Binary data example

Introduction

Population adjustment methods such as *matching-adjusted indirect comparison* (MAIC) are increasingly used to compare marginal treatment effects when there are cross-trial differences in effect modifiers and limited patient-level data. MAIC is based on propensity score weighting, which is sensitive to poor covariate overlap and cannot extrapolate beyond the observed covariate space. Current outcome regression-based alternatives can extrapolate but target a conditional treatment effect that is incompatible in the indirect comparison. When adjusting for covariates, one must integrate or average the conditional estimate over the relevant population to recover a compatible marginal treatment effect.

We propose a marginalization method based on *parametric G-computation* that can be easily applied where the outcome regression is a generalized linear model or a Cox model. The approach views the covariate adjustment regression as a nuisance model and separates its estimation from the evaluation of the marginal treatment effect of interest. The method can accommodate a Bayesian statistical framework, which naturally integrates the analysis into a probabilistic framework. A simulation study provides proof-of-principle and benchmarks the method's performance against MAIC and the conventional outcome regression. Parametric G-computation achieves more precise and more accurate estimates than MAIC, particularly when covariate overlap is poor, and yields unbiased marginal treatment effect estimates under no failures of assumptions. Furthermore, the marginalized regression-adjusted estimates provide greater precision and accuracy than the conditional estimates produced by the conventional outcome regression.

General problem

Consider one AB trial, for which the company has IPD, and one AC trial, for which only published aggregate data are available. We wish to estimate a comparison of the effects of treatments B and C on an appropriate scale in some target population P, denoted by the parameter $d_{BC(P)}$. We make use of bracketed subscripts to denote a specific population. Within the AB population there are parameters $\mu_{A(AB)}$, $\mu_{B(AB)}$ and $\mu_{C(AB)}$ representing the expected outcome on each treatment (including parameters for treatments not studied in the AB trial, e.g. treatment C). The AB trial provides estimators $\bar{Y}_{A(AB)}$ and $\bar{Y}_{B(AB)}$ of $\mu_{A(AB)}$, $\mu_{B(AB)}$, respectively, which are the summary outcomes. It is the same situation for the AC trial.

For a suitable scale, for example a logit, or risk difference, we form estimators $\Delta_{AB(AB)}$ and $\Delta_{AC(AC)}$ of the trial level (or marginal) relative treatment effects.

$$\Delta_{AB(AB)} = g(\bar{Y}_{B(AB)}) - g(\bar{Y}_{A(AB)})$$

Example analysis

First, let us load necessary packages.

```r
library(boot)       # non-parametric bootstrap in MAIC and ML G-computation
library(copula)     # simulating BC covariates from Gaussian copula
library(rstanarm)   # fit outcome regression, draw outcomes in Bayesian G-computation
library(outstandR)
```

Data

We consider binary outcomes using the log-odds ratio as the measure of effect. The binary outcome may be response to treatment or the occurrence of an adverse event. For trials AC and BC, outcome y_n for subject n is simulated from a Bernoulli distribution with probabilities of success generated from logistic regression.

For the BC trial, the individual-level covariates and outcomes are aggregated to obtain summaries. The continuous covariates are summarized as means and standard deviations, which would be available to the analyst in the published study in a table of baseline characteristics in the RCT publication. The binary outcomes are summarized in an overall event table. Typically, the published study only provides aggregate information to the analyst.

Use the `gen_data()` function available with the `outstandR` package.

```r
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
library(MASS)
#>
#> Attaching package: 'MASS'
#> The following object is masked from 'package:dplyr':
#>
#>     select

N <- 200
allocation <- 2/3      # active treatment vs. placebo allocation ratio (2:1)
b_trt <- log(0.17)     # conditional effect of active treatment vs. common comparator
b_X <- -log(0.5)       # conditional effect of each prognostic variable
b_EM <- -log(0.67)     # conditional interaction effect of each effect modifier
meanX_AC <- c(0.45, 0.45)       # mean of normally-distributed covariate in AC trial
meanX_BC <- c(0.6, 0.6)         # mean of each normally-distributed covariate in BC
meanX_EM_AC <- c(0.45, 0.45)    # mean of normally-distributed EM covariate in AC trial
meanX_EM_BC <- c(0.6, 0.6)      # mean of each normally-distributed EM covariate in BC
sdX <- c(0.4, 0.4)     # standard deviation of each covariate (same for AC and BC)
sdX_EM <- c(0.4, 0.4)  # standard deviation of each EM covariate
corX <- 0.2            # covariate correlation coefficient
b_0 <- -0.6            # baseline intercept coefficient  ##TODO: fixed value

AC.IPD <- gen_data(N, b_trt, b_X, b_EM, b_0,
                   meanX_AC, sdX,
                   meanX_EM_AC, sdX_EM,
                   corX, allocation,
                   family = binomial("logit"))
```

Similarly, for the aggregate data but with the additional summarise step.

```r
BC.IPD <- gen_data(N, b_trt, b_X, b_EM, b_0,
                   meanX_BC, sdX,
                   meanX_EM_BC, sdX_EM,
                   corX, allocation,
                    family = binomial("logit"))

cov.X <- BC.IPD %>%
  summarise(across(starts_with("X"),
                   list(mean = mean, sd = sd),
                   .names = "{fn}.{col}"))

out.B <- dplyr::filter(BC.IPD, trt == 1) %>%
  summarise(y.B.sum = sum(y),
            y.B.bar = mean(y),
            y.B.sd = sd(y),
            N.B = n())

out.C <- dplyr::filter(BC.IPD, trt == 0) %>%
  summarise(y.C.sum = sum(y),
            y.C.bar = mean(y),
            y.C.sd = sd(y),
            N.C = n())

BC.ALD <- cbind.data.frame(cov.X, out.C, out.B)
```

The true logistic outcome model which we use to simulate the data will be:

$$\text{logit}(p_{it}) = \beta_0 + \beta_X(X_3 + X_4) + [\beta_{trt} + \beta_{EM}(X_1 + X_2)]\, \text{I}(trt \neq C)$$

So, the log OR for the treatment effect of interest, i.e. AC or BC, is β_{trt} (`b_trt`).

This general format of data sets consist of the following.

`AC.IPD`: Individual patient data

- `X*`: patient measurements
- `trt`: treatment ID (integer)
- `y`: (logical) indicator of whether event was observed

`BC.ALD`: Aggregate-level data

- `mean.X*`: mean patient measurement
- `sd.X*`: standard deviation of patient measurement
- `y.*.sum`: total number of events
- `y.*.bar`: proportion of events
- `N.*`: total number of individuals

Note that the wildcard `*` here is usually an integer from 1 or the trial identifier B, C.

Our data look like the following.

```
head(AC.IPD)
#>          X1         X2        X3        X4 trt y
#> 1 0.4702081  0.3423801 0.7496692 0.7053185   1 1
#> 2 0.7544037  0.1784623 0.3175410 0.6476784   1 1
#> 3 0.7971106  0.4680845 0.5442560 0.6983100   1 0
#> 4 0.6274933  0.4013188 0.3343988 0.3502497   1 0
#> 5 0.8074803  0.8232989 0.4619130 0.7067436   1 0
#> 6 0.4150220 -0.1120460 1.0902699 0.6454173   1 1
```

There are 4 correlated continuous covariates generated per subject, simulated from a multivariate normal distribution. Treatment `trt` 1 corresponds to new treatment A, and 0 is standard of care or status quo C.

```
BC.ALD
#>     mean.X1     sd.X1   mean.X2    sd.X2   mean.X3     sd.X3  mean.X4     sd.X4 y.C.sum   y.C.bar
#> 1 0.5990645 0.4054803 0.6116619 0.417698 0.6037028 0.4056714 0.617801 0.4018668      34 0.5074627 0.
#>      y.B.sd N.B
#> 1 0.4635249 133
```

In this case, we have 4 covariate mean and standard deviation values; and the event total, average and sample size for each treatment B, and C.

Output statistics

We will implement for MAIC, STC, and G-computation methods to obtain the *marginal variance*, defined as

$$\frac{1}{n_C} + \frac{1}{n_{\bar{C}}} + \frac{1}{n_B} + \frac{1}{n_{\bar{B}}}$$

and the *marginal treatment effect*, defined as the log-odds ratio,

$$\log\left(\frac{n_B/(N_B - n_B)}{n_C/(N_B - n_B)}\right) = \log(n_B n_{\bar{C}}) - \log(n_C n_{\bar{B}})$$

where n_B, n_C are the number of events in each arm and $\bar{C}$ is the compliment of C so e.g. $n_{\bar{C}} = N_C - n_c$.

Model fitting in R

The `{outstandR}` package has been written to be easy to use and essential consists of a single function, `outstandR()`. This can be used to run all of the different types of model, which we will call *strategies*. The first two arguments of `outstandR()` are the individual and aggregate-level data, respectively.

A `strategy` argument of `outstandR` takes functions called `strategy_*()`, where the wildcard `*` is replaced by the name of the particular method required, e.g. `strategy_maic()` for MAIC. Each specific example is provided below.

MAIC

Using the individual level data for AC firstly we perform non-parametric bootstrap of the `maic.boot` function with `R = 1000` replicates. This function fits treatment coefficient for the marginal effect for A vs C. The returned value is an object of class `boot` from the `{boot}` package. We then calculate the bootstrap mean and variance in the wrapper function `maic_boot_stats`.

Defining $X_1, X2$ as effect modifiers and X_3, X_4 as prognostic variables then the formula used in this model is

$$y = X_3 + X_4 + \beta_t X_1 + \beta_t X_2$$

which corresponds to the following `R` `formula` object passed as an argument to the strategy function.

```
lin_form <- as.formula("y ~ X3 + X4 + trt*X1 + trt*X2")
```

```
outstandR_maic <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_maic(formula = lin_form,
                                     family = binomial(link = "logit")))
```

The returned object is of class `outstandR`.

```r
outstandR_maic
```

We see that this is a list object with 3 parts, each containing statistics between each pair of treatments. These are the mean contrasts, variances and confidence intervals (CI), respectively. The default CI is for 95% but can be altered in `outstandR` with the `CI` argument.

Outcome scale If we do not explicitly specify the outcome scale, the default is that used for fit the data in the regression model. As we saw, in this case, the default is log-odds ratio corresponding to the `"logit"` link function for binary data. However, we can change this to some other scale which may be more appropriate for a particular analysis. So far implemented in the package, the links and their corresponding relative treatment effect scales are as follows:

Data Type	Model	Scale	Argument
Binary	`logit`	Log-odds ratio	`log_odds`
Count	`log`	Log-risk ratio	`log_relative_risk`
Continuous	`mean`	Mean difference	`risk_difference`

The full list of possible transformed treatment effect scales are: *log-odds ratio*, *log-risk ratio*, *mean difference*, risk difference, hazard ratio, hazard difference.

To change the outcome scale, we can pass the `scale` argument in the `outstandR()` function. For example, to change the scale to risk difference, we can use the following code.

```r
outstandR_maic_lrr <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_maic(formula = lin_form,
                                     family = binomial(link = "logit")),
            scale = "log_relative_risk")
```

```r
outstandR_maic_lrr
```

STC

STC is the conventional outcome regression method. It involves fitting a regression model of outcome on treatment and covariates to the IPD. IPD effect modifiers are centred at the mean *BC* values.

$$g(\mu_n) = \beta_0 + \beta_X(\boldsymbol{x}_n - \boldsymbol{\theta}) + \boldsymbol{\beta_{EM}}(\beta_{trt} + (\boldsymbol{x_n^{EM}} - \boldsymbol{\theta^{EM}}))\, \mathrm{I}(trt \neq C)$$

where β_0 is the intercept, β_1 are the covariate coefficients, β_z and β_2 are the effect modifier coefficients, z_n are the indicator variables of effect alternative treatment. $g(\cdot)$ is the link function e.g. log.

As already mentioned, running the STC analysis is almost identical to the previous analysis but we now use the `strategy_stc()` strategy function instead and a formula with centered covariates.

$$y = X_3 + X_4 + \beta_t(X_1 - \bar{X_1}) + \beta_t(X_2 - \bar{X_2})$$

However, `outstandR()` knows how to handle this so we can simply pass the same (uncentred) formula as before.

```r
outstandR_stc <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_stc(formula = lin_form,
                                    family = binomial(link = "logit")))
outstandR_stc
```

Change the outcome scale

```
outstandR_stc_lrr <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_stc(formula = lin_form,
                                    family = binomial(link = "logit")),
            scale = "log_relative_risk")
```

```
outstandR_stc_lrr
```

For the last two approaches, we perform G-computation firstly with a frequentist MLE approach and then a Bayesian approach.

Parametric G-computation with maximum-likelihood estimation

G-computation marginalizes the conditional estimates by separating the regression modelling from the estimation of the marginal treatment effect for A versus C. First, a regression model of the observed outcome y on the covariates x and treatment z is fitted to the AC IPD:

$$g(\mu_n) = \beta_0 + \boldsymbol{x}_n \boldsymbol{\beta_X} + (\beta_z + \boldsymbol{x_n^{EM}} \boldsymbol{\beta_{EM}})\, \mathrm{I}(trt \neq C)$$

In the context of G-computation, this regression model is often called the “Q-model.” Having fitted the Q-model, the regression coefficients are treated as nuisance parameters. The parameters are applied to the simulated covariates $x*$ to predict hypothetical outcomes for each subject under both possible treatments. Namely, a pair of predicted outcomes, also called potential outcomes, under A and under C, is generated for each subject.

By plugging treatment C into the regression fit for every simulated observation, we predict the marginal outcome mean in the hypothetical scenario in which all units are under treatment C:

$$\hat{\mu}_0 = \int_{x^*} g^{-1}(\hat{\beta}_0 + x^* \hat{\beta}_1) p(x^*)\, \mathrm{d}x^*$$

To estimate the marginal or population-average treatment effect for A versus C in the linear predictor scale, one back-transforms to this scale the average predictions, taken over all subjects on the natural outcome scale, and calculates the difference between the average linear predictions:

$$\hat{\Delta}^{(2)}_{10} = g(\hat{\mu}_1) - g(\hat{\mu}_0)$$

```
outstandR_gcomp_ml <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_gcomp_ml(formula = lin_form,
                                         family = binomial(link = "logit")))
outstandR_gcomp_ml
```

Change the outcome scale

```
outstandR_gcomp_ml_lrr <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_gcomp_ml(formula = lin_form,
                                         family = binomial(link = "logit")),
            scale = "log_relative_risk")
```

```
outstandR_gcomp_ml_lrr
```

Bayesian G-computation with MCMC

The difference between Bayesian G-computation and its maximum-likelihood counterpart is in the estimated distribution of the predicted outcomes. The Bayesian approach also marginalizes, integrates or standardizes over the joint posterior distribution of the conditional nuisance parameters of the outcome regression, as well as the joint covariate distribution.

Draw a vector of size N^* of predicted outcomes y^*_z under each set intervention $z^* \in \{0, 1\}$ from its posterior predictive distribution under the specific treatment. This is defined as $p(y^*_{z^*} \mid \mathcal{D}_{AC}) = \int_\beta p(y^*_{z^*} \mid \beta) p(\beta \mid \mathcal{D}_{AC}) d\beta$ where $p(\beta \mid \mathcal{D}_{AC})$ is the posterior distribution of the outcome regression coefficients β, which encode the predictor-outcome relationships observed in the AC trial IPD. This is given by:

$$p(y^*_{z*} \mid \mathcal{D}_{AC}) = \int_{x^*} p(y^* \mid z^*, x^*, \mathcal{D}_{AC}) p(x^* \mid \mathcal{D}_{AC}) \, \mathrm{d}x^*$$

$$= \int_{x^*} \int_\beta p(y^* \mid z^*, x^*, \beta) p(x^* \mid \beta) p(\beta \mid \mathcal{D}_{AC}) \, d\beta \, \mathrm{d}x^*$$

In practice, the integrals above can be approximated numerically, using full Bayesian estimation via Markov chain Monte Carlo (MCMC) sampling.

The average, variance and interval estimates of the marginal treatment effect can be derived empirically from draws of the posterior density.

We can draw a vector of size N^* of predicted outcomes y^*_z under each set intervention z^* from its posterior predictive distribution under the specific treatment.

```
outstandR_gcomp_stan <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_gcomp_stan(formula = lin_form,
                                           family = binomial(link = "logit")))
#>
#> SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 1).
#> Chain 1:
#> Chain 1: Gradient evaluation took 5.4e-05 seconds
#> Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.54 seconds.
#> Chain 1: Adjust your expectations accordingly!
#> Chain 1:
#> Chain 1:
#> Chain 1: Iteration:    1 / 4000 [  0%]  (Warmup)
#> Chain 1: Iteration:  400 / 4000 [ 10%]  (Warmup)
#> Chain 1: Iteration:  800 / 4000 [ 20%]  (Warmup)
#> Chain 1: Iteration: 1200 / 4000 [ 30%]  (Warmup)
#> Chain 1: Iteration: 1600 / 4000 [ 40%]  (Warmup)
#> Chain 1: Iteration: 2000 / 4000 [ 50%]  (Warmup)
#> Chain 1: Iteration: 2001 / 4000 [ 50%]  (Sampling)
#> Chain 1: Iteration: 2400 / 4000 [ 60%]  (Sampling)
#> Chain 1: Iteration: 2800 / 4000 [ 70%]  (Sampling)
#> Chain 1: Iteration: 3200 / 4000 [ 80%]  (Sampling)
#> Chain 1: Iteration: 3600 / 4000 [ 90%]  (Sampling)
#> Chain 1: Iteration: 4000 / 4000 [100%]  (Sampling)
#> Chain 1:
#> Chain 1:  Elapsed Time: 0.509 seconds (Warm-up)
#> Chain 1:                0.525 seconds (Sampling)
#> Chain 1:                1.034 seconds (Total)
#> Chain 1:
```

```
#>
#> SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 2).
#> Chain 2:
#> Chain 2: Gradient evaluation took 1.7e-05 seconds
#> Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
#> Chain 2: Adjust your expectations accordingly!
#> Chain 2:
#> Chain 2:
#> Chain 2: Iteration:    1 / 4000 [  0%]  (Warmup)
#> Chain 2: Iteration:  400 / 4000 [ 10%]  (Warmup)
#> Chain 2: Iteration:  800 / 4000 [ 20%]  (Warmup)
#> Chain 2: Iteration: 1200 / 4000 [ 30%]  (Warmup)
#> Chain 2: Iteration: 1600 / 4000 [ 40%]  (Warmup)
#> Chain 2: Iteration: 2000 / 4000 [ 50%]  (Warmup)
#> Chain 2: Iteration: 2001 / 4000 [ 50%]  (Sampling)
#> Chain 2: Iteration: 2400 / 4000 [ 60%]  (Sampling)
#> Chain 2: Iteration: 2800 / 4000 [ 70%]  (Sampling)
#> Chain 2: Iteration: 3200 / 4000 [ 80%]  (Sampling)
#> Chain 2: Iteration: 3600 / 4000 [ 90%]  (Sampling)
#> Chain 2: Iteration: 4000 / 4000 [100%]  (Sampling)
#> Chain 2:
#> Chain 2:  Elapsed Time: 0.522 seconds (Warm-up)
#> Chain 2:                0.533 seconds (Sampling)
#> Chain 2:                1.055 seconds (Total)
#> Chain 2:
outstandR_gcomp_stan
```

Change the outcome scale

```
outstandR_gcomp_stan_lrr <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_gcomp_stan(formula = lin_form,
                                           family = binomial(link = "logit")),
            scale = "log_relative_risk")
#>
#> SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 1).
#> Chain 1:
#> Chain 1: Gradient evaluation took 2.3e-05 seconds
#> Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.23 seconds.
#> Chain 1: Adjust your expectations accordingly!
#> Chain 1:
#> Chain 1:
#> Chain 1: Iteration:    1 / 4000 [  0%]  (Warmup)
#> Chain 1: Iteration:  400 / 4000 [ 10%]  (Warmup)
#> Chain 1: Iteration:  800 / 4000 [ 20%]  (Warmup)
#> Chain 1: Iteration: 1200 / 4000 [ 30%]  (Warmup)
#> Chain 1: Iteration: 1600 / 4000 [ 40%]  (Warmup)
#> Chain 1: Iteration: 2000 / 4000 [ 50%]  (Warmup)
#> Chain 1: Iteration: 2001 / 4000 [ 50%]  (Sampling)
#> Chain 1: Iteration: 2400 / 4000 [ 60%]  (Sampling)
#> Chain 1: Iteration: 2800 / 4000 [ 70%]  (Sampling)
#> Chain 1: Iteration: 3200 / 4000 [ 80%]  (Sampling)
#> Chain 1: Iteration: 3600 / 4000 [ 90%]  (Sampling)
#> Chain 1: Iteration: 4000 / 4000 [100%]  (Sampling)
#> Chain 1: 
#> Chain 1:  Elapsed Time: 0.507 seconds (Warm-up)
#> Chain 1:                0.535 seconds (Sampling)
#> Chain 1:                1.042 seconds (Total)
#> Chain 1: 
#> 
#> SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 2).
#> Chain 2: 
#> Chain 2: Gradient evaluation took 1.7e-05 seconds
#> Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
#> Chain 2: Adjust your expectations accordingly!
#> Chain 2: 
#> Chain 2: 
#> Chain 2: Iteration:    1 / 4000 [  0%]  (Warmup)
#> Chain 2: Iteration:  400 / 4000 [ 10%]  (Warmup)
#> Chain 2: Iteration:  800 / 4000 [ 20%]  (Warmup)
#> Chain 2: Iteration: 1200 / 4000 [ 30%]  (Warmup)
#> Chain 2: Iteration: 1600 / 4000 [ 40%]  (Warmup)
#> Chain 2: Iteration: 2000 / 4000 [ 50%]  (Warmup)
#> Chain 2: Iteration: 2001 / 4000 [ 50%]  (Sampling)
#> Chain 2: Iteration: 2400 / 4000 [ 60%]  (Sampling)
#> Chain 2: Iteration: 2800 / 4000 [ 70%]  (Sampling)
#> Chain 2: Iteration: 3200 / 4000 [ 80%]  (Sampling)
#> Chain 2: Iteration: 3600 / 4000 [ 90%]  (Sampling)
#> Chain 2: Iteration: 4000 / 4000 [100%]  (Sampling)
#> Chain 2: 
#> Chain 2:  Elapsed Time: 0.508 seconds (Warm-up)
#> Chain 2:                0.537 seconds (Sampling)
#> Chain 2:                1.045 seconds (Total)
#> Chain 2: 
```

```
outstandR_gcomp_stan_lrr
```

Multiple imputation marginalisation

ref

$$equation here$$

```
outstandR_mim <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_mim(formula = lin_form,
                                    family = binomial(link = "logit")))
#> 
#> SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 1).
#> Chain 1: 
#> Chain 1: Gradient evaluation took 2.2e-05 seconds
#> Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.22 seconds.
#> Chain 1: Adjust your expectations accordingly!
#> Chain 1: 
#> Chain 1: 
#> Chain 1: Iteration:    1 / 4000 [  0%]  (Warmup)
#> Chain 1: Iteration:  400 / 4000 [ 10%]  (Warmup)
#> Chain 1: Iteration:  800 / 4000 [ 20%]  (Warmup)
#> Chain 1: Iteration: 1001 / 4000 [ 25%]  (Sampling)
#> Chain 1: Iteration: 1400 / 4000 [ 35%]  (Sampling)
#> Chain 1: Iteration: 1800 / 4000 [ 45%]  (Sampling)
#> Chain 1: Iteration: 2200 / 4000 [ 55%]  (Sampling)
#> Chain 1: Iteration: 2600 / 4000 [ 65%]  (Sampling)
#> Chain 1: Iteration: 3000 / 4000 [ 75%]  (Sampling)
#> Chain 1: Iteration: 3400 / 4000 [ 85%]  (Sampling)
#> Chain 1: Iteration: 3800 / 4000 [ 95%]  (Sampling)
#> Chain 1: Iteration: 4000 / 4000 [100%]  (Sampling)
#> Chain 1:
#> Chain 1:  Elapsed Time: 0.244 seconds (Warm-up)
#> Chain 1:                0.615 seconds (Sampling)
#> Chain 1:                0.859 seconds (Total)
#> Chain 1:
#>
#> SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 2).
#> Chain 2:
#> Chain 2: Gradient evaluation took 1.6e-05 seconds
#> Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
#> Chain 2: Adjust your expectations accordingly!
#> Chain 2:
#> Chain 2:
#> Chain 2: Iteration:    1 / 4000 [  0%]  (Warmup)
#> Chain 2: Iteration:  400 / 4000 [ 10%]  (Warmup)
#> Chain 2: Iteration:  800 / 4000 [ 20%]  (Warmup)
#> Chain 2: Iteration: 1001 / 4000 [ 25%]  (Sampling)
#> Chain 2: Iteration: 1400 / 4000 [ 35%]  (Sampling)
#> Chain 2: Iteration: 1800 / 4000 [ 45%]  (Sampling)
#> Chain 2: Iteration: 2200 / 4000 [ 55%]  (Sampling)
#> Chain 2: Iteration: 2600 / 4000 [ 65%]  (Sampling)
#> Chain 2: Iteration: 3000 / 4000 [ 75%]  (Sampling)
#> Chain 2: Iteration: 3400 / 4000 [ 85%]  (Sampling)
#> Chain 2: Iteration: 3800 / 4000 [ 95%]  (Sampling)
#> Chain 2: Iteration: 4000 / 4000 [100%]  (Sampling)
#> Chain 2:
#> Chain 2:  Elapsed Time: 0.269 seconds (Warm-up)
#> Chain 2:                0.633 seconds (Sampling)
#> Chain 2:                0.902 seconds (Total)
#> Chain 2:
outstandR_mim
```

Change the outcome scale

```
outstandR_mim_lrr <-
  outstandR(AC.IPD, BC.ALD,
            strategy = strategy_mim(formula = lin_form,
                                    family = binomial(link = "logit")),
            scale = "log_relative_risk")
#>
#> SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 1).
#> Chain 1:
#> Chain 1: Gradient evaluation took 2.2e-05 seconds
#> Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.22 seconds.
#> Chain 1: Adjust your expectations accordingly!
#> Chain 1:
#> Chain 1:
#> Chain 1: Iteration:    1 / 4000 [  0%]  (Warmup)
#> Chain 1: Iteration:  400 / 4000 [ 10%]  (Warmup)
#> Chain 1: Iteration:  800 / 4000 [ 20%]  (Warmup)
#> Chain 1: Iteration: 1001 / 4000 [ 25%]  (Sampling)
#> Chain 1: Iteration: 1400 / 4000 [ 35%]  (Sampling)
#> Chain 1: Iteration: 1800 / 4000 [ 45%]  (Sampling)
#> Chain 1: Iteration: 2200 / 4000 [ 55%]  (Sampling)
#> Chain 1: Iteration: 2600 / 4000 [ 65%]  (Sampling)
#> Chain 1: Iteration: 3000 / 4000 [ 75%]  (Sampling)
#> Chain 1: Iteration: 3400 / 4000 [ 85%]  (Sampling)
#> Chain 1: Iteration: 3800 / 4000 [ 95%]  (Sampling)
#> Chain 1: Iteration: 4000 / 4000 [100%]  (Sampling)
#> Chain 1:
#> Chain 1:  Elapsed Time: 0.241 seconds (Warm-up)
#> Chain 1:                0.789 seconds (Sampling)
#> Chain 1:                1.03 seconds (Total)
#> Chain 1:
#>
#> SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 2).
#> Chain 2:
#> Chain 2: Gradient evaluation took 1.7e-05 seconds
#> Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
#> Chain 2: Adjust your expectations accordingly!
#> Chain 2:
#> Chain 2:
#> Chain 2: Iteration:    1 / 4000 [  0%]  (Warmup)
#> Chain 2: Iteration:  400 / 4000 [ 10%]  (Warmup)
#> Chain 2: Iteration:  800 / 4000 [ 20%]  (Warmup)
#> Chain 2: Iteration: 1001 / 4000 [ 25%]  (Sampling)
#> Chain 2: Iteration: 1400 / 4000 [ 35%]  (Sampling)
#> Chain 2: Iteration: 1800 / 4000 [ 45%]  (Sampling)
#> Chain 2: Iteration: 2200 / 4000 [ 55%]  (Sampling)
#> Chain 2: Iteration: 2600 / 4000 [ 65%]  (Sampling)
#> Chain 2: Iteration: 3000 / 4000 [ 75%]  (Sampling)
#> Chain 2: Iteration: 3400 / 4000 [ 85%]  (Sampling)
#> Chain 2: Iteration: 3800 / 4000 [ 95%]  (Sampling)
#> Chain 2: Iteration: 4000 / 4000 [100%]  (Sampling)
#> Chain 2:
#> Chain 2:  Elapsed Time: 0.259 seconds (Warm-up)
#> Chain 2:                0.715 seconds (Sampling)
#> Chain 2:                0.974 seconds (Total)
#> Chain 2:
```

```
outstandR_mim_lrr
```

Model comparison

***AC* effect in *BC* population** The true *AC* effect on the log OR scale in the *BC* (aggregate trial data) population is $\beta_{trt}^{AC} + \beta_{EM}(\bar{X}_1^{AC} + \bar{X}_2^{AC})$. Calculated by

```
d_AC_true <- b_trt + b_EM * (BC.ALD$mean.X1 + BC.ALD$mean.X2)
```

The naive approach is to just convert directly from one population to another, ignoring the imbalance in effect modifiers.

```
d_AC_naive <-
  AC.IPD |>
  group_by(trt) |>
  summarise(y_sum = sum(y), y_bar = mean(y), n = n()) |>
  tidyr::pivot_wider(names_from = trt,
                     values_from = c(y_sum, y_bar, n)) |>
  mutate(d_AC =
           log(y_bar_1/(1-y_bar_1)) - log(y_bar_0/(1-y_bar_0)),
         var_AC =
           1/(n_0-y_sum_0) + 1/y_sum_0 + 1/(n_1-y_sum_1) + 1/y_sum_1)
```

AB effect in BC population This is the indirect effect. The true AB effect in the BC population is $\beta_{trt}^{AC} - \beta_{trt}^{BC}$.

Following the simulation study in Remiro et al (2020) these cancel out and the true effect is zero.

The naive comparison calculating AB effect in the BC population is

```
d_BC <-
  with(BC.ALD, log(y.B.bar/(1-y.B.bar)) - log(y.C.bar/(1-y.C.bar)))

d_AB_naive <- d_AC_naive$d_AC - d_BC

var.d.BC <- with(BC.ALD, 1/y.B.sum + 1/(N.B - y.B.sum) + 1/y.C.sum + 1/(N.C - y.C.sum))
var.d.AB.naive <- d_AC_naive$var_AC + var.d.BC
```

Of course, the BC contrast is calculated directly.

Results

We will combine all outputs for a log-odds ratio table of all contrasts and methods in the BC population.

```
res_tab <-
  data.frame(
    d_true = c(d_AB_true, d_AC_true, d_BC),
    d_naive = c(d_AB_naive, d_AC_naive$d_AC, d_BC),
    `MAIC` = unlist(outstandR_maic$contrasts),
    `STC` = unlist(outstandR_stc$contrasts),
    `Gcomp ML` = unlist(outstandR_gcomp_ml$contrasts),
    `Gcomp Bayes` = unlist(outstandR_gcomp_stan$contrasts),
    `MIM` = unlist(outstandR_mim$contrasts))

res_tab_var <-
  data.frame(
    d_true = c(NA, NA, NA),
    d_naive = c(var.d.AB.naive, d_AC_naive$var_AC, var.d.BC),
    `MAIC` = unlist(outstandR_maic$variances),
    `STC` = unlist(outstandR_stc$variances),
    `Gcomp ML` = unlist(outstandR_gcomp_ml$variances),
    `Gcomp Bayes` = unlist(outstandR_gcomp_stan$variances),
    `MIM` = unlist(outstandR_mim$variances))

knitr::kable(res_tab)
```

	d_true	d_naive	MAIC	STC	Gcomp.ML	Gcomp.Bayes	MIM
AB	0.0000000	-0.9600106	-0.9218652	-1.3219017	-1.0226613	-1.0297326	-1.0376828
AC	-1.2870881	-1.7980800	-1.7599347	-2.1599712	-1.8607308	-1.8678020	-1.8757523
BC	-0.8380695	-0.8380695	-0.8380695	-0.8380695	-0.8380695	-0.8380695	-0.8380695

```r
d_AB_true_lrr <- 0
d_AC_true_lrr <- log(plogis(d_A_true) / plogis(d_C_true))
```

On log relative risk scale the table is

```r
res_tab_lrr <-
  data.frame(
    # d_true = c(d_AB_true_lrr, d_AC_true_lrr, d_BC_lrr),
    # d_naive = c(d_AB_naive_lrr, d_AC_naive_lrr$d_AC, d_BC_lrr),
    `MAIC` = unlist(outstandR_maic_lrr$contrasts),
    `STC` = unlist(outstandR_stc_lrr$contrasts),
    `Gcomp ML` = unlist(outstandR_gcomp_ml_lrr$contrasts),
    `Gcomp Bayes` = unlist(outstandR_gcomp_stan_lrr$contrasts),
    `MIM` = unlist(outstandR_mim_lrr$contrasts))

knitr::kable(res_tab_lrr)
```

	MAIC	STC	Gcomp.ML	Gcomp.Bayes	MIM
AB	-0.3824215	-0.853833	-0.378691	-0.3865399	-0.3707505
AC	-0.8808665	-1.352278	-0.877136	-0.8849849	-0.8691955
BC	-0.4984450	-0.498445	-0.498445	-0.4984450	-0.4984450

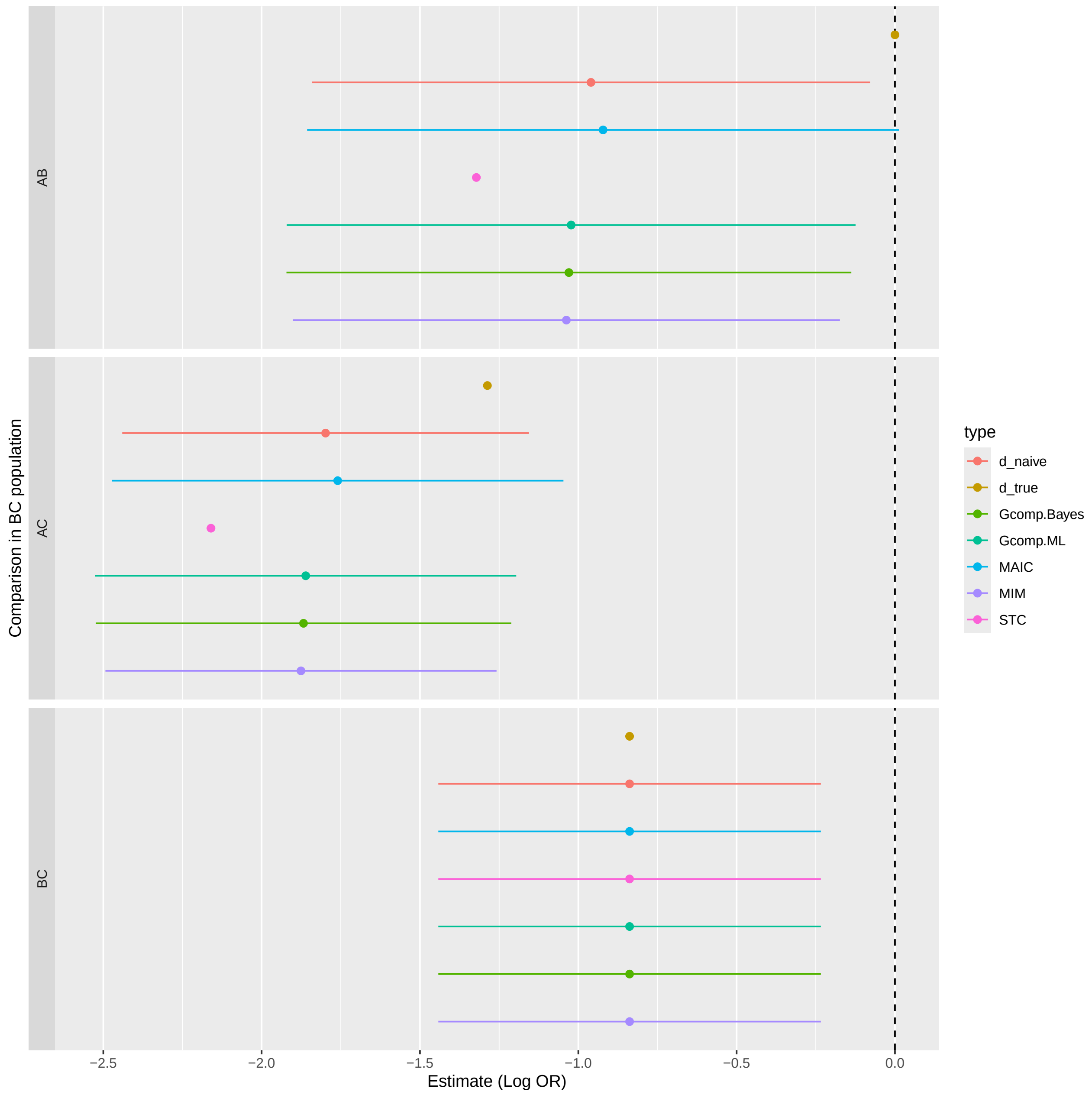
The same output in a forest plot is

```r
library(ggplot2)

var_dat <-
  t(res_tab_var) |>
  as.data.frame() |>
  tibble::rownames_to_column("type") |>
  reshape2::melt(variable.name = "Comparison",
                 value.name = "var")

plotdat <-
  t(res_tab) |>
  as.data.frame() |>
  tibble::rownames_to_column("type") |>
  reshape2::melt(variable.name = "Comparison",
                 value.name = "Estimate") |>
  mutate(id = 1:n(),
         type = as.factor(type)) |>
  merge(var_dat) |>
  mutate(lo = Estimate + qnorm(0.025) * sqrt(var),
         hi = Estimate + qnorm(0.975) * sqrt(var))

ggplot(aes(x = Estimate, y = id, col = type), data = plotdat) +
  geom_vline(xintercept = 0, lty = 2) +
  geom_point(size = 2) +
  geom_segment(aes(y = id, yend = id, x = lo, xend = hi), na.rm = TRUE) +
  xlab("Estimate (Log OR)") +
  facet_grid(Comparison~., switch = "y", scales = "free_y", space = "free_y") +
  scale_y_reverse(name = "Comparison in BC population",
                  breaks = NULL, expand = c(0, 0.6))
```


AB
AC
BC
Comparison in BC population
Estimate (Log OR)
type
d_naive
d_true
Gcomp.Bayes
Gcomp.ML
MAIC
MIM
STC