Covariate misclassification from a latent Gaussian variable with measurement error

Code and details for 'Bayesian models for missing and misclassified variables using integrated nested Laplace approximations'

Emma Skarstein, Leonardo Bastos, Håvard Rue and Stefanie Muff

```
library(INLA)
library(dplyr)
library(ggplot2)
```

Models adjusting for misclassification arising from a dichotomized variable with error

In this example, we simulate a continuous latent variable with measurement error, which is then dichotomized.

```
set.seed(1)
n <- 200

x_c <- rnorm(n, 0, 1) # Continuous, not observed
x_d <- ifelse(x_c > 0, 1, 0) # Dichotomized, not observed
w_c <- x_c + rnorm(n) # Continuous with cont. ME, may be observed
w_d <- ifelse(w_c > 0, 1, 0) # Dichotomized, observed
```

From the variables we have simulated, we can generate a sample misclassification matrix:

```
pix0 <- sum(x_d==0)
pix1 <- sum(x_d==1)
table(unobserved = x_d, observed = w_d)/matrix(c(pix0, pix0, pix1, pix1),
                                               byrow = TRUE, nrow = 2)
```

```
##           observed
## unobserved         0         1
##          0 0.6603774 0.3396226
##          1 0.2872340 0.7127660
```

Lastly, we generate the response.

```
y <- 1 + x_c + rnorm(n)
```

Now, the observed data is `y` and `w_d` (often, researchers choose to use a dichotomized variable even though the continuous version is available, so perhaps `w_c` is also observed, but simply ignored, or it may have been

available at some point but was not sent to the statistician). For the purposes of this example, we assume that we are interested in the relationship between `y` and `x_c`.

As described in Section 3.2.2, we then have the model

$$\begin{aligned}
\boldsymbol{y} &= \beta_0 \mathbf{1} + \beta_{x_c} \boldsymbol{x}_c + \boldsymbol{Z}^\top \boldsymbol{\beta} + \boldsymbol{\varepsilon} \ , & \boldsymbol{\varepsilon} &\sim \mathcal{N}(\mathbf{0}, \sigma_\varepsilon^2 \boldsymbol{I}) \ , \\
\boldsymbol{w}_c &= \boldsymbol{x}_c + \boldsymbol{u}_{w_c} \ , & \boldsymbol{u}_{w_c} &\sim \mathcal{N}(\mathbf{0}, \sigma_{w_c}^2 \boldsymbol{I}) \ , \\
\boldsymbol{w}_d &= \mathrm{I}(\boldsymbol{w}_c > 0) \ , & & \\
\boldsymbol{x}_c &= \alpha_0 \mathbf{1} + \widetilde{\boldsymbol{Z}}^\top \boldsymbol{\alpha} + \boldsymbol{\varepsilon}_{x_c} \ , & \boldsymbol{\varepsilon}_{x_c} &\sim \mathcal{N}(\mathbf{0}, \sigma_{x_c}^2 \boldsymbol{I}) \ ,
\end{aligned}$$

where $\boldsymbol{Z}$ and $\widetilde{\boldsymbol{Z}}$ are both covariate matrices with whichever covariates are relevant to include, these are assumed to be without error. Now, we have that

$$\begin{aligned}
\Pr(w_{d,i} = 1 \mid w_{c,i}) &= \Pr(w_{c,i} > 0) \\
&= \Pr(x_{c,i} + u_{w_c,i} > 0) \\
&= \Pr\left(\frac{u_{w_c,i}}{\sigma_{w_c}} < \frac{x_{c,i}}{\sigma_{w_c}} \right) \\
&= \Phi\left(\frac{x_{c,i}}{\sigma_{w_c}} \right) \ ,
\end{aligned}$$

since $u_{w_c,i}/\sigma_{w_c} \sim \mathcal{N}(0,1)$. This relation means that we can model $\boldsymbol{w}_d$ as a Bernoulli variable with a probit link function. We can re-write the hierarchical model from above:

$$\begin{aligned}
\boldsymbol{y} &= \beta_0 \mathbf{1} + \beta_{x_c} \boldsymbol{x}_c + \boldsymbol{Z}^\top \boldsymbol{\beta} + \boldsymbol{\varepsilon} \ , & \boldsymbol{\varepsilon} &\sim \mathcal{N}(\mathbf{0}, \sigma_\varepsilon^2 \boldsymbol{I}) \ , \\
\boldsymbol{w}_d &\sim \text{Bernoulli}(\Phi(\boldsymbol{x_c}/\sigma_{w_c})) \ , & & \\
\boldsymbol{x}_c &= \alpha_0 \mathbf{1} + \widetilde{\boldsymbol{Z}}^\top \boldsymbol{\alpha} + \boldsymbol{\varepsilon}_{x_c} \ , & \boldsymbol{\varepsilon}_{x_c} &\sim \mathcal{N}(\mathbf{0}, \sigma_{x_c}^2 \boldsymbol{I}) \ ,
\end{aligned}$$

We use stacks in R-INLA to structure the hierarchical model:

```
stk_y <- inla.stack(data = list(y = y),
                    A = list(1),
                    effects = list(
                      list(beta.0 = rep(1, n),
                           beta.x_c = 1:n)),
                    tag = "y")

stk_w_d <- inla.stack(data = list(w_d = w_d),
                      A = list(1),
                      effects = list(
                        list(id.x_c = 1:n,
                             weight.x_c = 1)),
                      tag = "w_d")

stk_x_c <- inla.stack(data = list(x_c = rep(0, n)),
                      A = list(1),
                      effects = list(
                        list(id.x_c = 1:n,
                             weight.x_c = -1,
                             alpha.0 = rep(1, n))),
                      tag = "x_c")

stk_full <- inla.stack(stk_y, stk_w_d, stk_x_c)
```

```
formula <- list(y, w_d, x_c) ~ -1 + beta.0 + alpha.0 +
  f(beta.x_c, copy = "id.x_c",
    hyper = list(beta = list(param = c(0, 1/1000), fixed = FALSE))) +
  f(id.x_c, weight.x_c, model = "iid", values = 1:n,
    hyper = list(prec = list(initial = -15, fixed = TRUE)))

res <- inla(formula, data = inla.stack.data(stk_full), Ntrials = rep(1, n),
            family = c("gaussian", "binomial", "gaussian"),
            control.family = list(
              list(hyper = list(prec = list(initial = log(1),
                                            param = c(10, 9),
                                            fixed = FALSE))),
              list(link = "probit"),
              list(hyper = list(prec = list(initial = log(1),
                                            param = c(10, 9),
                                            fixed = FALSE)))
            ),
            control.predictor = list(compute = TRUE))
```

Looking at the results

```
data <- data.frame(y = y, x_c = x_c, w_c = w_c, w_d = w_d, x_d = x_d)
```

Results from using correct continuous variable:

```
correct_model <- inla(y ~ x_c, data = data)
correct_model$summary.fixed
```

```
##                   mean         sd 0.025quant  0.5quant 0.975quant      mode
## (Intercept) 0.9553833 0.07565846  0.8068701 0.9553833   1.103896 0.9553833
## x_c         1.0785024 0.08157634  0.9183726 1.0785024   1.238632 1.0785024
##                      kld
## (Intercept) 1.943431e-09
## x_c         1.943381e-09
```

Results from using dichotomized version but continuous latent variable (our proposed model):

```
res$summary.fixed["beta.0",]
```

```
##             mean        sd 0.025quant  0.5quant 0.975quant      mode
## beta.0 0.9373776 0.1387185  0.6597809 0.9387132    1.20735 0.9386891
##                 kld
## beta.0 7.097634e-08
```

```
res$summary.hyperpar["Beta for beta.x_c",]
```

```
##                       mean        sd 0.025quant 0.5quant 0.975quant     mode
## Beta for beta.x_c 1.157997 0.2971564   0.584487 1.154131   1.754411 1.137612
```

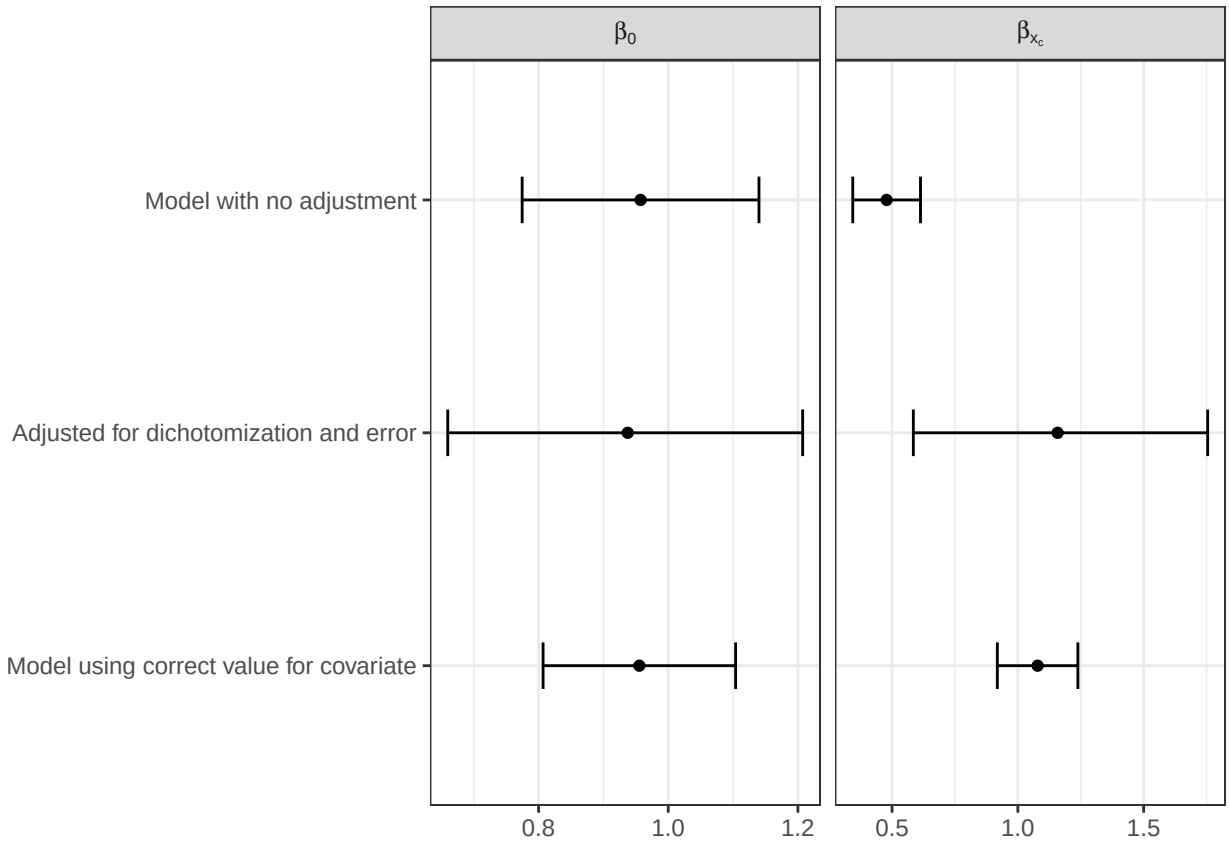
Results from using continuous variable with measurement error with no correction:

```
naive_model <- inla(y ~ w_c, data = data)
naive_model$summary.fixed
```

```
##                   mean         sd 0.025quant  0.5quant 0.975quant      mode
## (Intercept) 0.9572666 0.09306920  0.7745771 0.9572666  1.1399560 0.9572666
## w_c         0.4784398 0.06858374  0.3438138 0.4784398  0.6130657 0.4784398
```

```
##                       kld
## (Intercept) 1.942834e-09
## w_c         1.942790e-09
```

```
adjusted_results <- rbind("(Intercept)" = res$summary.fixed["beta.0",1:5],
                          "x_c" = res$summary.hyperpar["Beta for beta.x_c",1:5])
adjusted_results$variable <- rownames(adjusted_results)

naive_results <- naive_model$summary.fixed
rownames(naive_results) <- c("(Intercept)", "x_c")
naive_results$variable <- rownames(naive_results)

correct_results <- correct_model$summary.fixed
correct_results$variable <- rownames(correct_results)

all_res <- dplyr::bind_rows(adjusted = adjusted_results,
                            naive = naive_results,
                            correct = correct_results, .id = "Model")

all_res$labels <- paste0("beta", "[", c(0, "x[c]"), "]")
all_res$Model <- factor(all_res$Model, levels = c("naive", "adjusted", "correct"))
all_res$Model <- plyr::revalue(all_res$Model,
                               c("naive" = "Model with no adjustment",
                                 "adjusted" = "Adjusted for dichotomization and error",
                                 "correct" = "Model using correct value for covariate"))

ggplot(all_res, aes(y = Model)) +
  geom_point(aes(x = mean)) +
  geom_errorbarh(aes(xmin = .data$"0.025quant", xmax = .data$"0.975quant"), height = .2) +
  scale_y_discrete(limits = rev) +
  facet_wrap(vars(labels), scales = "free_x", labeller = label_parsed) +
  theme_bw() +
  theme(axis.title = element_blank())
```




```
ggsave("figures/cont_to_binary.pdf", height = 2.3, width = 7)
```

How do different measurement error variances correspond to different misclassification probabilities?

Using simulated data as well as the exact sensitivity and specificity calculated given our simulation setup, we can take a look at how the sensitivity and specificity of the classification of the continuous variable change for increasing measurement error variance.

First, we simulate data for different levels of measurement error:

```
simulate_dichotomized <- function(me_variance = 1, n = 200){
  x_c <- rnorm(n, 0, 1) # Continuous, not observed
  x_d <- ifelse(x_c > 0, 1, 0) # Dichotomized, not observed
  w_c <- x_c + rnorm(n, 0, sqrt(me_variance)) # Continuous with cont. ME, may be observed
  w_d <- ifelse(w_c > 0, 1, 0) # Dichotomized, observed

  return(list(x_d = x_d, w_d = w_d))
}

calculate_mc <- function(x_d, w_d){
  pix0 <- sum(x_d==0)
  pix1 <- sum(x_d==1)
  mc_tab <- table(unobserved = x_d, observed = w_d)/
    matrix(c(pix0, pix0, pix1, pix1), byrow = TRUE, nrow = 2)
  return(list(spec = mc_tab[1,1], sens = mc_tab[2,2]))
}
```

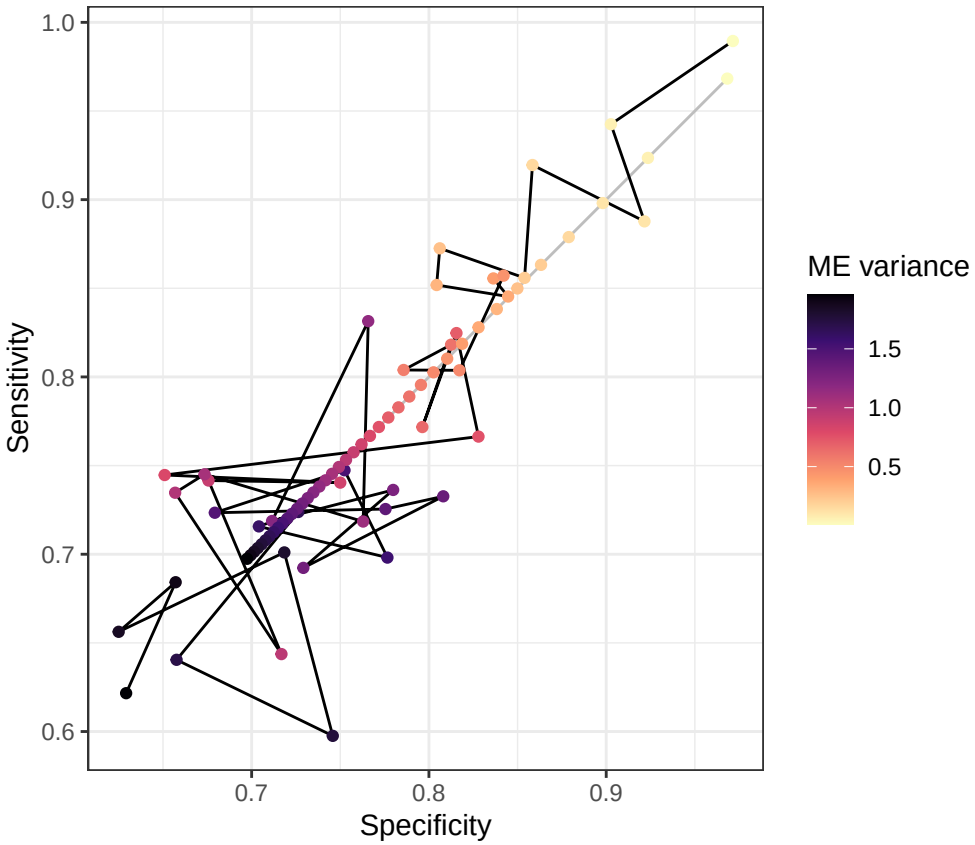
```r
me_variances <- seq(0.01, 2, by = 0.05)
result_df <- data.frame(me_var = NA, sens = NA, spec = NA)
for(i in 1:length(me_variances)){
  var_list <- simulate_dichotomized(me_variance = me_variances[i])
  sens_spec <- calculate_mc(var_list$x_d, var_list$w_d)
  result_df[i,] <- c(me_variances[i], sens_spec$sens, sens_spec$spec)
}
```

Next, we also calculate the expected sensitivity and specificity for different values of measurement error variance given the simulation setup:

```r
sens_spec_exact <- function(sigma_u2 = 1){
  mu_x <- 0
  mu_v <- mu_x
  sigma_x2 <- 1
  sigma_v2 <- sigma_x2 + sigma_u2
  cov_v_x <- sigma_x2+ 2*mu_x^2
  sigma_mat <- matrix(c(sigma_v2, cov_v_x, sigma_x2, cov_v_x), byrow = TRUE, nrow = 2)

  # P(V>0, X>0)
  p_v_x <- mvtnorm::pmvnorm(c(0, 0), mean = c(mu_v, mu_x), sigma = sigma_mat)
  # P(X>0)
  p_x <- pnorm(0, mean = mu_x, sd = sqrt(sigma_x2), lower.tail = FALSE)
  # P(V>0)
  p_v <- pnorm(0, mean = mu_v, sd = sqrt(sigma_v2), lower.tail = FALSE)

  # Sensitivity
  sens <- p_v_x/p_x
  # Specificity
  spec <- (1-(p_v + p_x - p_v_x))/(1-p_x)

  return(list(sens_calc = sens, spec_calc = spec, me_var = sigma_u2))
}

sens_spec_list <- lapply(me_variances, sens_spec_exact)
exact_sens_spec <- as.data.frame(do.call(rbind, sens_spec_list)) %>%
  mutate(across(everything(), as.numeric))
```

We combine the results and plot the sensitivity and specificity for different levels of measurement error variance:

```r
sim_and_calc <- inner_join(exact_sens_spec, result_df, by = "me_var")

ggplot(sim_and_calc) +
  geom_path(aes(x = spec_calc, y = sens_calc), color = "grey") +
  geom_path(aes(x = spec, y = sens)) +
  geom_point(aes(x = spec, y = sens, color = me_var)) +
  geom_point(aes(x = spec_calc, y = sens_calc, color = me_var)) +
  scale_color_viridis_c(option = "magma", direction = -1) +
  coord_equal() +
  labs(x = "Specificity", y = "Sensitivity", color = "ME variance") +
  theme_bw()
```



This plot shows the resulting sensitivity and specificity when the measurement error variance goes from 0.01 to 2. The very jumpy line is the sample sensitivity and specificity from the simulated data, while the straight line indicates the expected values.