15	7
4	15	31	9
5	31	63	11
h	$2^h - 1$	$2^{h+1} - 1$	$2h + 1$

Berkvens D et al. (2006) Estimating Disease Prevalence in a Bayesian Framework Using Probabilistic Constraints. doi: 10.1097/01.ede.0000198422.64801.8d

Conditional dependencies

- For example for two diagnostic tests named T_1 and T_2 the probabilities of the four different options of binary test results $(++, +-, -+, --)$ including also two conditional dependencies
  - *covs*12, the covariance between the sensitivities of test 1 and 2
  - *covc*12, the covariance between specificities of test 1 and 2)

  could be modelled as follows:

$$P(T_1^+, T_2^+) = [Pr \cdot (Se_1 \cdot Se_2 + covs12) + (1 - Pr) \cdot ((1 - Sp_1) \cdot (1 - Sp_2) + covc12)]^{a_i}$$
$$P(T_1^+, T_2^-) = [Pr \cdot (Se_1 \cdot (1 - Se_2) - covs12) + (1 - Pr) \cdot ((1 - Sp_1) \cdot Sp_2 - covc12)]^{b_i}$$
$$P(T_1^-, T_2^+) = [Pr \cdot ((1 - Se_1) \cdot Se_2 - covs12) + (1 - Pr) \cdot (Sp_1 \cdot (1 - Sp_2) - covc12)]^{c_i}$$
$$P(T_1^-, T_2^-) = [Pr \cdot ((1 - Se_1) \cdot (1 - Se_2) + covs12) + (1 - Pr) \cdot (Sp_1 \cdot Sp_2 + covc12)]^{d_i}.$$

Example of a COVID-19 data set

Dealing with correlation

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

2 pops - 3 tests

```
# 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
```

2 pops - 3 tests

```
# 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:

```
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:

```
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:

```
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 runjas package:

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

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 pos
        se_prob[1,p] <- prev[p] * ((1-se[1])*(1-se[2])*(1-se[3]) +covse12 +covs
        # Probability of observing NoseAG- ThroatAG- ThroatPCR- from a true neg
        sp_prob[1,p] <- (1-prev[p]) * (sp[1]*sp[2]*sp[3] +covsp12 +covsp13 +cov

        # Probability of observing NoseAG+ ThroatAG- ThroatPCR- from a true pos
        se_prob[2,p] <- prev[p] * (se[1]*(1-se[2])*(1-se[3]) -covse12 -covse13
        # Probability of observing NoseAG+ ThroatAG- ThroatPCR- from a true neg
        sp_prob[2,p] <- (1-prev[p]) * ((1-sp[1])*sp[2]*sp[3] -covsp12 -covsp13

        # Probability of observing NoseAG- ThroatAG+ ThroatPCR- from a true pos
        se_prob[3,p] <- prev[p] * ((1-se[1])*se[2]*(1-se[3]) -covse12 +covse13
```

```
    covse23 ~ dunif( (se[2]-1)*(1-se[3]) , min(se[2],se[3]) - se[2]*se[3] )  ##
    # covse23 <- 0  ## if the sensitivity of these tests can be assumed to be i
    # Calculated relative to the min/max for ease of interpretation:
    corse23 <- ifelse(covse23 < 0, -covse23 / ((se[2]-1)*(1-se[3])), covse23 /

    # 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] )  ##
    # covsp23 <- 0  ## if the specificity of these tests can be assumed to be i
    # Calculated relative to the min/max for ease of interpretation:
    corsp23 <- ifelse(covsp23 < 0, -covsp23 / ((sp[2]-1)*(1-sp[3])), covsp23 /

}

#monitor# se, sp, prev, covse12, corse12, covsp12, corsp12, covse13, corse13, c

## 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{
```

And can be run directly from R:

```
results <- run.jags('covidmodel.txt')
results
```

	Lower95	Median	Upper95	SSeff	psrf
se[1]	0.462	0.576	0.685	666	1.002
se[2]	0.518	0.691	0.804	331	1.019
se[3]	0.728	0.933	1.000	193	1.011
sp[1]	0.940	0.944	0.949	1197	1.008
sp[2]	0.946	0.950	0.954	2343	1.001
sp[3]	0.987	0.990	0.993	1926	1.000
prev[1]	0.005	0.008	0.011	437	1.007
prev[2]	0.039	0.046	0.061	180	1.040
covse12	-0.041	-0.001	0.030	1080	1.001
corse12	-0.400	-0.011	0.203	1055	1.001
covsp12	-0.001	0.000	0.001	3158	1.000
corsp12	-0.160	0.001	0.022	3927	1.000
covse13	-0.028	0.001	0.042	620	1.002
corse13	-0.990	0.038	0.792	1467	1.001
covsp13	0.000	0.000	0.001	3632	1.000
corsp13	-0.275	0.041	0.137	4819	1.001
covse23	-0.013	0.013	0.068	336	1.026
corse23	-0.698	0.324	1.000	1583	1.005
covsp23	-0.001	0.000	0.000	4067	1.000
corsp23	-0.917	-0.228	0.062	4685	1.000

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):

You will also need to uncomment out the relevant initial values for BOTH chains:

Session Summary

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

Session Summary

- 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

Exercise 1

Use the template_huiwalter function to look at the simple 2-test 5-population example from the previous session. Use this data simulation code:

```
# Set a random seed so that the data are reproducible:
set.seed(2022-07-14)

sensitivity <- c(0.9, 0.6)
specificity <- c(0.95, 0.9)
N <- 1000

# Change the number of populations here:
Populations <- 5
# Change the variation in prevalence here:
(prevalence <- runif(Populations, min=0.1, max=0.9))
## [1] 0.7069277 0.5860048 0.2746717 0.2833154 0.2223198
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-Sta
  mutate(Test2 = rbinom(N, 1, sensitivity[2]*Status + (1-specificity[2])*(1-Sta
  select(-Status)

(twoXtwoXpop <- with(data, table(Test1, Test2, Population)))
## , , Population = 1
##
```

Solution 1

There is no particular solution to the first part of this exercise, but please ask if you have any questions about the model code that template_huiwalter generates. Remember that re-running the template_huiwalter function will over-write your existing model including any changes you made, so be careful!

We can run the model as follows:

```
results_nocov <- run.jags("template_2test.txt")
results_nocov
##
## JAGS model summary statistics from 20000 samples (chains = 2; adapt+burnin =
##
##           Lower95  Median Upper95    Mean       SD Mode      MCerr MC%ofSD SSe
## se[1]     0.76877 0.84622 0.92131 0.84578 0.038946   -- 0.00071843     1.8  29
## se[2]     0.55967  0.6317 0.70599 0.63173 0.037366   -- 0.00069277     1.9  29
## sp[1]     0.85486 0.91121 0.97116 0.91159 0.029329   -- 0.00060579     2.1  23
## sp[2]     0.87605 0.91509 0.95522 0.91508 0.020129   -- 0.00033918     1.7  35
## prev[1]   0.64617 0.74773 0.84331  0.7466 0.050319   -- 0.00082298     1.6  37
## prev[2]   0.53081 0.63291 0.73245 0.63248 0.051631   -- 0.00083422     1.6  38
## prev[3]   0.22984 0.31816 0.41352 0.31937 0.047452   -- 0.00074205     1.6  40
## prev[4]   0.16655 0.25978  0.3616 0.26183 0.050104   -- 0.00094806     1.9  27
## prev[5] 0.061849 0.13201 0.21955 0.13543 0.041295   --   0.000766     1.9  29
## covse12        0       0       0       0        0    0         --      --
```