Package: whatifbandit (via r-universe)

July 23, 2026

Title Analyzing Randomized Experiments Using Multi-Arm Bandits

Version 1.0.2

Description Simulates response-adaptive experimental trials using Multi-Arm Bandits. Adaptive robust estimators defined in Hadad et al. (2021) <doi:10.1073/pnas.2014602118> and Offer-Westort et al. (2021) <doi:10.1111/ajps.12597> are used to robustly estimate conditional expectations and treatment effects. Provides significant simulation customization options for imperfect information, non-stationary bandits, and increased exploration strategies for assignments.

License GPL (>= 3)

URL https://github.com/Noch05/whatifbandit

BugReports https://github.com/Noch05/whatifbandit/issues

Depends R (>= 4.1.0)

Imports bandit, clubSandwich, data.table (>= 1.18.0), dplyr, estimatr, furrr, lubridate, methods, purrr, randomizr, rlang, tibble, tidyr

Suggests future, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

LazyData true

RoxygenNote 8.0.0

NeedsCompilation no

Author Noah Ochital [aut, cre, cph] (ORCID: <https://orcid.org/0009-0009-0398-2963>), Ryan T. Moore [ctb, cph] (ORCID: <https://orcid.org/0000-0002-3916-8113>)

Maintainer Noah Ochital <no9857a@american.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-07-23 10:50:02 UTC

RemoteUrl https://github.com/cran/whatifbandit

RemoteRef HEAD

RemoteSha a1ea820c6ed4c75c701d1a54ed85ae7c17166dac

Contents

`joint_test` *Joint Hypothesis Test for Multi-Arm Bandit Trials*

Description

Conducts a joint hypothesis test of no treatment effects across all arms, i.e. that all arms have the same true probability of success, either using a bootstrap procedure or the randomization inference procedure adapted from Offer-Westort et al. (2021). See details for a description of both methods

Usage

```
joint_test(mab, method = c("bootstrap", "randomization"), r = 100)
```

Arguments

`mab`	A `single_rct_mab` or `single_param_mab` object.
`method`	A character string; either `"bootstrap"` or `"randomization"`.
`r`	A positive integer; number of simulations used to build the null distribution. Default is 100.

Value

A named list object containing

- `f_statistic`: The observed F-statistic from the IPW regression.
- `null_distribution`: A numeric vector of F-statistics under the null.
- `p_value`: The proportion of simulated F-statistics more extreme than observed.
- `method`: The method used.
- `r`: Number of replications used.
- `effective_r`: Number of generated f_stats which are not `NULL` or `NA`.

NOTE

This procedure is experimental and has no Type I error guarantee. Offer-Westort et. al (2021) also note the test suffers from low power, but it is provided for experimentation nonetheless.

`method = "randomization"` operates under the sharp null that each unit would express the same outcome no matter the treatment they were assigned. To achieve this the trial is re-simulated but new outcomes are not generated or imputed, however the adaptive algorithm still changes the assignments. This results in a null distribution that captures how the adaptive algorithm will assign even when the outcomes are not related to treatments at all.

`method = "bootstrap"` operates under the null hypothesis that there is no difference between treatment arms within each block/cluster. If there are no blocks or clusters, a p-matrix is built from the pooled sample mean of the original trial. With block and or cluster pooled sample means are estimated within each block or cluster. The block and or cluster assignment proportions are taken from the original dataset.

References

Offer-Westort, Molly, Alexander Coppock, and Donald P. Green. "Adaptive Experimental Design: Prospects and Applications in Political Science." American Journal of Political Science 65, no. 4 (2021): 826–44. doi:10.1111/ajps.12597.

Examples

```
data(tanf)
set.seed(5)
adaptive <- mab_from_rct(success ~ condition, data = tanf, algorithm = "thompson",
period_method = "batch", period_length = 500)

# Low `r` for examples, use replications in practice
joint_test(adaptive, "randomization", r = 2)
joint_test(adaptive, "bootstrap", r = 2)
```

`mab_from_rct` *Simulate a Multi-Arm-Bandit Trial from an Existing Randomized Controlled Trial, With Bernoulli Distributed Outcomes.*

Description

Simulates a response-adaptive, Multi-Arm-Bandit (MAB) trial using experimental data from an original randomized controlled trial (RCT), and adaptive inference strategies described in Hadad et al. (2021) and Offer-Westort et al. (2021) to robustly estimate treatment effects. See the details to learn more.

Usage

```
mab_from_rct(
  formula,
  data,
  algorithm = c("thompson", "ucb1", "static"),
  random_assign_prop = 0,
  control_augment = 0,
  control_condition = NULL,
  period_method = c("batch", "date", "individual"),
  time_unit = NULL,
  period_length,
  prior_periods = NULL,
  discount_rate = 1,
  date_col = NULL,
  month_col = NULL,
  delayed_feedback = FALSE,
  assignment_date_col = NULL,
  success_date_col = NULL,
  whole_experiment = FALSE,
  ndraws = 5000,
  r = 1,
  verbose = FALSE,
  check_args = TRUE,
  keep_data = FALSE,
  keep_models = FALSE,
  contrasts = NULL,
  ...
)
```

Arguments

`formula`	A `formula` object specifying outcome variable, treatment indicator, treatment blocking and treatment clustering for optional blocked and/or clustered randomized designs. The treatment variable should always be the first variable following ~ (Additional covariates to be added in later updates). Clustering and blocking variables should be included in specific `block()` or `cluster()` groups in the formula, e.g. `outcome ~ treatment + block(x1, x2, x3) + cluster(x4)`.
`data`	A `data.frame`, `data.table`, or any object which inherits from `data.frame`, containing input data from the trial. This should be the results of a traditional Randomized Controlled Trial (RCT).
`algorithm`	A character string specifying the MAB algorithm to use. Options are `"thompson"`, `"ucb1"`, or "static", ignoring case. Algorithm defines the adaptive assignment process. For more details on these specific algorithms see Thompson 1933; Auer et al. 2002; Agrawal and Goyal 2012; Kuleshov and Precup 2014 and Slivkins 2024.
`random_assign_prop`	Proportion of each treatment wave assigned via static, equal probabilities of assignment. Adaptive probabilities are updated by `p * (1 - random_assign_prop)`

+ random_assign_prop * 1/k, where k is the number of treatment arms.

control_augment
: Minimum proportion of each treatment assignment wave guaranteed to receive the treatment labeled as "Control". Ranges from 0 to 1, and the default is 0. Adjustment is always made after the adjustment from random_assign_prop.

control_condition
: Value of the control condition. Only necessary when control_augment is greater than 0. Internally this value is coerced to a string, so it should be passed as a string, or a type that can easily be converted to a string.

period_method
: A character string; one of "date", "batch", or "individual", to define the assignment into treatment waves. When using "batch" or "individual", ensure your dataset is pre-arranged in the proper order observations should be considered so that groups are assigned correctly. For "date", observations will be considered in chronological order. "individual" assignment can be computationally intensive for larger datasets.

time_unit
: A character string specifying the unit of time for assigning periods when period_method ="date". Acceptable values are "day", "week", or "month". "month" does not require an additional column with the months of each observation, but it can accept a separate month_col. If month_col is specified, the periods follow the calendar months strictly, and when it is not specified months are simply used as the time interval. For example if a dataset has dates starting on July 26th, under month based assignment and a specified month_col the dates July 26th and August 3rd would be in different periods, but if the month_col was not specified, they would be in the same period because the dates are less than one month apart.

period_length
: A positive integer; represents the length of each treatment period. If period_method is "date", this length refers the number of units specified in time_unit. (i.e., if "day", 10 would be 10 days). If period_method = "batch", this refers to the number of units in each batch.

prior_periods
: A positive integer; number of previous periods to use in the treatment assignment model. Default is NULL, where all prior periods are considered. See below for details.

discount_rate
: Rate for discounting observations from earlier periods when updating assignment probabilities. A value between 0 and 1, where outcomes from k periods ago are weighted by discount_rate^k. Default is 1 for no discounting.

date_col
: Bare column in data; contains original date of event/trial. Only necessary when assigning by "Date". Must be of type Date.

month_col
: Bare column in data; contains month of treatment. Only necessary when time_unit = "month", and when periods should be determined directly by the calendar months instead of month based time periods. This column can be a string/factor variable with the month names or numeric with the month number. It can easily be created from your date_col via lubridate::month(data[[date_col]]) or format(data[[date_col]], "%m").

delayed_feedback
: Logical; if FALSE, assumes instantaneous feedback for outcomes, as soon as a treatment is assigned, the outcome is realized and known. If TRUE, delayed

feedback is assumed, so as soon as treatment is assigned, a potential outcome is realized, but it is not known to the simulation, until a certain date. When recomputing the adaptive assignment probabilities, outcomes that have not been observed on the date of assignment are treated as failures.

`assignment_date_col`
: Bare column in `data`; contains original dates treatments were assigned to observations. Only necessary when `delayed_feedback = TRUE`. Used to simulate imperfect observation of outcomes in the simulation. Must be of type `Date`, not a character string.

`success_date_col`
: Bare column in `data`; contains original dates each success occurred. Only necessary when `delayed_feedback = TRUE`. Must be of type `Date`, not a character string.

`whole_experiment`
: Logical; if `TRUE`, uses all past experimental data for imputing outcomes. If `FALSE`, uses only data available up to the current period. In large datasets or with a high number of periods, setting this to `FALSE` can be more computationally intensive, though not a significant contributor to total run time. Default is `FALSE`.

`ndraws`
: Number of draws used to approximate Thompson sampling probabilities. Used only when direct calculation fails or overflows. Default is 5000 but can be raised or lowered depending on performance and accuracy concerns.

`r`
: Positive integer; number of replications (under different random seed). Replications of the MAB procedure on a fixed dataset provides important diagnostic information on the stochasticity/variance of the re-simulation method. Replications can be conducted in parallel, by setting an appropriate `future::plan()`. See details below.

`verbose`
: Logical; whether or not to print intermediate messages. Default is `FALSE`.

`check_args`
: Logical; whether or not to validate passed arguments. Default is `TRUE` and recommended not to be changed.

`keep_data`
: Logical; Whether or not to keep the final data from each trial. Recommended `FALSE`. When `r = 1` the final data is always kept and reported.

`keep_models`
: Logical; Whether or not to keep the final IPW and OLS models from each trial. Recommended `FALSE`. When `r = 1` models are always kept and reported. Required to be `TRUE` to compute arbitrary pairwise contrasts. Only utilized under clustering.

`contrasts`
: Character string specifying which pairwise contrasts to precompute after each replication. One of `"control"` (each arm vs. control arm), `"best"` (each arm vs. the MAB-selected best arm), `"both"`, or `"all"` (all `choose(k, 2)` pairwise comparisons, expensive for large `k`). All contrasts are tested under the two-sided null of no difference. Defaults to `NULL`, arbitrary contrasts can be computed after if `keep_models == TRUE`

`...`
: Additional named arguments passed to `furrr::furrr_options()`

Details

Clustering:

Clusters should be contained inside each assignment wave (a warning is thrown if this is not the case), so it is possible to have 2 observations in the same cluster assigned to different treatments if they were assigned in different waves. This is assumed because without it the adaptive probabilities will not be impacting assignments. For example if someone in cluster 1 is assigned in period 1, then all other members are forced to have the same treatment, even if they are assigned in period 5, 10, 20 etc.

Implementation:

At each period, either the Thompson sampling probabilities or UCB1 values are calculated based on the outcomes from the number of `prior_periods` specified weighted by `discount_rate`. New treatments are then assigned randomly using the Thompson sampling probabilities via the randomizr package, or as the treatment with the highest UCB1 values, while implementing the specific treatment blocking and control augmentation specified.

After assigning treatments, observations with new treatments have their outcomes imputed using success rates from the original randomized trial. These rates are estimated as grouped means within each treatment arm. If blocking is specified, rates are estimated within each combination of treatment arm and block.

If `delayed_feedback = TRUE`, new dates of success will be imputed using the means of those dates in the period, grouped by treatment block if necessary. Observations for which their treatment changed, but their outcome was success in the original and simulation, do not have their date changed. When the next period starts, the success dates are checked against the maximum/latest `assignment_date` for the period, and if any success occurs after that, it is treated as a failure for the purpose of the bandit decision algorithms.

Inference:

At the end of the simulation the results are aggregated together to calculate the Adaptively Weighted Augmented Inverse Probability Estimator (Hadad et al. 2021) using the mean and variance formulas provided, under the constant allocation rate adaptive schema. These estimators are unbiased and asymptotically normal under the adaptive conditions and their differences are also unbiased asymptotically normal estimators for treatment effects. See Hadad et al. (2021). Asymptotic validity hinges on sub-optimal arms continually being assigned, if arms have low to 0 probability of being assigned, the central limit theorem proved no longer applies. Thus it is recommended to use `random_assign_prop` and `control_augment` to ensure all arms have non-zero probabilities of assignment over the whole trial.

Under clustering the unit of observation becomes the cluster, the sample size becomes the number of clusters. Individual estimates are aggregated in each period by cluster before being used to compute the final AIPW estimate and variance (CR0 style). The variance is adjusted by the Stata CR1 adjustment, $(\frac{G}{G-1} * \frac{N-1}{N-k})$ where k is the number of treatments, and G is the number of clusters. Degrees of freedom of `G-1` are also provided, for use of the more conservative t-distribution, though inference is still only valid asymptotically.

Inverse Probability Weighted (IPW) estimates are also provided using `estimatr::lm_robust()`. Offer-Westort et al. (2021). In clustered cases CR2 standard errors are used, and CR1 (Stata) used if CR2 computation fails. HC2 standard errors are used in non-clustered cases. In high sample sizes for the arms chosen, standard t-tests of the estimates and their contrasts can be asymptotically valid. F-statistics are provided for joint tests provided in `joint_test()`.

AIPW and IPW are unbiased, with AIPW having lower variances generally, while standard unweighted OLS estimates will be biased with spuriously low variance, but are provided for comparisons.

Performance Concerns:

This procedure has the potential to be computationally expensive and time-consuming. Performance depends on the relative size of each period, number of periods, the overall size of the dataset, and number of replications. This function has separate support for `data.frames` and `data.tables`. If a `data.frame` is passed, the function uses a combination of `dplyr`, `tidyr` and base `R` to shape data, and run the simulation. However, if a `data.table` is passed the function exclusively uses the `data.table` code for all the same operations.

In general, smaller batches run faster under base `R`, while larger ones could benefit from the performance and memory efficiencies provided by `data.table`. However, we've observed larger datasets can cause numerical instability with some calculations in the Thompson sampling procedure. Internal safeguards exist to prevent this, but the best way to preempt any issues is to set `prior_periods` to a low number.

`r > 1`:

Multiple simulations allows researchers to gauge the variance of the simulation procedure itself, by repeating it several times under different random states, using the same fixed data.

Parallel Processing:

The function provides support for parallel processing via the future and furrr packages. When conducting a large number of simulations, parallelization can improve performance if sufficient system resources are available. Parallel processing must be explicitly set by the user, through `future::plan()`. Windows users should set the plan to "multisession", while Linux and MacOS users can use "multicore" or "multisession". Users running in a High Performance Computing environment (HPC), are encouraged to use future.batchtools, for their respective HPC scheduler. Note that parallel processing is not guaranteed to work on all systems, and may require additional setup or debugging effort from the user. For any issues, users are encouraged to consult the documentation of the above packages.

Value

Depends on `r` value if `r = 1`, an S3 `single_rct_mab` class object, and if `r > 1`, an S3 `muti_rct_mab`, with the following:

- `new_data`: `tibble` or `data.table` containing the new treatment assignments and outcomes under the simulation. If `r >1` and `keep_data = TRUE`, the tables from each trial are nested inside.
- `bandits`: A list with 3 elements:
  - `statistic`: Thompson Sampling or UCB1 statistics computed for each treatment at each period of each trial.
  - `assignment_prob`: Assignment probabilities for each treatment at each period of each trial.
  - `assignment_quant`: Assignment quantities for each treatment in each trial.
- `means`: A `tibble` or `data.table` containing point estimates, and standard errors for the AIPW, IPW, and OLS estimators for each treatment in each trial.

- `f_stats`: A named numeric vector of F statistics from IPW and OLS regressions. When `r` > 1, it is a data.frame/data.table of F statistics with columns corresponding to IPW and OLS.
- `contrasts`: A `tibble` or `data.table` containing point estimates, and standard errors for the estimated linear contrasts of treatment arm estimates for each trial. Only when `contrasts` is not `NULL`.
- `models`: List containing `lm_robust` objects from IPW and OLS regressions, only stored when `keep_models = TRUE` or `r = 1` and clusters are provided.
- `config`: Configuration list of 3 elements:
  - `args`: List of arguments passed to `simulate_mab()`.
  - `call`: The original call to `simulate_mab()`.
  - `parallel`: The `furrr::furrr_options()` object used for parallelization.

References

Agrawal, Shipra, and Navin Goyal. 2012. "Analysis of Thompson Sampling for the Multi-Armed Bandit Problem." *Proceedings of the 25th Annual Conference on Learning Theory*, June 16, 39.1-39.26. https://proceedings.mlr.press/v23/agrawal12.html.

Asyuraa, F. C., S. Abdullah, and T. E. Sutanto. 2021. "Empirical Evaluation on Discounted Thompson Sampling for Multi-Armed Bandit Problem with Piecewise-Stationary Bernoulli Arms." Journal of Physics: Conference Series 1722 (1): 012096. doi:10.1088/17426596/1722/1/012096

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. "Finite-Time Analysis of the Multiarmed Bandit Problem." *Machine Learning* 47 (2): 235–56. doi:10.1023/A:1013689704352.

Bengtsson, Henrik. 2025. "Future: Unified Parallel and Distributed Processing in R for Everyone." https://cran.r-project.org/package=future.

Bengtsson, Henrik. 2025. "Future.Batchtools: A Future API for Parallel and Distributed Processing Using 'Batchtools.'" https://cran.r-project.org/package=future.batchtools.

Garivier, Aurélien, and Eric Moulines. 2008. "On Upper-Confidence Bound Policies for Non-Stationary Bandit Problems." arXiv:0805.3415. Preprint, arXiv, May 22. doi:10.48550/arXiv.0805.3415

Hadad, Vitor, David A. Hirshberg, Ruohan Zhan, Stefan Wager, and Susan Athey. 2021. "Confidence Intervals for Policy Evaluation in Adaptive Experiments." *Proceedings of the National Academy of Sciences of the United States of America* 118 (15): e2014602118. doi:10.1073/pnas.2014602118.

Kuleshov, Volodymyr, and Doina Precup. 2014. "Algorithms for Multi-Armed Bandit Problems." *arXiv*. doi:10.48550/arXiv.1402.6028.

Loecher, Thomas Lotze and Markus. 2022. "Bandit: Functions for Simple a/B Split Test and Multi-Armed Bandit Analysis." https://cran.r-project.org/package=bandit.

Offer-Westort, Molly, Alexander Coppock, and Donald P. Green. 2021. "Adaptive Experimental Design: Prospects and Applications in Political Science." *American Journal of Political Science* 65 (4): 826–44. doi:10.1111/ajps.12597.

Slivkins, Aleksandrs. 2024. "Introduction to Multi-Armed Bandits." *arXiv*. doi:10.48550/arXiv.1904.07272.

Vaughan, Davis, Matt Dancho, and RStudio. 2022. "Furrr: Apply Mapping Functions in Parallel Using Futures." https://cran.r-project.org/package=furrr.

See Also

furrr, future, `joint_test()`, `simulate_mab()`

Examples

```
data(tanf)
set.seed(454)

mab_from_rct(success ~ condition, data = tanf, algorithm = "thompson",
period_method = "batch", period_length = 500, delayed_feedback = TRUE,
assignment_date_col = appt_date, success_date_col = date_of_recert)

mab_from_rct(success ~ condition, data = tanf, algorithm = "ucb1",
period_method = "date", time_unit = "day", date_col = appt_date,
 period_length = 60, r = 2, discount_rate = 0.8)

mab_from_rct(success ~ condition + block(service_center), data = tanf, algorithm = "thompson",
period_method = "batch", period_length = 500, control_condition = "no_letter",
control_augment = 0.2, prior_periods = 1)
```

`simulate_mab` *Simulate an Adaptive Trial With Bernoulli Distributed Outcomes*

Description

Simulates a response-adaptive randomized experiment with Bernoulli distributed outcomes. At each period, observed outcomes are used to update assignment probabilities according to the specified `algorithm`. `algorithm = "static"` is the non-adaptive uniform baseline, where probabilities of being assigned to one treatment is the same as any other.

Usage

```
simulate_mab(
  n,
  t = n,
  p,
  algorithm = c("thompson", "ucb1"),
  blocks = NULL,
  clusters = NULL,
  control_augment = 0,
  random_assign_prop = 0,
  delayed_feedback = FALSE,
  assignment_dates = NULL,
  time_model = NULL,
  period_sizes = NULL,
  prior_periods = NULL,
  discount_rate = 1,
  dt = FALSE,
  ndraws = 5000,
  r = 1,
  keep_data = FALSE,
```

```
  keep_models = FALSE,
  contrasts = NULL,
  check_args = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

n	A positive integer. Total number of units to simulate.
t	Total number of assignment periods. Positive integer. Default is `t = n` for pure sequential (one unit per period) assignment. The sizes of each period will be equal as `n %/% t`, except for the last period which will be `n %/% t + n %% t`, when `period_sizes = NULL`.
p	The true probabilities of success for each treatment arm. Specified as a matrix, where `rownames(p)` are the treatment arm names. If there is a control condition, specify its rowname as `"Control"`. `colnames(p)` are the cluster or block labels, e.g. `matrix(c(0.5, 0.3, 0.5, 0.6), nrow = 2, ncol = 2, dimnames(list(c("Control", "T1"), c("B1", "B2"))))`. Probabilities are accessed as `p[treatment, block]`. With blocks and clusters utilize the clusters for the columns because clusters are fully nested in blocks. For no clusters or blocks simply use a matrix with 1 column.
algorithm	A character string specifying the MAB algorithm to use. Options are `"thompson"`, `"ucb1"`, or "static", ignoring case. Algorithm defines the adaptive assignment process. For more details on these specific algorithms see Thompson 1933; Auer et al. 2002; Agrawal and Goyal 2012; Kuleshov and Precup 2014 and Slivkins 2024.
blocks	A named numeric vector of block membership probabilities (must sum to 1), where `names(blocks)` are the block labels. Units are assigned to blocks via `randomizr::complete_ra()`. Pass `NULL` (default) for no blocking.
clusters	Cluster membership probabilities. Can be: **Numeric vector** A named vector where `names(clusters)` are the cluster labels e. g. `c(C1 = 0.4, C2 = 0.6)`. Used when there is not blocking. **Named list of vectors** A named list where `names(clusters)` are block labels, and each element is a named vector of per-block cluster proportions, e.g. `list(B1 = c(C1 = 0.4, C2=0.6), B2 = c(C3 = 0.2, C4 = 0.8))` Clusters are accessed as `clusters[[block]][cluster]`. Inside each block, cluster proportions must sum to 1, and the same cluster cannot appear in multiple blocks. Units are assigned to clusters via `randomizr::complete_ra()`. Pass `NULL` (default) for no clustering.
control_augment	Minimum proportion of each treatment assignment wave guaranteed to receive the treatment labeled as `"Control"`. Ranges from 0 to 1, and the default is 0. Adjustment is always made after the adjustment from random_assign_prop.

random_assign_prop
: Proportion of each treatment wave assigned via static, equal probabilities of assignment. Adaptive probabilities are updated by `p * (1 - random_assign_prop) + random_assign_prop * 1/k`, where k is the number of treatment arms.

delayed_feedback
: Logical; if `FALSE`, assumes instantaneous feedback for outcomes, as soon as a treatment is assigned, the outcome is realized and known. If `TRUE`, delayed feedback is assumed, so as soon as treatment is assigned, a potential outcome is realized, but it is not known to the simulation, until a certain date. When recomputing the adaptive assignment probabilities, outcomes that have not been observed on the date of assignment are treated as failures.

assignment_dates
: An optional `Date` vector of dates representing when units are assigned. If shorter than `n` it is recycled and sorted. If `NULL` (default) no assignment dates are recorded.

time_model
: An optional function with signature:
`function(n, conditions, successes, current_period, blocks = NULL, clusters = NULL, ...)`
It returns a vector of lubridate::period objects which will then be added to `assignment_dates` to produce `success_date`. Used to simulate delayed feedback mechanism during the trial, so outcomes are imperfectly observed. Only used when`assignment_dates` is also supplied. Dates can be generated even when `delayed_feedback == FALSE`, but they will not be used. Default `NULL`. Other optional arguments Cannot share names with arguments in `furrr::furrr_options()`.

period_sizes
: Numeric vector of `length(t)`, with the specific number of units to be assigned in each period. Used when it is required to assign different numbers of units to treatment across the periods of the trial.

prior_periods
: A positive integer; number of previous periods to use in the treatment assignment model. Default is `NULL`, where all prior periods are considered. See below for details.

discount_rate
: Rate for discounting observations from earlier periods when updating assignment probabilities. A value between 0 and 1, where outcomes from k periods ago are weighted by `discount_rate^k`. Default is 1 for no discounting.

dt
: Logical. If `TRUE` returns a `data.table::data.table()`; otherwise returns a `tibble::tibble()`. Default `FALSE`.

ndraws
: Number of draws used to approximate Thompson sampling probabilities. Used only when direct calculation fails or overflows. Default is 5000 but can be raised or lowered depending on performance and accuracy concerns.

r
: Positive integer; number of replications (under different random seed). Replications of the MAB procedure on a fixed dataset provides important diagnostic information on the stochasticity/variance of the re-simulation method. Replications can be conducted in parallel, by setting an appropriate `future::plan()`. See details below.

keep_data
: Logical; Whether or not to keep the final data from each trial. Recommended `FALSE`. When `r = 1` the final data is always kept and reported.

keep_models	Logical; Whether or not to keep the final IPW and OLS models from each trial. Recommended FALSE. When r = 1 models are always kept and reported. Required to be TRUE to compute arbitrary pairwise contrasts. Only utilized under clustering.
contrasts	Character string specifying which pairwise contrasts to precompute after each replication. One of "control" (each arm vs. control arm), "best" (each arm vs. the MAB-selected best arm), "both", or "all" (all choose(k, 2) pairwise comparisons, expensive for large k). All contrasts are tested under the two-sided null of no difference. Defaults to NULL, arbitrary contrasts can be computed after if keep_models == TRUE
check_args	Logical; whether or not to validate passed arguments. Default is TRUE and recommended not to be changed.
verbose	Logical; whether or not to print intermediate messages. Default is FALSE.
...	Additional named arguments forwarded to time_model and furrr::furrr_options().

Details

Blocking and Clustering:

When blocking and/or clustering are specified, these assignments will be randomly pregenerated before the start of the adaptive sequential assignment. These arguments allow simulating a trial when there may be heterogeneous outcomes across a treatment block or treatment cluster, so different assignment probabilities can be provided for the same treatment, depending on the block and/or cluster of a unit.

Clusters should be contained inside each assignment wave (a warning is thrown if this is not the case), so it is possible to have 2 observations in the same cluster assigned to different treatments if they were assigned in different waves. This is assumed because without it the adaptive probabilities will not be impacting assignments. For example if someone in cluster 1 is assigned in period 1, then all other members are forced to have the same treatment, even if they are assigned in period 5, 10, 20 etc.

Implementation:

At each period, either the Thompson sampling probabilities or UCB1 values are calculated based on the outcomes from the number of prior_periods specified weighted by discount_rate. New treatments are then assigned randomly using the Thompson sampling probabilities via the randomizr package, or as the treatment with the highest UCB1 values, while implementing the specific treatment blocking and control augmentation specified.

After assigning treatments, observations will have their outcomes generated via a Bernoulli draw associated to the probability in the p matrix corresponding to their treatment and block/cluster. If delayed_feedback = TRUE, dates of success will be generated via the provided time_model() function. When the next period starts, the success dates are checked against the maximum/latest assignment_date for the period, and if any success occurs after that, it is treated as a failure for the purpose of the bandit decision algorithms.

Inference:

At the end of the simulation the results are aggregated together to calculate the Adaptively Weighted Augmented Inverse Probability Estimator (Hadad et al. 2021) using the mean and variance formulas provided, under the constant allocation rate adaptive schema. These estimators are unbiased

and asymptotically normal under the adaptive conditions and their differences are also unbiased asymptotically normal estimators for treatment effects. Hadad et al. (2021). Asymptotic validity, hinges on sub-optimal arms continually being assigned, if arms have low to 0 probability of being assigned, the central limit theorem proved no longer applies. Thus it is recommended to use `random_assign_prop` and `control_augment` to ensure all arms non-zero probabilities of assignment over the whole trial.

Under clustering the unit of observation becomes the cluster, the sample size the number of clusters. Individual estimates are aggregated in each period by cluster before being used to compute the final AW-AIPW estimate and variance (CR0 style). The variance is adjusted by the Stata CR1 adjustment, $(\frac{G}{G-1} * \frac{N-1}{N-k})$ where k is the number of treatments, and G is the number of clusters. Degrees of freedom of `G-1` are also provided, for use of the more conservative t-distribution, though inference is still only valid asymptotically.

Inverse Probability Weighted (IPW) estimates are also provided using `estimatr::lm_robust()`. Offer-Westort et al. (2021). In clustered cases CR2 standard errors are used, and CR1 (Stata) used if CR2 computation fails. HC2 standard errors are used in non-clustered cases. In high sample sizes for the arms chosen, standard t-tests of the estimates and their contrasts can be asymptotically valid. F-statistics are provided for joint tests provided in `joint_test()`.

AW-AIPW and IPW are unbiased, with AW-AIPW having lower variances generally, while standard unweighted OLS estimates will be biased with spuriously low variance but are provided for comparisons.

`r > 1`**:**

Multiple simulations allow researchers to gauge the variance of the procedure and produce bootstrap estimates of variance of the procedure under the passed parameters. For each simulation new data is drawn according to the passed population parameters. This differs from `mab_from_rct()` where resimulations occurs on the same fixed dataset.

Further details about the adaptive procedure can be found in `mab_from_rct()`

Performance Concerns:

This procedure has the potential to be computationally expensive and time-consuming. Performance depends on the relative size of each period, `t`, `n`, and `r`. This function has separate support for `data.frames` and `data.tables`, selected by the `dt` argument. This flag defines two separate tracks where either combination of `dplyr`, `tidyr` and base `R` to shape data, and run the simulation or only `data.table` code operations are used.

In general, smaller batches run faster under base `R`, while larger ones could benefit from the performance and memory efficiencies provided by `data.table`. However, we've observed larger sizes can cause numerical instability with some calculations in the Thompson sampling procedure. Internal safeguards exist to prevent this, but the best way to preempt any issues is to set `prior_periods` to a low number.

Parallel Processing:

The function provides support for parallel processing via the future and furrr packages. When conducting a large number of simulations, parallelization can improve performance if sufficient system resources are available. Parallel processing must be explicitly set by the user, through `future::plan()`. Windows users should set the plan to "multisession", while Linux and MacOS users can use "multicore" or "multisession". Users running in a High Performance Computing environment (HPC), are encouraged to use future.batchtools, for their respective HPC scheduler. Note that parallel processing is not guaranteed to work on all systems, and may require additional

setup or debugging effort from the user. For any issues, users are encouraged to consult the documentation of the above packages.

Value

Depends on `r` value if `r = 1`, an S3 `single_param_mab` class object, and if `r > 1`, an S3 `muti_param_mab`, with the following:

- `new_data`: `tibble` or `data.table` containing the new treatment assignments and outcomes under the simulation. If `r >1` and `keep_data = TRUE`, the tables from each trial are nested inside.
- `bandits`: A list with 3 elements:
  - `statistic`: Thompson Sampling or UCB1 statistics computed for each treatment at each period of each trial.
  - `assignment_prob`: Assignment probabilities for each treatment at each period of each trial.
  - `assignment_quant`: Assignment quantities for each treatment in each trial.
- `means`: A `tibble` or `data.table` containing point estimates, and standard errors for the AW-AIPW, IPW, and OLS estimators for each treatment in each trial.
- `f_stats`: A named numeric vector of F statistics from IPW and OLS regressions. When `r > 1`, it is a data.frame/data.table of F statistics with columns corresponding to IPW and OLS.
- `contrasts`: A `tibble` or `data.table` containing point estimates, and standard errors for the estimated linear contrasts of treatment arm estimates for each trial. Only when `contrasts` is not `NULL`.
- `models`: List containing `lm_robust` objects from IPW and OLS regressions, only stored when `keep_models = TRUE` or `r = 1` and clusters are provided.
- `config`: Configuration list of 3 elements:
  - `args`: List of arguments passed to `simulate_mab()`.
  - `call`: The original call to `simulate_mab()`.
  - `parallel`: The `furrr::furrr_options()` object used for parallelization.

References

Agrawal, Shipra, and Navin Goyal. 2012. "Analysis of Thompson Sampling for the Multi-Armed Bandit Problem." *Proceedings of the 25th Annual Conference on Learning Theory*, June 16, 39.1-39.26. https://proceedings.mlr.press/v23/agrawal12.html.

Asyuraa, F. C., S. Abdullah, and T. E. Sutanto. 2021. "Empirical Evaluation on Discounted Thompson Sampling for Multi-Armed Bandit Problem with Piecewise-Stationary Bernoulli Arms." Journal of Physics: Conference Series 1722 (1): 012096. doi:10.1088/17426596/1722/1/012096

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. "Finite-Time Analysis of the Multiarmed Bandit Problem." *Machine Learning* 47 (2): 235–56. doi:10.1023/A:1013689704352.

Bengtsson, Henrik. 2025. "Future: Unified Parallel and Distributed Processing in R for Everyone." https://cran.r-project.org/package=future.

Bengtsson, Henrik. 2025. "Future.Batchtools: A Future API for Parallel and Distributed Processing Using 'Batchtools.'" https://cran.r-project.org/package=future.batchtools.

Garivier, Aurélien, and Eric Moulines. 2008. "On Upper-Confidence Bound Policies for Non-Stationary Bandit Problems." arXiv:0805.3415. Preprint, arXiv, May 22. doi:10.48550/arXiv.0805.3415

Hadad, Vitor, David A. Hirshberg, Ruohan Zhan, Stefan Wager, and Susan Athey. 2021. "Confidence Intervals for Policy Evaluation in Adaptive Experiments." *Proceedings of the National Academy of Sciences of the United States of America* 118 (15): e2014602118. doi:10.1073/pnas.2014602118.

Kuleshov, Volodymyr, and Doina Precup. 2014. "Algorithms for Multi-Armed Bandit Problems." *arXiv*. doi:10.48550/arXiv.1402.6028.

Loecher, Thomas Lotze and Markus. 2022. "Bandit: Functions for Simple a/B Split Test and Multi-Armed Bandit Analysis." https://cran.r-project.org/package=bandit.

Offer-Westort, Molly, Alexander Coppock, and Donald P. Green. 2021. "Adaptive Experimental Design: Prospects and Applications in Political Science." *American Journal of Political Science* 65 (4): 826–44. doi:10.1111/ajps.12597.

Slivkins, Aleksandrs. 2024. "Introduction to Multi-Armed Bandits." *arXiv*. doi:10.48550/arXiv.1904.07272.

Vaughan, Davis, Matt Dancho, and RStudio. 2022. "Furrr: Apply Mapping Functions in Parallel Using Futures." https://cran.r-project.org/package=furrr.

See Also

mab_from_rct()

Examples

```
n <- 100
t <- 10

p <- matrix(c(0.2, 0.5, 0.45, 0.5), ncol =1, dimnames = list(paste0("T", 1:4)))

set.seed(543)
simulate_mab(n, t, p = p, random_assign_prop = 0.1,
algorithm =  "ucb1", discount_rate  = 0.5)

simulate_mab(n, t, p = p, random_assign_prop = 0.1,
period_sizes = c(37, rep(7, 9)), algorithm = "thompson")


simulate_mab(n, t, p = matrix(c(0.1, 0.5, 0.3, 0.2, 0.3, 0.3),
dimnames = list(c("T1", "T2"), c("B1", "B2", "B3")),
ncol = 3, nrow = 2), blocks = c("B1" =0.3, "B2" = 0.5, "B3" = 0.2),
algorithm = "thompson")
```

---

tanf *Public TANF Recipient Data From Washington D.C*

Description

A modified version of the data set used in https://thelabprojects.dc.gov/benefits-reminder-letter with one additional column added for analysis.

Usage

```
data(tanf)
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 3517 rows and 21 columns.

Details

Variables are as follows:

ic_case_id Unique, anonymized case identifier.

service_center DC Department of Human Services Center assigned each case.

condition The assigned letter condition: "No Letter", "Open Appointment", or "Specific Appointment".

recert_month Recertification Month.

letter_sent_date Date the second (treatment) letter was sent.

recert_id Administrative recertification identifier.

return_to_sender Indicates whether letter was returned as undeliverable

pdc_status PDC Status

renewal_date Date by which renewal must be completed.

notice_date.x Date the first notice was sent (initial legal communication)

days_betwn_notice_and_recert_due Number of days between the first notice and the recertification due date.

cert_period_start Start date of the recertification period.

cert_period_end End date of recertification period.

recert_status Status of recertification process (Pending, Denied, etc.)

denial_reason Reason for denial if recertification was not approved.

recert_month_year Combined recertification month and year.

notice_date.y Alternate record of first notice date.

recert_status_dcas Official recertification status from DCAS

date_of_recert Date the recertification was successfully submitted (if applicable).

success Binary variable indicating successful recertification based on recert_status (newly added column).

Source

https://github.com/thelabdc/DHS-TANFRecertification-Public/blob/main/data/df_replication_anonymized.csv

Index