gapclosing: An R package

Ian Lundberg (ilundberg@cornell.edu)

2023-03-08

Contents

Welcome! **1**

1 Big idea **2**
1.1 Motivation . . . 2
1.2 Data structure . . . 2
1.3 Coding from scratch . . . 3
1.4 Basic package functionality . . . 4

2 Concepts in detail **6**
2.1 Define the intervention . . . 7
2.2 Make causal assumptions for identification . . . 7
2.3 Specify a treatment model and/or an outcome model . . . 8

3 Machine learning examples **10**
3.1 Generalized Additive Models . . . 10
3.2 Random forests . . . 11
3.3 Estimates from these three algorithms are roughly the same . . . 11
3.4 Word of warning . . . 12

4 Advanced treatment assignments **12**
4.1 Stochastic treatments . . . 13
4.2 Individualized treatments . . . 13

5 Conclusion **14**
5.1 Computing environment . . . 14

Welcome!

This website documents the **gapclosing** package for R. The package helps answers questions of the form: to what degree would a hypothetical intervention close gaps across populations?

This causal question is of paramount importance for those who want to intervene to reduce disparities across categories such as gender, race, and class. How to answer that question is the subject of the companion papaer to this package.

> Lundberg, Ian. 2022. **The gap-closing estimand: A causal approach to study interventions that close disparities across social categories.** Sociological Methods and Research.

To get started with these methods, first install R and RStudio. Then install the package from CRAN.

```
install.packages("gapclosing")
```

You can now do lots of things!

- Estimate treatment and outcome prediction functions statistical and machine learning methods
- Combine those in doubly-robust estimators of gap-closing estimands
- Produce confidence intervals by the bootstrap
- Visualize the result

Questions and comments to Ian Lundberg, ilundberg@cornell.edu.

1 Big idea

This part introduces the big ideas: why we should do this, what data look like, and what's going on under the hood. It closes with basic package functionality.

1.1 Motivation

Gaps across social categories like race, class, and gender are important to understand. We would like to know whether there is anything we can do to close these gaps. What if we intervened to reduce incarceration or increase access to education? Would those interventions close gaps across categories of race, class, or gender?

These types of questions are at the core of a growing literature in epidemiology addresses these questions with techniques for causal decomposition analysis (VanderWeele and Robinson (2014), Jackson and VanderWeele (2018), Jackson (2020)). This package provides software to support inquiry into gap-closing estimands, as discussed in Lundberg (2022).

A guiding principle is to distinguish human tasks from software tasks that can be automated.

As a human, you will:

- define the intervention
- make causal assumptions for identification
- specify a treatment model and/or an outcome model

Then package will:

- estimate models
- produce doubly-robust estimates
- sample split to improve convergence
- estimate standard errors by bootstrapping
- visualize the result

1.2 Data structure

In a data frame `data`, we have a gap-defining category such as race, gender, or class. We have a binary treatment variable that could have counterfactually been different for any individual. We want to know the degree to which an intervention to change the treatment would close gaps across the categories.

> **These boxes will present an example.**
>
> *Example.* Suppose we have the following data.
>
> - X (`category`): Category of interest, taking values {A, B, C}
> - T (`treatment`): Binary treatment variable, taking values 0 and 1
> - L (`confounder`): A continuous confounding variable, Uniform(-1,1)
> - Y (`outcome`): A continuous outcome variable, conditionally normal

```
set.seed(08544)
library(gapclosing)
```

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

```
simulated_data <- generate_simulated_data(n = 1000)
head(simulated_data)
#>   category confounder treatment    outcome
#> 1        C   1.640509         1 -0.7970377
#> 2        B   1.032373         0  1.9304909
#> 3        C   2.217630         1 -0.2009803
#> 4        A  -1.642914         0 -0.5015702
#> 5        A  -1.844897         0 -1.2027025
#> 6        A  -2.415100         0 -3.9288188
```

1.3 Coding from scratch

With the most simple models, you can carry out a gap-closing analysis without the software package. First, fit any prediction function for the outcome as a function of the category of interest, confounders, and treatment.

```
example_ols <- lm(outcome ~ category*treatment + confounder,
                  data = simulated_data)
```

For everyone in the sample, predict under a counterfactual treatment value (e.g., `treatment = 1`).

```
fitted <- simulated_data %>%
  mutate(outcome_under_treatment_1 = predict(example_ols,
                                             newdata = simulated_data %>%
                                               mutate(treatment = 1)))
```

Average the counterfactual estimates within each category.

```
fitted %>%
  # Group by the category of interest
  group_by(category) %>%
  # Take the average prediction
  summarize(factual = mean(outcome),
            counterfactual = mean(outcome_under_treatment_1))
#> # A tibble: 3 x 3
#>   category factual counterfactual
#>   <chr>      <dbl>          <dbl>
#> 1 A        -1.09           0.0488
#> 2 B        -0.0384        -0.0370
#> 3 C         0.495          0.132
```

The software package supports these steps as well as more complex things you might want:

- three estimation strategies
  - outcome prediction
  - treatment prediction
  - doubly robust estimation
- machine learning prediction functions
- counterfactual treatments that differ across units
- bootstrapping for standard errors
- easy visualization

1.4 Basic package functionality

The `gapclosing()` function estimates gaps across categories and the degree to which they would close under the specified `counterfactual_assignments` of the treatment.

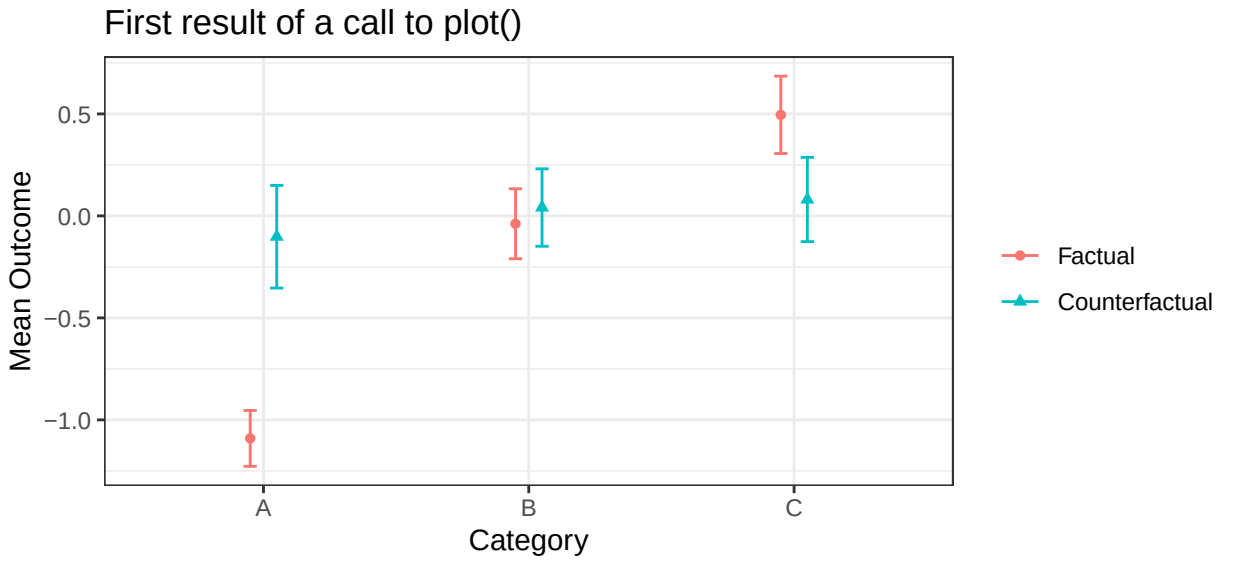
```
estimate <- gapclosing(
  data = simulated_data,
  counterfactual_assignments = 1,
  outcome_formula = formula(outcome ~ confounder + category*treatment),
  treatment_formula = formula(treatment ~ confounder + category),
  category_name = "category",
  se = TRUE,
  # Setting bootstrap_samples very low to speed this tutorial
  # Should be set higher in practice
  bootstrap_samples = 20,
  # You can process the bootstrap in parallel with as many cores as available
  parallel_cores = 1
)
```

By default, this function will do the following:

- Fit logistic regression to predict treatment assignment
- Fit OLS regression to predict outcomes
- Combine the two in a doubly-robust estimator estimated on a single sample
- Return a `gapclosing` object which supports `summary`, `print`, and `plot` functions.

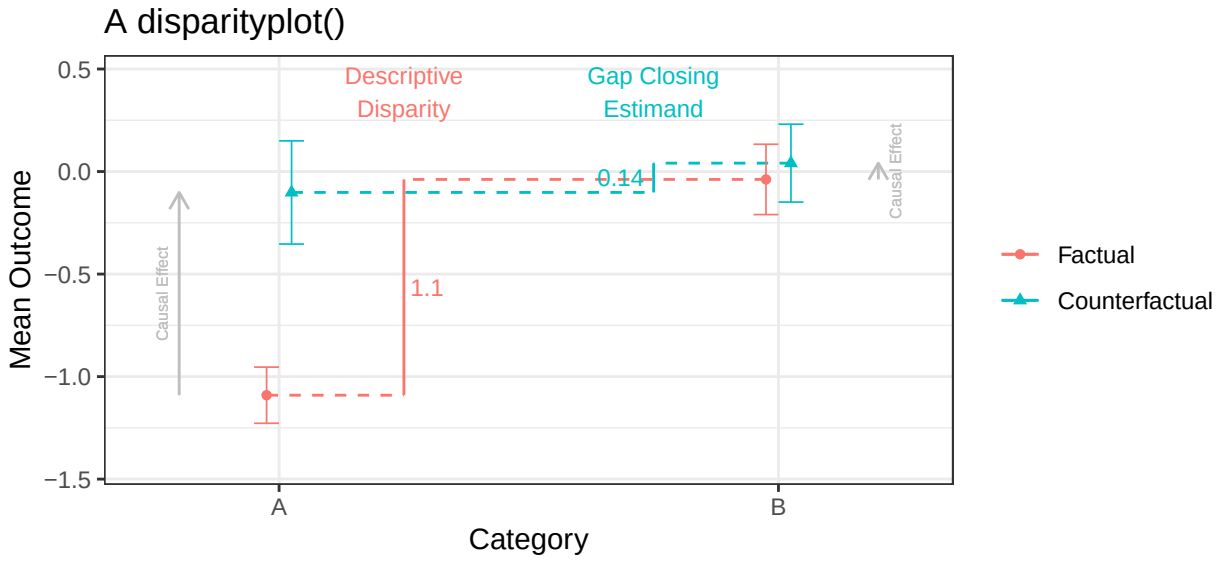
In this example, the `plot(estimate)` function produces the following visualization. The factual outcomes are unequal across categories, but the counterfactual outcomes are roughly equal. In this simulated setting, the intervention almost entirely closes the gaps across the categories.

```
plot(estimate)
```



The `disparityplot()` function lets us zoom in on the factual and counterfactual disparity between two categories, of interest. In this case, we see that the intervention lifts outcomes in category A to be more comparable to category B. A `disparityplot` is a `ggplot2` object and can be customized by passing additional layers.

```
disparityplot(estimate, category_A = "A", category_B = "B") +
  ggtitle("A disparityplot()")
```



The `summary` function will print estimates, standard errors, and confidence intervals for all of these results.

```
summary(estimate)
#> Gap-closing estimates using doubly_robust estimation on one sample.
#>
#> Treatment model was glm estimation with model formula:
#> formula(treatment ~ confounder + category)
#>
#> Outcome model was lm estimation with model formula:
#> formula(outcome ~ confounder + category * treatment)
#>
#> Factual estimates are means within and disparities across category.
#> Counterfactual estimates are under an intervention to set  to 1.
#> Standard errors are calculated from 20 bootstrap samples.
#>
#> Factual mean outcomes:
#> # A tibble: 3 x 5
#>   category estimate     se ci.min ci.max
#>   <chr>       <dbl>  <dbl>  <dbl>  <dbl>
#> 1 A         -1.09   0.0698 -1.23  -0.954
#> 2 B         -0.0384 0.0875 -0.210  0.133
#> 3 C          0.495  0.0968  0.306  0.685
#>
#> Counterfactual mean outcomes (post-intervention means):
#> # A tibble: 3 x 5
#>   category estimate     se ci.min ci.max
#>   <chr>       <dbl>  <dbl>  <dbl>  <dbl>
#> 1 A         -0.102  0.128  -0.354  0.150
#> 2 B          0.0409 0.0969 -0.149  0.231
#> 3 C          0.0805 0.105  -0.126  0.287
#>
#> Factual disparities:
#> # A tibble: 6 x 5
#>   category estimate    se ci.min ci.max
#>   <chr>       <dbl> <dbl>  <dbl>  <dbl>
#> 1 A - B      -1.05  0.121 -1.29  -0.815
```

```
#> 2 A - C      -1.59  0.128 -1.84  -1.33
#> 3 B - A       1.05  0.121  0.815  1.29
#> 4 B - C      -0.534 0.134 -0.796 -0.272
#> 5 C - A       1.59  0.128  1.33   1.84
#> 6 C - B       0.534 0.134  0.272  0.796
#>
#> Counterfactual disparities (gap-closing estimands):
#> # A tibble: 6 x 5
#>   category estimate    se ci.min ci.max
#>   <chr>       <dbl> <dbl>  <dbl>  <dbl>
#> 1 A - B     -0.143  0.161 -0.459  0.173
#> 2 A - C     -0.183  0.165 -0.505  0.140
#> 3 B - A      0.143  0.161 -0.173  0.459
#> 4 B - C     -0.0396 0.168 -0.369  0.290
#> 5 C - A      0.183  0.165 -0.140  0.505
#> 6 C - B      0.0396 0.168 -0.290  0.369
#>
#> Additive gap closed: Counterfactual - Factual
#> # A tibble: 6 x 5
#>   category estimate     se ci.min ci.max
#>   <chr>       <dbl>  <dbl>  <dbl>  <dbl>
#> 1 A - B      -0.909 0.154  -1.21  -0.608
#> 2 A - C      -1.40  0.132  -1.66  -1.14
#> 3 B - A       0.909 0.154   0.608  1.21
#> 4 B - C      -0.494 0.0957 -0.682 -0.307
#> 5 C - A       1.40  0.132   1.14   1.66
#> 6 C - B       0.494 0.0957  0.307  0.682
#>
#> Proportional gap closed: (Counterfactual - Factual) / Factual
#> # A tibble: 6 x 5
#>   category estimate     se ci.min ci.max
#>   <chr>       <dbl>  <dbl>  <dbl>  <dbl>
#> 1 A - B       0.864 0.141   0.589   1.14
#> 2 A - C       0.885 0.0955  0.698   1.07
#> 3 B - A       0.864 0.141   0.589   1.14
#> 4 B - C       0.926 0.323   0.292   1.56
#> 5 C - A       0.885 0.0955  0.698   1.07
#> 6 C - B       0.926 0.323   0.292   1.56
#>
#> Type plot(name_of_this_object) to visualize results.
```

2 Concepts in detail

This section provides a more detailed overview of the use of `gapclosing()`.

It is structured from the perspective of the three key tasks for the researcher:

1. define the intervention
2. make causal assumptions for identification
3. specify a treatment and/or outcome model

Along the way, this section introduces many of the possible arguments in a `gapclosing()` call.

2.1 Define the intervention

To answer a gap-closing question, we first need to define what that intervention would be. To what treatment value would units be counterfactually assigned? There are several options.

- **Option 1.** Set `counterfactual_assignments = 0` or `counterfactual_assignments = 1`. In this case, we are studying the disparity we would expect if a random person of each category were assigned control or if assigned treatment (respectively).
- **Option 2.** Set `counterfactual_assignments` $= \pi$ for some π between 0 and 1. In this case, we are studying the disparity we would expect if a random person of each category were assigned to treatment with probability π and to control with probability $1 - \pi$.
- **Option 3.** Assign treatments by a probability π_i that may differ for each person i. In this case, we create a vector $\vec{\pi}$ of length n and pass that in the `counterfactual_assignments` argument.

  Example. We are interested in the disparity across populations defined by `category` that would persist under counterfactual assignment to set `treatment` to the value 1. `counterfactual_assignments = 1`

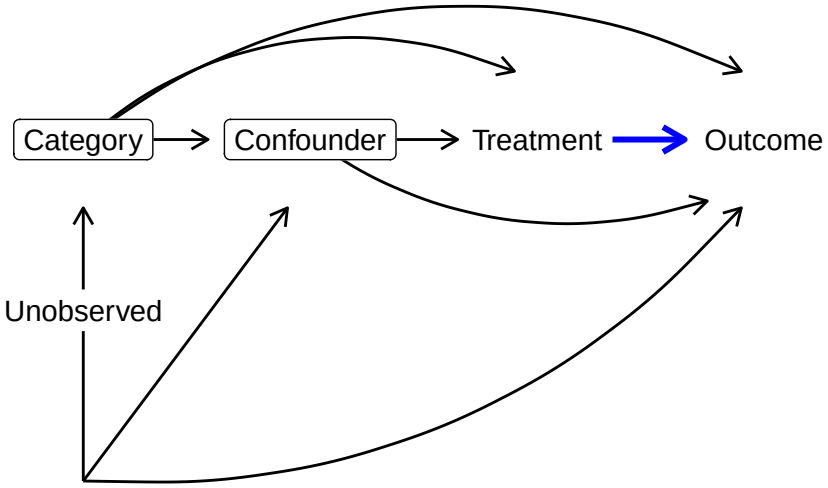
2.2 Make causal assumptions for identification

The package does help you with this step. Gap-closing estimands involve unobserved potential outcomes. Because they are unobserved, the data cannot tell us which variables are needed for estimation. Instead, that is a conceptual choice to be carried out with tools like Directed Acyclic Graphs (DAGs). See the accompanying Lundberg (2022) paper for more on identification.

Example. Assume that the set of variables $\{X, L\}$ is a sufficient conditioning set to identify the gap-closing estimand. Formally, this requires us to assume that within each stratum of X and L the expected value of the potential outcome $Y(1)$ is the same as the expected value among units who factually have $T = 1$ within those strata.

$$\mathbb{E}(Y(1) \mid X, L) = \mathbb{E}(Y \mid X, L, T = 1)$$

DAGs are a good way to reason about this assumption: in this example, conditioning (depicted by boxes) on `category` and `confounder` is sufficient to identify the causal effect of `treatment` (blue edge in the DAG), because doing so blocks all backdoor paths between the treatment and the outcome. Notably, the gap-closing estimand makes no claims about the causal effect of `category` since the counterfactual is defined over `treatment` only.



Once we select a sufficient conditioning set, those predictors will appear in both the treatment and/or the outcome model used for estimation.

2.3 Specify a treatment model and/or an outcome model

We need to estimate one or both of (1) the probability of treatment given confounders and (2) the conditional mean of the outcome given treatment and confounders. We do that by providing one or both of the following.

- Provide a `treatment_formula` with the binary treatment variable on the left side and all confounders on the right side. This formula will be used in `treatment_algorithm` (see next step) to estimate the probability of treatment given confounders for each unit. Then, the sample-average outcome under counterfactual assignment to any given treatment can be estimated through inverse probability weighting among units who factually received that treatment.

  Example.
  `treatment_formula = formula(outcome ~ confounder + category)`

- Provide an `outcome_formula` with the outcome variable on the left side and the treatment and all confounders on the right side. This formula will be used in `outcome_algorithm` (see next step) to estimate the conditional mean of the outcome given treatment and confounders. Then, it can be used to predict the potential outcome each unit would expect under any treatment of interest, given confounding variables.

  Example.
  `outcome_formula = formula(outcome ~ confounder + category*treatment)`

Whether `treatment_formula` or `outcome_formula` is left `NULL` will determine the estimation procedure.

- If only `treatment_formula` is provided, then estimation will be by inverse probability of treatment weighting.
- If only `outcome_formula` is provided, then estimation will be by prediction of unobserved potential outcomes (the *g*-formula, see Hernán and Robins (2021)).
- If both `treatment_formula` and `outcome_formula` are provided, then the primary estimate will be doubly robust, with separate treatment and outcome modeling estimates accessible through the returned object.

2.3.1 Choose an estimation algorithm

The `treatment_formula` and `outcome_formula` are handed to `treatment_algorithm` and `outcome_algorithm`, which can take the following values.

- `glm` (for `treatment_algorithm` only): A logistic regression model estimated by the `glm` function. If there are important interactions among treatment, category, and covariates, these should be included explicitly.
- `lm` (for `outcome_algorithm` only): An OLS regression model estimated by the `lm` function. If there are important interactions among treatment, category, and covariates, these should be included explicitly.
- `ridge`: A ridge (i.e. L2-penalized) logistic regression model (for `treatment_algorithm`) or linear regression model (for `outcome_algorithm`) estimated by the `glmnet` function in the `glmnet` package (Friedman et al., 2010), with `alpha = 0` to indicate the ridge penalty. Elastic net and lasso regression are not supported because those approaches could regularize some coefficients to exactly zero, and if you have chosen the needed confounders (step 2) then you would not want their coefficients to be regularized to zero. The penalty term is chosen by `cv.glmnet` and is set to the value `lambda.min` (see `glmnet` documentation). If there are important interactions among treatment, category, and covariates, these should be included explicitly.
- `gam`: A Generalized Additive Model (logistic if used as `treatment_algorithm`, linear if used as `outcome_algorithm`) estimated by the `gam` function in the `mgcv` package (Wood, 2017). The model formula should be specified as in the `mgcv` documentation and may include smooth terms `s()` for continuous covariates. If there are important interactions among treatment, category, and covariates, these should be included explicitly.
- `ranger`: A random forest estimated by the `ranger` function in the `ranger` package (Wright and Ziegler, 2017). If used as `treatment_algorithm`, one forest will be fit and predicted treatment probabilities

will be truncated to the [.001,.999] range to avoid extreme inverse probability of treatment weights. If used as `outcome_algorithm`, the forest will be estimated separately on treated and control units; the treatment variable does not need to be included in `outcome_formula` in this case.

Example.
```
treatment_algorithm = "glm"
outcome_algorithm = "lm"
```

If the data are a sample from a population selected with unequal probabilities, you can also use the `weight_name` option to pass estimation functions the name of the sampling weight (a variable in `data` proportional to the inverse probability of sample inclusion). If omitted, a simple random sample is assumed.

2.3.2 Why doubly robust? A side note

Doubly-robust estimation yields advantages that can be conceptualized in two ways.

- From the perspective of parametric models, if either the treatment or the outcome model is correct, then the estimator is consistent (Bang and Robins, 2005). See Glynn and Quinn (2010) for an accessible introduction.

- From the perspective of machine learning, double robustness adjusts for the fact that outcome modeling alone is optimized for the wrong prediction task. An outcome model would be optimized to predict where we observed data, but our actual task is to predict over a predictor distribution *different* from that which was observed (because the treatment has been changed). This is an opportunity to improve the predictions. By building a model of treatment, we can reweight the residuals of the outcome model to estimate the average prediction error over the space where we want to make predictions. Subtracting off this bias can improve the outcome modeling estimator. This pivot is at the core of moves toward targeted learning (Van der Laan and Rose, 2011) and double machine learning (Chernozhukov et al., 2018), building on a long line of research in efficient estimation (Robins and Rotnitzky, 1995; Hahn, 1998).

Although double robustness has strong mathematical properties, in any given application with a finite sample it is possible that treatment or outcome modeling could outperform doubly-robust estimation. Therefore, the package supports all three approaches.

2.3.3 Why sample splitting? Another side note

Taking the bias-correction view of double robustness above, it is clear that sample splitting affords a further opportunity for improvement: if you learn an outcome model *and* estimate its average bias on the same sample, you might get a poor estimate of the bias. For this reason, one should consider using one sample (which I call `data_learn`) to learn the prediction functions and another sample (which I call `data_estimate`) to estimate the bias and aggregate to an estimate of the estimand.

In particular, the option `sample_split = "cross_fit"` allows the user to specify that estimation should proceed by a cross-fitting procedure which is analogous to cross-validation.

1. Split the sample into folds $f = 1, \ldots,$`n_folds` (default here is `n_folds = 2`)

2. Use all folds except f' to estimate the treatment and outcome models

3. Aggregate to an estimate using the predictions in f'

4. Average the estimate that results from (2) and (3) repeated `n_folds` times with each fold playing the role of f' in turn

This is the procedure that Chernozhukov et al. (2018) argue is critical to double machine learning for causal estimation, although this type of sample splitting is not new (Bickel, 1982).

If `sample_split = "cross_fit"`, the default is to conduct 2-fold cross-fitting, but this can be changed with the `n_folds` argument. The user can also specify their own vector `folds` of fold assignments of length `nrow(data)`, if there is something about the particular setting that would make a manual fold assignment preferable.

Example. (this is the default and can be left implicit)
`sample_split = "one_sample"`

2.3.4 Produce standard errors

The package supports bootstrapped standard error estimation. The procedure with `bootstrap_method = "simple"` (the default) is valid when the data are a simple random sample from the target population. In this case, each bootstrap iteration conducts estimation on a resampled dataset selected with replacement with equal probabilities. The standard error is calculated as the standard deviation of the estimate across bootstrap samples, and confidence intervals are calculated by a normal approximation.

Example. (line 2 is the default and can be left implicit)
`se = T bootstrap_samples = 1000`

In some settings, the sample size may be small and categories or treatments of interest may be rare. In these cases, it is possible for one or more simple bootstrap samples to contain zero cases in some (treatment $\times$ category) cell of interest. To avoid this problem, `bootstrap_method = "stratified"` conducts bootstrap resampling within blocks defined by (treatment $\times$ category). This procedure is valid if you assume that the data are selected at random from the population within these strata, so that across repeated samples from the true population the proportion in each stratum would remain the same.

Many samples are not simple random samples. In complex sample settings, users should implement their own standard error procedures to accurately capture sampling variation related to how their data were collected. The way the data were collected could motivate a resampling strategy to mimic the sources of variation in that sampling process, which the user can implement manually by calling `gapclosing` to calculate a point estimate on each resampled dataset with `se = FALSE`.

3 Machine learning examples

Suppose you want to relax parametric functional form assumptions by plugging in a machine learning estimator. As the user, this simply involves changing the arguments to the `gapclosing()` function.

3.1 Generalized Additive Models

Perhaps you are concerned about linearity assumptions: the continuous confounder, for instance, might actually have a nonlinear association with the outcome. We know the truth is linear in this simulated example, but in practice you would never know. You can address this concern by estimating with a GAM, using the `s()` operator from `mgcv` for smooth terms (see Wood (2017)).

```
estimate_gam <- gapclosing(
  data = simulated_data,
  counterfactual_assignments = 1,
  outcome_formula = formula(outcome ~ s(confounder) + category*treatment),
  treatment_formula = formula(treatment ~ s(confounder) + category),
  category_name = "category",
  treatment_algorithm = "gam",
  outcome_algorithm = "gam",
  sample_split = "cross_fit"
  # Note: Standard errors with `se = TRUE` are supported.
  # They are omitted here only to speed vignette build time.
)
```

3.2 Random forests

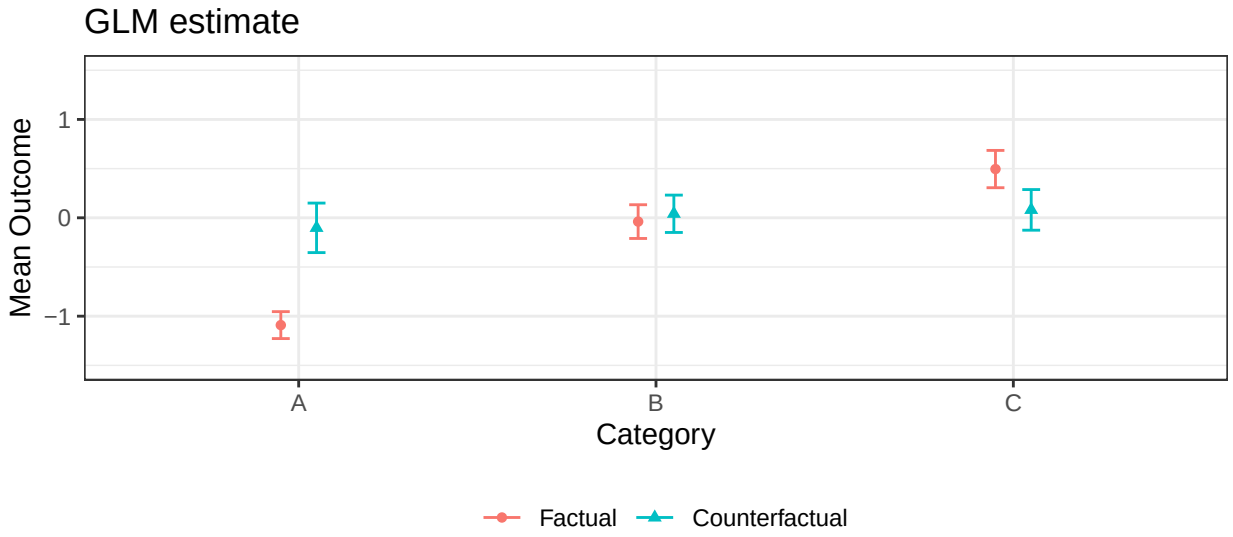
Perhaps you are concerned that the true treatment probability and expected outcome functions have many interactions among the predictors. You can set `treatment_algorithm` and `outcome_algorithm` to "ranger" to estimate via the `ranger` function in the `ranger` package (Wright and Ziegler, 2017).

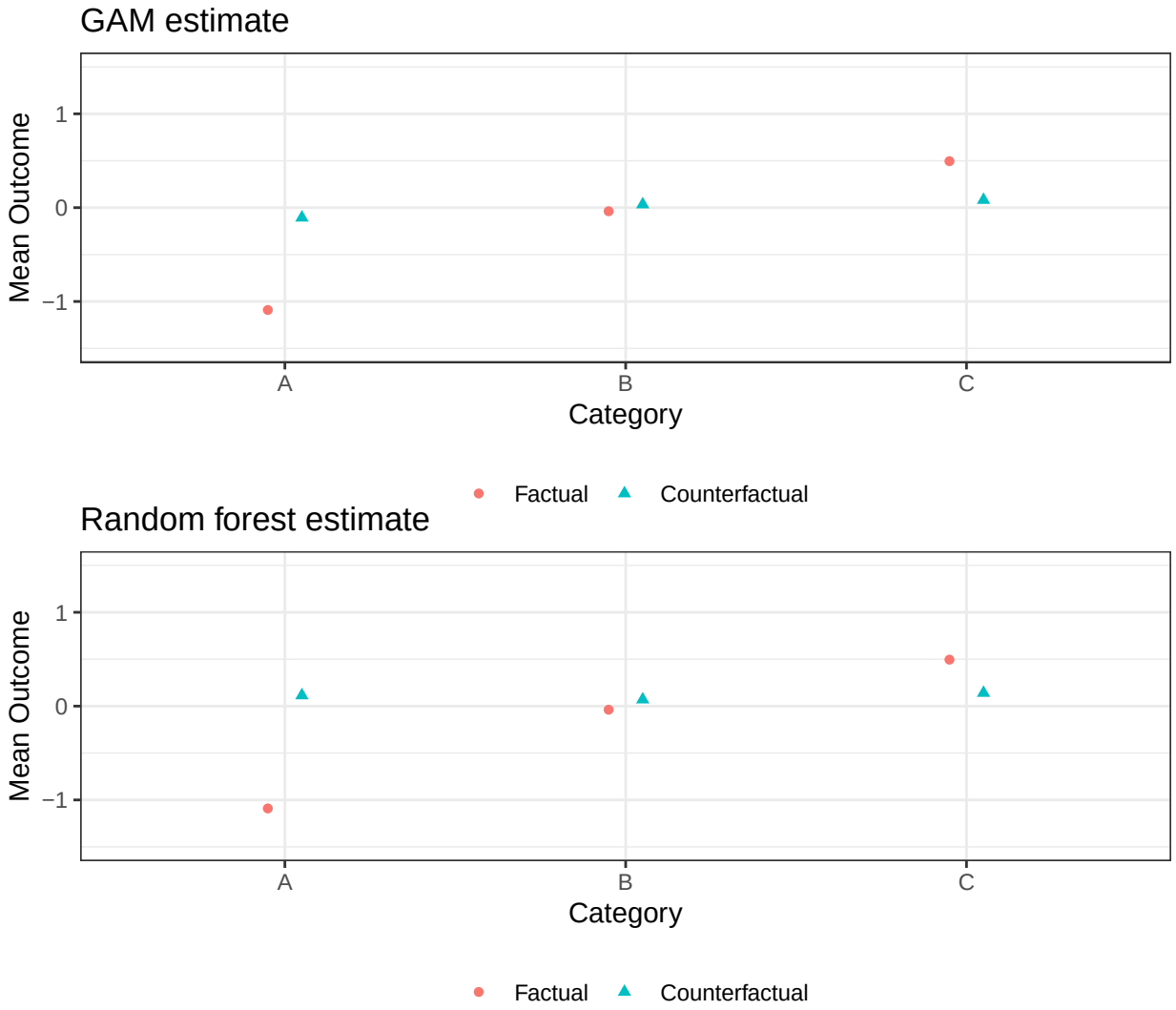
One aspect of the way `gapclosing()` operationalizes `ranger()` is unique out of all the estimation algorithm options. When you choose a random forest, it is because you believe there are many important interactions. Some of the most important interactions may be between the treatment and the other predictors. Therefore, `outcome_algorithm = ranger` enforces those interactions by estimating the outcome model separately for treated and control units. For this reason, when `outcome_algorithm = ranger` there is no need to include the treatment variable explicitly in the `outcome_formula`.

```
estimate_ranger <- gapclosing(
  data = simulated_data,
  counterfactual_assignments = 1,
  outcome_formula = formula(outcome ~ confounder + category),
  treatment_formula = formula(treatment ~ confounder + category),
  category_name = "category",
  treatment_algorithm = "ranger",
  outcome_algorithm = "ranger",
  sample_split = "cross_fit"
  # Note: Standard errors with `se = TRUE` are supported.
  # They are omitted here only to speed vignette build time.
)
```

3.3 Estimates from these three algorithms are roughly the same

In this simulation, the GLM models are correctly specified and there are no nonlinearities or interactions for the machine learning approaches to learn. In this case, the sample size is large enough that those approaches correctly learn the linear functional form, and all three estimation strategies yield similar estimates.





Note that confidence intervals for GAM and random forest can also be generated with `SE = TRUE`, which is turned off here only to speed vignette build time.

3.4 Word of warning

The assumptions of a parametric model are always doubtful, leading to a common question of whether one should always use a more flexible machine learning approach like `ranger`. In a very large sample, a flexible learner would likely be the correct choice. In the sample sizes of social science settings, the amount of data may sometimes be insufficient for these algorithms to discover a complex functional form. When the parametric assumptions are approximately true, the parametric estimators may have better performance in small sample sizes. What counts as "small" and "large" is difficult to say outside of any specific setting.

4 Advanced treatment assignments

So far, we have focused on estimation for a fixed treatment assignment: assign to treatment 1 with probability 1.

We might also want to know about the gap-closing estimand

- if we assigned people to treatment stochastically
- if each person's assignment were individualized

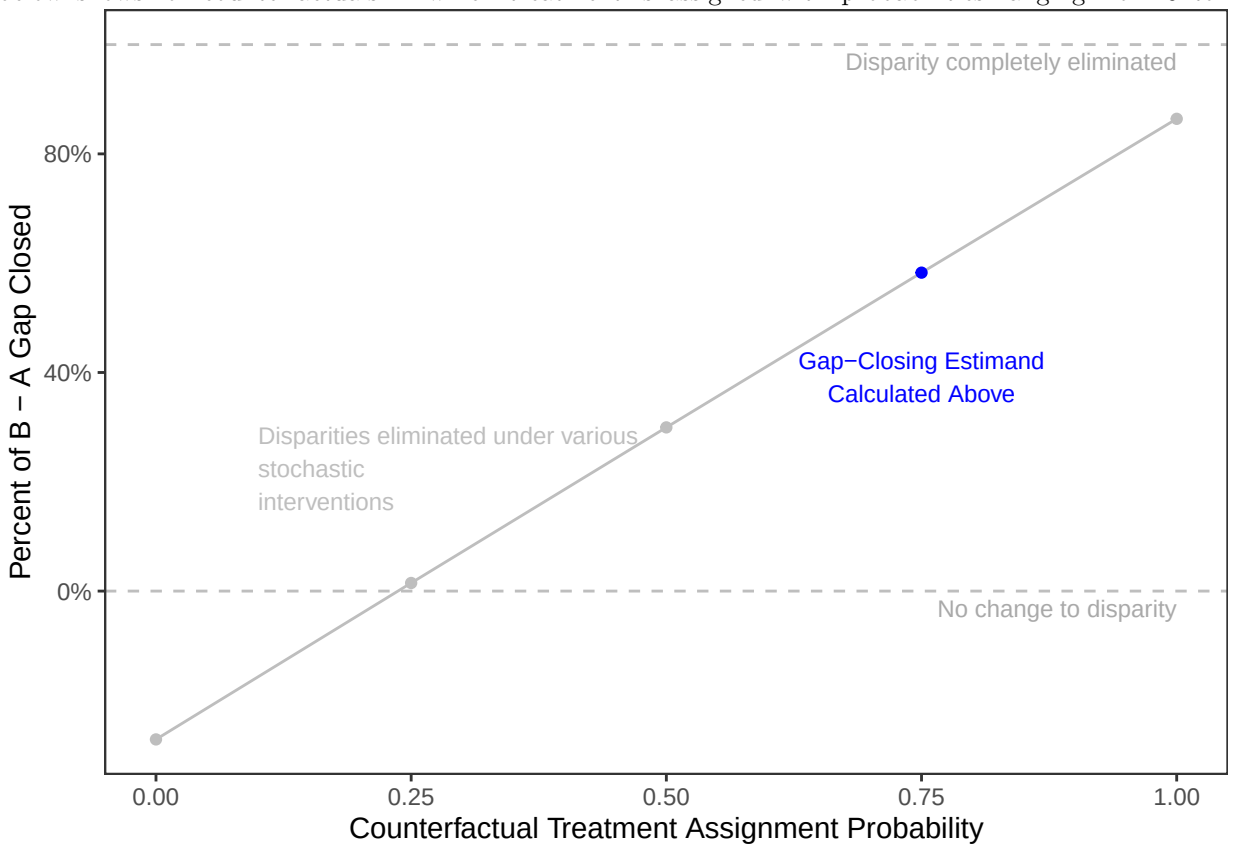
4.1 Stochastic treatments

We may want to study a counterfactual where treatment is assigned with some probability between 0 and 1. The `counterfactual_assignments` argument can handle this possibility.

For example, consider the gap-closing estimand if assigned to treatment 1 with each probability .75.

```
estimate_stochastic <- gapclosing(
  data = simulated_data,
  counterfactual_assignments = .75,
  outcome_formula = formula(outcome ~ confounder + category*treatment),
  treatment_formula = formula(treatment ~ confounder + category),
  category_name = "category"
)
```

The disparity between categories A and B under that stochastic intervention (0.75 probability of treatment = 1) is estimated to be 0.44, whereas under the previous deterministic intervention to assign treatment to the value 1 the disparity would be 0.14. This illustrates an important point: the gap-closing estimand can be different depending on the counterfactual assignment rule, as the figure below shows for counterfactuals in which treatment is assigned with probabilities ranging from 0 to 1.



4.2 Individualized treatments

Treatment may also be different (and possibly stochastic) for each unit in the counterfactual world of interest.

For example, suppose we assign those in Category A to treatment 1 with probability .5, those in Category B to treatment with probability .4, and those in Category C to treatment with probability .3. In this case, `counterfactual_assignments` will be set to a vector of length `nrow(data)`.

```
our_assignments <- case_when(simulated_data$category == "A" ~ .5,
                             simulated_data$category == "B" ~ .4,
                             simulated_data$category == "C" ~ .3)
estimate_stochastic <- gapclosing(
  data = simulated_data,
  counterfactual_assignments = our_assignments,
  outcome_formula = formula(outcome ~ confounder + category*treatment),
  treatment_formula = formula(treatment ~ confounder + category),
  category_name = "category"
)
```

That intervention would close the B - A gap by 31%.

5 Conclusion

The `gapclosing` package is designed to support inquiry into gap closing estimands, thus promoting new understanding about interventions that can close gaps across social categories. The goal of the package is to automate technical tasks (sample splitting, aggregation to doubly robust estimates, visualization), thus freeing the researcher to devote more attention to scientific tasks like defining the intervention and making causal assumptions.

If you use this package and find a bug, it would be most helpful if you would create an issue on GitHub. Suggestions for additional features are also welcome.

This vignette was compiled on 2023-03-08 23:03:09.

5.1 Computing environment

```
sessionInfo()
#> R version 4.2.2 (2022-10-31)
#> Platform: aarch64-apple-darwin20 (64-bit)
#> Running under: macOS Ventura 13.2.1
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
#>
#> locale:
#> [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets
#> [6] methods   base
#>
#> other attached packages:
#> [1] ggplot2_3.4.0    dplyr_1.1.0      gapclosing_1.0.2
#>
#> loaded via a namespace (and not attached):
#>  [1] Rcpp_1.0.10       highr_0.10        pillar_1.8.1
#>  [4] compiler_4.2.2    forcats_1.0.0     iterators_1.0.14
#>  [7] tools_4.2.2       digest_0.6.31     lattice_0.20-45
#> [10] nlme_3.1-162      evaluate_0.20     lifecycle_1.0.3
#> [13] tibble_3.1.8      gtable_0.3.1      mgcv_1.8-41
#> [16] pkgconfig_2.0.3   rlang_1.0.6       Matrix_1.5-3
```

```
#> [19] foreach_1.5.2     cli_3.6.0          rstudioapi_0.14
#> [22] yaml_2.3.7        parallel_4.2.2     xfun_0.37
#> [25] fastmap_1.1.0     ranger_0.14.1      withr_2.5.0
#> [28] knitr_1.42        generics_0.1.3     vctrs_0.5.2
#> [31] grid_4.2.2        tidyselect_1.2.0   glue_1.6.2
#> [34] R6_2.5.1          fansi_1.0.4        rmarkdown_2.20
#> [37] bookdown_0.33     farver_2.1.1       tidyr_1.3.0
#> [40] purrr_1.0.1       magrittr_2.0.3     splines_4.2.2
#> [43] scales_1.2.1      codetools_0.2-19   htmltools_0.5.4
#> [46] colorspace_2.1-0  labeling_0.4.2     utf8_1.2.3
#> [49] munsell_0.5.0     doParallel_1.0.17
```

References

Bang, H. and Robins, J. M. (2005). Doubly robust estimation in missing data and causal inference models. *Biometrics*, 61(4):962–973.

Bickel, P. J. (1982). On adaptive estimation. *The Annals of Statistics*, pages 647–671.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., and Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68.

Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22.

Glynn, A. N. and Quinn, K. M. (2010). An introduction to the augmented inverse propensity weighted estimator. *Political Analysis*, pages 36–56.

Hahn, J. (1998). On the role of the propensity score in efficient semiparametric estimation of average treatment effects. *Econometrica*, pages 315–331.

Hernán, M. A. and Robins, J. M. (2021). *Causal Inference: What if*. Chapman & Hall/CRC.

Jackson, J. W. (2020). Meaningful causal decompositions in health equity research: definition, identification, and estimation through a weighting framework. *Epidemiology*, 32(2):282–290.

Jackson, J. W. and VanderWeele, T. J. (2018). Decomposition analysis to identify intervention targets for reducing disparities. *Epidemiology*, 29(6):825.

Lundberg, I. (2022). The gap-closing estimand: A causal approach to study interventions that close disparities across social categories. *Sociological Methods and Research*.

Robins, J. M. and Rotnitzky, A. (1995). Semiparametric efficiency in multivariate regression models with missing data. *Journal of the American Statistical Association*, 90(429):122–129.

Van der Laan, M. J. and Rose, S. (2011). *Targeted learning: Causal inference for observational and experimental data*. Springer Science & Business Media.

VanderWeele, T. J. and Robinson, W. R. (2014). On causal interpretation of race in regressions adjusting for confounding and mediating variables. *Epidemiology*, 25(4):473.

Wood, S. N. (2017). *Generalized additive models: An introduction with R*. Chapman and Hall / CRC.

Wright, M. N. and Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in c++ and r. *Journal of Statistical Software*, 77(i01).