inlami testing

Steffi Muff

2023-11-24

Installing and loading

Installation and loading works without any problems.

```
# devtools::install_github('emmaSkarstein/inlami')
library(inlami)
```

Warm-up: Simulation 1 with classical measurement error

I start by generating data for a simulation. Regression model

$$y = \beta_0 + \beta_x x + \beta_z z + \varepsilon \ ,$$

with $x \sim \mathcal{N}(0,1)$, $z \sim \text{Bern}(0.5)$, $w = x + u$ and $u \sim \mathcal{N}(0, 0.5)$, thus $\sigma_u^2 = 0.5$ and $\tau_u = 2$ (precision, which is how the parametrization seems to work).

```
set.seed(34312)
nn <- 1000
x <- rnorm(nn)
# Error-prone variable
w <- x + rnorm(nn, 0, sd = sqrt(0.5))
z <- rbinom(nn, 1, 0.5)

y <- 1 + x + z + rnorm(nn, 0, 1)

data1 <- data.frame(cbind(x = x, z = z, y = y, w = w))
```

```
summary(lm(y ~ x + z))$coef
```

```
##              Estimate Std. Error  t value      Pr(>|t|)
## (Intercept) 0.9744259 0.04673182 20.85145  2.018740e-80
## x           0.9878008 0.03261162 30.28984 2.039621e-143
## z           1.0451892 0.06376319 16.39173  1.193883e-53
```

```
summary(lm(y ~ w + z))$coef
```

```
##              Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 1.0002965 0.05393332 18.54691 3.348934e-66
## w           0.6381789 0.03042749 20.97376 3.397491e-81
## z           1.0443126 0.07360222 14.18860 9.318290e-42
```

Note that, below, I'm using the imputation model `w~z`, even though x does not depend on z here, but in a practical case we wouldn't know this in advance.

```
main_formula <- y ~ w + z
imputation_formula <- w ~ z
```

```r
model1 <- fit_inlami(data = data1,
                     formula_moi = main_formula,
                     formula_imp = imputation_formula,
                     family_moi = "gaussian",
                     error_type = c("classical"),
                     #prior.prec.moi = c(10, 9),
                     # prior.prec.berkson = c(10, 9),
                     prior.prec.classical = c(19, 9),
                     prior.prec.imp = c(10, 9),
                     prior.beta.error = c(0, 1/1000),
                     #initial.prec.moi = 1,
                     #initial.prec.berkson = 1,
                     initial.prec.classical = 2,
                     initial.prec.imp = 1)

summary(model1)
```

```
## Formula for model of interest:
## y ~ w + z
##
## Formula for imputation model:
## w ~ z
##
## Error types:
## [1] "classical"
##
## Fixed effects for model of interest:
##            mean         sd 0.025quant 0.5quant 0.975quant     mode          kld
## beta.0 1.000824 0.05782989  0.8873623 1.000818   1.114316 1.000818 7.299831e-10
## beta.z 1.035708 0.07895699  0.8805242 1.035795   1.190391 1.035795 1.030040e-09
##
## Coefficient for error prone variable:
##             mean        sd 0.025quant  0.5quant 0.975quant      mode
## beta.w 0.9831324 0.1181658  0.7588381 0.9803971   1.223839 0.9683031
##
## Fixed effects for imputation model:
##                 mean         sd 0.025quant     0.5quant 0.975quant         mode
## alpha.0 -0.001458907 0.05609410 -0.1114750 -0.001458907  0.1085572 -0.001458907
## alpha.z  0.023815708 0.07654728 -0.1263148  0.023815708  0.1739462  0.023815708
##                  kld
## alpha.0 5.706899e-12
## alpha.z 5.707035e-12
##
## Model hyperparameters (apart from beta.w):
##                                             mean        sd 0.025quant
## Precision for the Gaussian observations    0.984084 0.1128272  0.7872329
## Precision for the Gaussian observations[2] 2.011823 0.4391813  1.2557397
## Precision for the Gaussian observations[3] 1.064897 0.1363440  0.8300335
##                                             0.5quant 0.975quant      mode
## Precision for the Gaussian observations    0.9755351   1.230790 0.9543371
## Precision for the Gaussian observations[2] 1.9752580   2.974821 1.9175329
## Precision for the Gaussian observations[3] 1.0535745   1.365831 1.0260185
```

This seems to work very well.

Some thoughts

- if `family_moi` is not Gaussian, do we then still need `prior.prec.moi`? Or is it then forbidden to use it, since the family then does not have a residual term?
- I would like to have the option to use fixed priors, for example for the error variance σ_u^2. Is this possible?
- One thing I confused myself with was that I used `x` as the variable in the models, but of course we usually don't know `x`, but only the observed version `w`. After I fixed that, the modeling worked out well.
- In the output, we have all these lines for `Precision for the Gaussian observations`. I know this is how INLA does it, but maybe it is possible to rename to somewhat more informative naming?

Simulation 2: Classical measurement error with random effects main model

```r
set.seed(34312)
nn <- 1000

z <- rbinom(nn, 1, 0.5)

x <- rnorm(nn, 1 + 0.5 * z, 1)
# Error-prone variable
w <- x + rnorm(nn, 0, sd = sqrt(0.5))

# Random effect to include in the regression model:
re <- rep(rnorm(nn/50), each = 20)

y <- 1 + x + z + re + rnorm(nn, 0, 1)

data2 <- data.frame(cbind(x = x, z = z, y = y, w = w, id = rep(seq(1:50),
    each = 20)))
```

Regression models with correct and error-prone variables. The attenuation in β_w is similar to before. Note that we now need random effects models (I use the `lme4` package for this):

```r
library(lme4)
summary(lmer(y ~ x + z + (1 | id), data = data2))
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: y ~ x + z + (1 | id)
##    Data: data2
##
## REML criterion at convergence: 3012.6
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.4055 -0.6664 -0.0273  0.6783  2.9553
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  id       (Intercept) 1.581    1.2573
##  Residual             0.993    0.9965
## Number of obs: 1000, groups:  id, 50
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  0.87902    0.18708   4.699
## x            1.00184    0.03308  30.285
```

```
## z            0.95959    0.06637  14.458
##
## Correlation of Fixed Effects:
##   (Intr) x
## x -0.192
## z -0.130 -0.223
```

```
summary(lmer(y ~ w + z + (1 | id), data = data2))
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: y ~ w + z + (1 | id)
##    Data: data2
##
## REML criterion at convergence: 3314.4
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.1655 -0.6904 -0.0191  0.6853  3.1214
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  id       (Intercept) 1.518    1.232
##  Residual             1.367    1.169
## Number of obs: 1000, groups:  id, 50
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  1.29352    0.18533   6.979
## w            0.64918    0.03214  20.199
## z            1.06550    0.07775  13.704
##
## Correlation of Fixed Effects:
##   (Intr) w
## w -0.179
## z -0.166 -0.219
```

Error modeling with INLA:

```
main_formula <- y ~ w + z + f(id, model="iid")
imputation_formula <- w ~ z

model2 <- fit_inlami(data = data2,
                         formula_moi = main_formula,
                         formula_imp = imputation_formula,
                         family_moi = "gaussian",
                         error_type = c("classical"),
                         #prior.prec.moi = c(10, 9),
                         # prior.prec.berkson = c(10, 9),
                         prior.prec.classical = c(19, 9),
                         prior.prec.imp = c(10, 9),
                         prior.beta.error = c(0, 1/1000),
                         #initial.prec.moi = 1,
                         #initial.prec.berkson = 1,
                         initial.prec.classical = 2,
                         initial.prec.imp = 1)
```

```r
summary(model2)
```

```
## Formula for model of interest:
## y ~ w + z + f(id, model = "iid")
##
## Formula for imputation model:
## w ~ z
##
## Error types:
## [1] "classical"
##
## Fixed effects for model of interest:
##             mean        sd 0.025quant  0.5quant 0.975quant      mode
## beta.0 0.8894947 0.2162545  0.4634684 0.8907982   1.309606 0.8912172
## beta.z 0.8625021 0.1000829  0.6613956 0.8647653   1.052380 0.8652276
##                 kld
## beta.0 6.246932e-09
## beta.z 1.286348e-07
##
## Coefficient for error prone variable:
##            mean        sd 0.025quant 0.5quant 0.975quant     mode
## beta.w 1.023308 0.1311869  0.7705212  1.02147      1.287 1.013591
##
## Fixed effects for imputation model:
##              mean         sd 0.025quant  0.5quant 0.975quant      mode
## alpha.0 1.0380795 0.05341060  0.9333274 1.0380795  1.1428316 1.0380795
## alpha.z 0.5219312 0.07457073  0.3756786 0.5219312  0.6681839 0.5219312
##                  kld
## alpha.0 4.203199e-12
## alpha.z 4.205176e-12
##
## Model hyperparameters (apart from beta.w):
##                                              mean        sd 0.025quant
## Precision for the Gaussian observations    0.9815304 0.1209167  0.7694706
## Precision for the Gaussian observations[2] 2.0287315 0.4392232  1.2832618
## Precision for the Gaussian observations[3] 1.1423000 0.1534152  0.8758379
## Precision for id                           0.6941152 0.1453111  0.4494074
##                                             0.5quant 0.975quant      mode
## Precision for the Gaussian observations    0.9726947   1.244843 0.9524370
## Precision for the Gaussian observations[2] 1.9886700   3.003546 1.9189432
## Precision for the Gaussian observations[3] 1.1302309   1.478795 1.1029376
## Precision for id                           0.6802213   1.018842 0.6544018
```

Some thoughts

- This works like a breeze.
- However, (new) INLA users must be aware that priors for any random effects components added with `f()` must be specified within the `f()` function. This is obvious to us, but maybe not to anybody?

Simulation 3: Classical measurement error and missing data with random effects main model

This is a relatively simple extension of simulation 2.

```r
set.seed(34312)
nn <- 1000

z <- rbinom(nn, 1, 0.5)

x <- rnorm(nn, 1 + 0.5 * z, 1)
# Error-prone variable
w <- x + rnorm(nn, 0, sd = sqrt(0.5))

# Generating missing data, depending on z
eta <- -2 + z
prob_missing <- exp(eta)/(1 + exp(eta))
missing_index <- rbinom(nn, 1, prob_missing)
# Proportion missing:
sum(missing_index)/nn
```

```
## [1] 0.175
```

```r
# Replace the values in w by missing in case the index for
# missingness is =1:
w <- ifelse(missing_index == 1, NA, w)

# Random effect to include in the regression model:
re <- rep(rnorm(nn/50), each = 20)

y <- 1 + x + z + re + rnorm(nn, 0, 1)

data3 <- data.frame(cbind(x = x, z = z, y = y, w = w, id = rep(seq(1:50),
    each = 20)))
```

```r
library(lme4)
summary(lmer(y ~ x + z + (1 | id), data = data3))$coef
```

```
##              Estimate Std. Error   t value
## (Intercept) 0.9922761 0.14571957  6.809491
## x           1.0109902 0.03254673 31.062727
## z           1.0013306 0.06529682 15.335060
```

```r
summary(lmer(y ~ w + z + (1 | id), data = data3))$coef
```

```
##              Estimate Std. Error   t value
## (Intercept) 1.4051469 0.14658123  9.586131
## w           0.6753527 0.03497651 19.308753
## z           1.1313730 0.08448100 13.392040
```

Modeling error and missing data with INLA:

```r
main_formula <- y ~ w + z + f(id,model="iid")
imputation_formula <- w ~ z

model3 <- fit_inlami(data = data3,
                         formula_moi = main_formula,
                         formula_imp = imputation_formula,
                         family_moi = "gaussian",
                         error_type = c("classical","missing"),
                         #prior.prec.moi = c(10, 9),
```

```
                           # prior.prec.berkson = c(10, 9),
                           prior.prec.classical = c(19, 9),
                           prior.prec.imp = c(10, 9),
                           prior.beta.error = c(0, 1/1000),
                           #initial.prec.moi = 1,
                           #initial.prec.berkson = 1,
                           initial.prec.classical = 2,
                           initial.prec.imp = 1)

summary(model3)
```

```
## Formula for model of interest:
## y ~ w + z + f(id, model = "iid")
##
## Formula for imputation model:
## w ~ z
##
## Error types:
## [1] "classical" "missing"
##
## Fixed effects for model of interest:
##             mean        sd 0.025quant  0.5quant 0.975quant      mode
## beta.0 1.0086368 0.1813751  0.6508086 1.0108022   1.357838 1.0118876
## beta.z 0.9019328 0.1031256  0.6935073 0.9045234   1.096981 0.9049064
##                 kld
## beta.0 4.303867e-08
## beta.z 1.448328e-07
##
## Coefficient for error prone variable:
##          mean        sd 0.025quant 0.5quant 0.975quant     mode
## beta.w 1.0515 0.1296505  0.8013482 1.049789   1.311789 1.042475
##
## Fixed effects for imputation model:
##              mean         sd 0.025quant  0.5quant 0.975quant      mode
## alpha.0 1.0148018 0.05539221  0.9061588 1.0148022  1.1234424 1.0148022
## alpha.z 0.5182331 0.07962652  0.3620229 0.5182466  0.6743668 0.5182466
##                  kld
## alpha.0 1.322519e-11
## alpha.z 1.814516e-11
##
## Model hyperparameters (apart from beta.w):
##                                             mean        sd 0.025quant
## Precision for the Gaussian observations    1.046245 0.1389741  0.8039485
## Precision for the Gaussian observations[2] 2.054485 0.4288540  1.3235878
## Precision for the Gaussian observations[3] 1.145428 0.1508188  0.8821868
## Precision for id                           1.230888 0.2662232  0.7833206
##                                            0.5quant 0.975quant     mode
## Precision for the Gaussian observations    1.035611   1.350181 1.011835
## Precision for the Gaussian observations[2] 2.016343   3.003939 1.949572
## Precision for the Gaussian observations[3] 1.133986   1.474994 1.108382
## Precision for id                           1.205200   1.826202 1.158262
```

Some thoughts

- When there is missingess in w, do I then have to explicitly specify `error_type = c("classical","missing")`?

What happens if I only specify `error_type = c("classical")`? I think it would be nice if in such a case, I would get a warning, saying that the error-prone variable also contains missing data, and then the user can decide to model this as well. But actually, I think it does happen automatically in INLA... So maybe the specification of `missing` is not really needed? It is anyway nice to have it - because if there is only missingness, then one otherwise has an empty argument.

Simulation 4: Logistic regression with classical error and missing data

```
set.seed(34312)
nn <- 1000

z <- rbinom(nn, 1, 0.5)

x <- rnorm(nn, 1 - 0.5 * z, 1)
# Error
w <- x + rnorm(nn, 0, 1)

# Generating missing data, depending on z
eta <- -1 + z
prob_missing <- exp(eta)/(1 + exp(eta))
missing_index <- rbinom(nn, 1, prob_missing)

# Proportion missing (same as in Simulation 3):
sum(missing_index)/nn
```

```
## [1] 0.354
```

```
# Replace the values in w by missing in case the index for
# missingness is =1:
w <- ifelse(missing_index == 1, NA, w)

# Random effect to include in the regression model:
re <- rep(rnorm(nn/50), each = 20)

# Linear predictor
eta <- x + z + re

# Generate Poisson response
y <- rpois(nn, exp(eta))

data4 <- data.frame(cbind(x = x, z = z, y = y, w = w, id = rep(seq(1:50),
    each = 20)))
```

Simple regression models with correct and error-prone variables to check the effect:

```
library(lme4)
summary(glmer(y ~ x + z + (1 | id), data = data4, family = "poisson"))$coef
```

```
##                Estimate Std. Error    z value  Pr(>|z|)
## (Intercept) -0.05309503 0.14521315 -0.3656351 0.7146374
## x            1.01740937 0.01204064 84.4979470 0.0000000
## z            1.02901327 0.02370157 43.4154010 0.0000000
```

```
summary(glmer(y ~ w + z + (1 | id), data = data4, family = "poisson"))$coef
```

```
##              Estimate Std. Error   z value      Pr(>|z|)
```

```
## (Intercept) 0.8087528 0.15146255  5.339622  9.314041e-08
## w           0.4725777 0.01126571 41.948318  0.000000e+00
## z           0.6466170 0.02801695 23.079490 7.440950e-118
```

Modeling error and missing data with INLA:

```
main_formula <- y ~ w + z + f(id,model="iid")
imputation_formula <- w ~ z

model4 <- fit_inlami(data = data4,
                         formula_moi = main_formula,
                         formula_imp = imputation_formula,
                         family_moi = "poisson",
                         error_type = c("classical","missing"),
                         # prior.prec.moi = c(10, 9),
                         # prior.prec.berkson = c(10, 9),
                         prior.prec.classical = c(19, 9),
                         prior.prec.imp = c(10, 9),
                         prior.beta.error = c(0, 1/1000),
                         #initial.prec.moi = 1,
                         #initial.prec.berkson = 1,
                         initial.prec.classical = 2,
                         initial.prec.imp = 1)

summary(model4)
```

What sort of family argument(s) would Poisson models have? Here I think one would probably need to do something similar to the Weibull regression case and spell out the `control.family` list (right?), but it was not immediately clear to me how to do this (namely because I don't know what the family argument for Poisson should be).

Framinham with repeated observations

Model with repeated measurements for the error, using the Framingham data.

```
framingham_model <- fit_inlami(formula_moi = disease ~ sbp + smoking,
    formula_imp = sbp ~ smoking, family_moi = "binomial", data = framingham,
    error_type = "classical", repeated_observations = TRUE, prior.prec.classical = c(100,
        1), prior.prec.imp = c(10, 1), prior.beta.error = c(0, 0.01),
    initial.prec.classical = 100, initial.prec.imp = 10, control.fixed = list(prec = list(beta.0 = 0.01
        beta.smoking = 0.01, alpha.0 = 0.01, alpha.smoking = 0.01)))
summary(framingham_model)
```

```
## Formula for model of interest:
## disease ~ sbp + smoking
##
## Formula for imputation model:
## sbp ~ smoking
##
## Error types:
## [1] "classical"
##
## Fixed effects for model of interest:
##                   mean        sd 0.025quant   0.5quant 0.975quant       mode
## beta.0       -2.3607128 0.2687680 -2.8893582 -2.3601373 -1.8353098 -2.3601248
## beta.smoking  0.3989499 0.2976652 -0.1843512  0.3988026  0.9830828  0.3988002
```

```
##                        kld
## beta.0       1.193032e-09
## beta.smoking 8.968709e-11
##
## Coefficient for error prone variable:
##              mean        sd 0.025quant 0.5quant 0.975quant     mode
## beta.sbp 1.904277 0.5619081  0.8468291  1.88849   3.057062 1.817648
##
## Fixed effects for imputation model:
##                      mean         sd  0.025quant    0.5quant 0.975quant
## alpha.0        0.01454288 0.01858122 -0.02190449  0.01454288 0.05099026
## alpha.smoking -0.01958464 0.02156253 -0.06187990 -0.01958464 0.02271062
##                      mode          kld
## alpha.0        0.01454288 7.072895e-11
## alpha.smoking -0.01958464 7.072890e-11
##
## Model hyperparameters (apart from beta.sbp):
##                                              mean       sd 0.025quant
## Precision for the Gaussian observations[2] 75.90183 3.690487   68.89263
## Precision for the Gaussian observations[3] 19.89455 1.234047   17.55065
##                                            0.5quant 0.975quant     mode
## Precision for the Gaussian observations[2] 75.81338   83.41982 75.63951
## Precision for the Gaussian observations[3] 19.86501   22.40732 19.82452
```

Some thoughts

- I can formulate and fit the model as below. However, I do not understand how the model knows that `sbp` corresponds to `sbp1` and `sbp2`. Is it automatically stacking it in the correct way? What happens if the number of repeats is different for different individuals?
- I actually think that what we really have is an error model that allows for a random, individual-specific effect

  $$w_{ij} = x_i + u_{ij} \ ,$$

  where $x_i \sim \mathcal{N}(0, \sigma_x^2)$. The user would need to specify the individual i that belongs to the observation in the respective w_{ij}. So, in principle the user would need an argument where he/she can specify this information (it would corresponds to information that used to be stored in id.x internally for the error model part).

Simulation 5: Model where exposure model contains random effects.

I'm just trying a case where the exposure model contains a random effect. I build on Simulation 1 and 2, in order to not add unneccesary complications. But note that the random effect is no longer present in the model of interest.

```r
set.seed(34312)
nn <- 1000

z <- rbinom(nn, 1, 0.5)

# Random effect to include in the regression model:
re <- rep(rnorm(nn/50), each = 20)

x <- rnorm(nn, 1 + 0.5 * z + re, 1)
# Error-prone variable
w <- x + rnorm(nn, 0, sd = sqrt(0.5))
```

```r
y <- 1 + x + z + rnorm(nn, 0, 1)

data5 <- data.frame(cbind(x = x, z = z, y = y, w = w, id = rep(seq(1:50),
    each = 20)))
```

```r
summary(lm(y ~ x + z, data = data5))$coef
```

```
##              Estimate Std. Error  t value      Pr(>|t|)
## (Intercept) 1.0121255 0.05527611 18.31036  8.734038e-65
## x           1.0027485 0.02280029 43.97963 1.045454e-235
## z           0.9587177 0.06366670 15.05839  2.514818e-46
```

```r
summary(lm(y ~ w + z, data = data5))$coef
```

```
##              Estimate Std. Error  t value      Pr(>|t|)
## (Intercept) 1.3021134 0.06289534 20.70286  1.749265e-79
## w           0.8143140 0.02446434 33.28576 5.699205e-164
## z           0.9904699 0.07515845 13.17842  1.119857e-36
```

So now I add the random effect to the exposure model, and it works. **This is really cool! Definitely a point to sell in the package paper!**

```r
main_formula <- y ~ w + z
imputation_formula <- w ~ z + f(id,model="iid")

model5 <- fit_inlami(data = data5,
                          formula_moi = main_formula,
                          formula_imp = imputation_formula,
                          family_moi = "gaussian",
                          error_type = c("classical"),
                          #prior.prec.moi = c(10, 9),
                          # prior.prec.berkson = c(10, 9),
                          prior.prec.classical = c(19, 9),
                          prior.prec.imp = c(10, 9),
                          prior.beta.error = c(0, 1/1000),
                          #initial.prec.moi = 1,
                          #initial.prec.berkson = 1,
                          initial.prec.classical = 2,
                          initial.prec.imp = 1)

summary(model5)
```

```
## Formula for model of interest:
## y ~ w + z
##
## Formula for imputation model:
## w ~ z + f(id, model = "iid")
##
## Error types:
## [1] "classical"
##
## Fixed effects for model of interest:
##             mean         sd 0.025quant 0.5quant 0.975quant      mode
## beta.0 1.0362363 0.07582700  0.8865573 1.036522   1.184398 1.0365134
## beta.z 0.9108545 0.07857957  0.7563091 0.911003   1.064555 0.9110038
##                 kld
```

```
## beta.0 4.153990e-08
## beta.z 9.167879e-10
##
## Coefficient for error prone variable:
##             mean         sd 0.025quant 0.5quant 0.975quant     mode
## beta.w 1.007393 0.03817923  0.9317813 1.007547   1.082107 1.008187
##
## Fixed effects for imputation model:
##               mean         sd 0.025quant  0.5quant 0.975quant      mode
## alpha.0 1.3084954 0.14655680  1.0199052 1.3085729  1.5966445 1.3085721
## alpha.z 0.5103392 0.07645265  0.3603727 0.5103461  0.6602665 0.5103462
##                  kld
## alpha.0 2.190857e-08
## alpha.z 1.074848e-11
##
## Model hyperparameters (apart from beta.w):
##                                              mean         sd 0.025quant
## Precision for the Gaussian observations    0.9821533 0.06804262  0.8539726
## Precision for the Gaussian observations[2] 2.2723641 0.26886024  1.7998347
## Precision for the Gaussian observations[3] 1.0596679 0.07572369  0.9163398
## Precision for id                           1.1133750 0.23692486  0.7057867
##                                             0.5quant 0.975quant      mode
## Precision for the Gaussian observations    0.9801646   1.121736 0.9769617
## Precision for the Gaussian observations[2] 2.2530625   2.856939 2.2079992
## Precision for the Gaussian observations[3] 1.0576829   1.214306 1.0552418
## Precision for id                           1.0935543   1.633610 1.0614106
```