Package 'whatifbandit'

July 14, 2025

Title Analyzing Randomized Experiments as Multi-Arm Bandits

Version 0.0.0.9000

Description Simulates the results of completed randomized controlled trials, as if they had been conducted as adaptive Multi-Arm Bandit (MAB) trials instead. Augmented inverse probability weighted estimation (AIPW) is used to robustly estimate the probability of success for each treatment arm under the adaptive conditions. Provides customization options to simulate perfect/imperfect information, stationary/non-stationary bandits, treatment blocking, and control augmentation strategy for assigning treatment arms.

License GPL (>= 3)

Depends R (>= 4.1.0)

Imports bandit, data.table, dplyr, furrr, lubridate, purrr, randomizr, rlang, tibble, tidyr

Suggests future, ggplot2, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Contents

`multiple_mab_simulation` *Conducts Multiple Multi-Arm Bandit Trials with Adaptive Inference in Parallel*

Description

Repeated Multi-Arm Bandit Simulations with the same settings in different random states. Allows for parallel processing using `future::plan()` and `furrr::future_map()`.

Usage

```
multiple_mab_simulation(
  data,
  assignment_method,
  algorithm,
  conditions,
  prior_periods,
  perfect_assignment,
  whole_experiment,
  blocking,
  data_cols,
  times,
  seeds,
  control_augment = 0,
  time_unit = NULL,
  period_length = NULL,
  block_cols = NULL,
  verbose = FALSE,
  keep_data = FALSE
)
```

Arguments

`data`	A data.frame, data.table, or tibble containing input data from the trial. This should be the results of a traditional Randomized Controlled Trial (RCT). data.frames will be converted to tibbles internally.
`assignment_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 time-consuming for larger datasets.

`algorithm` A character string specifying the MAB algorithm to use. Options are "Thompson" or "UCB1".

`conditions` A named character vector containing treatment conditions. The elements of this vector should be the names of each treatment as seen in your data, so to create it you can simply call `unique(df$cond)`. The names of each element are used to reference the contents but are not inherently important; choose names that are meaningful and consistent. If `control_augment` > 0, then the control condition of the trial in this vector must have the name "Control".

`prior_periods` A numeric value of length 1, or the character string "All"; number of previous periods to use in the treatment assignment model. This is used to implement the stationary/non-stationary bandit. For example, a non-stationary bandit assumes the true probability of success for each treatment changes over time, so to account for that, not all prior data should be used when making decisions because it could be "out of date".

`perfect_assignment`
A logical value; if TRUE, assumes perfect information for treatment assignment (i.e., all outcomes are observed regardless of the date). If FALSE, hides outcomes not yet theoretically observed, based on the dates treatments would have been assigned for each wave. This is useful when simulating batch-based assignment where treatments were assigned on a given day whether or not all the information from a prior batch was available and you have exact dates treatments were assigned.

`whole_experiment`
A logical value; 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 runtime.

`blocking` A logical value; whether or not to use treatment blocking. Treatment blocking is used to ensure an even-enough distribution of treatment conditions across blocks. For example, blocking by gender would mean the randomized assignment should split treatments evenly not just throughout the sample (so for 4 arms, 25-25-25-25), but also within each block, so 25% of men would receive each treatment and 25% of women the same.

`data_cols` A named character vector containing the names of columns in `data` as strings:

- `id_col`: Column in `data`; contains unique ID as a key.
- `success_col`: Column in `data`; binary successes from the original experiment.
- `condition_col`: Column in `data`; original treatment condition for each observation.
- `date_col`: Column in `data`; contains original date of event/trial. Only necessary when assigning by "Date". Must be of type `Date`, not a character string.
- `month_col`: Column in `data`; contains month of treatment. Only necessary when `time_unit = "Month"`. This can be a string or factor variable containing the names or numbers of months.
- `success_date_col`: Column in `data`; contains original dates each success occurred. Only necessary when `perfect_assignment = FALSE`. Must be of type `Date`, not a character string.
- `assignment_date_col`: Column in `data`; contains original dates treatments were assigned to observations. Only necessary when `perfect_assignment`

= FALSE. Used to simulate imperfect information on the part of researchers conducting an adaptive trial. Must be of type Date, not a character string.

times	Integer; number of simulations to conduct.
seeds	Integer vector of length(times) containing valid seeds to define random state for each trial.
control_augment	A numeric value in 0, 1; proportion of each wave guaranteed to receive the "Control" treatment. Default is 0.
time_unit	A character string specifying the unit of time for assigning periods when assignment_method is "Date". Acceptable values are "Day", "Week", or "Month".
period_length	A numeric value of length 1; represents the length of each treatment period. If assignment method is "Date", this refers to the length of periods by your specified time_unit (i.e., if "Day", 10 would be 10 days). If assignment method is "Batch", this refers to the number of people in each batch. This factor contributes most to the computational cost of calling the function, as large batch sizes make each iteration of the simulation run slower, while each additional period adds time because of the extra iterations. If you have a large dataset, consider passing it as a data.table.
block_cols	A character vector of variables to block by. This vector should not be named.
verbose	Logical; Toggles progress bar from furrr::future_map().
keep_data	Logical; Whether or not to keep the final data from each trial. Recommended FALSE for large datasets .

Value

multiple.mab class object, which is a named list containing:

- final_data_nest: Data.frame containing a nested data.frame with the final data from each trial
- bandits: Data.frame containing the Thompson/UCB1 statistics across all treatments, periods, and trials
- estimates: Data.frame containing the AIPW statistics across all treatments, and trials
- settings: A list of the configuration settings used in the trial.

References

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. https://doi.org/10.1073/pnas.2014602118.

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

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. https://doi.org/10.1111/ajps.12597.

See Also

- run_mab_trial()
- get_adaptive_aipw()

- check_args()
- single_mab_simulation()
- mab_simulation()
- pre_mab_simulation()
- furrr::future_map()
- future::plan()

Examples

```
# Multiple_mab_simulation() is a useful tool for running multiple trials
# using the same configuration settings, in different random states
data(tanf)
# Subsetting to make the example faster
tanf <- tanf[1:50, ]

# The seeds passed must be integers, so it is highly recommended to create them
# before using `sample.int()`
set.seed(1)
seeds <- sample.int(10000, 5)
conditions <- c("no_letter", "open_appt", "specific_appt")

# For this example, period_length is set a large interval and
# times is low to keep run time short.
start <- proc.time()
x <- multiple_mab_simulation(
  data = tanf,
  assignment_method = "Batch",
  period_length = 25,
  whole_experiment = TRUE,
  blocking = FALSE,
  perfect_assignment = TRUE,
  algorithm = "Thompson",
  prior_periods = "All",
  control_augment = 0,
  conditions = conditions,
  data_cols = c(
    condition_col = "condition",
    id_col = "id",
    success_col = "success"
  ),
  verbose = FALSE, times = 5, seeds = seeds, keep_data = TRUE
)
seq_time <- proc.time() - start
print(x)

# Its Recommenced to set keep_data at FALSE unless necessary to avoid
# the output from taking up to much memory
# Keep TRUE
object.size(x)
x$final_data_nest <- NULL
# Size if Keep was FALSE
object.size(x)

# multiple_mab_simulation() is implemented using furrr::future_map()
# so you can also run simulations in parallel using futures.
```

```
# Simply run your preferred plan and number of cores before multiple_mab_simulation.
# Like:
## Not run:

  future::plan("plan", workers = n)
  multiple_mab_simulation(data = tanf,
                          assignment_method = "Batch",
                          period_length = 25,
                          whole_experiment = TRUE,
                          blocking = FALSE,
                          perfect_assignment = TRUE,
                          algorithm = "Thompson",
                          prior_periods = "All",
                          control_augment = 0,
                          conditions = conditions,
                          data_cols = c(
                            condition_col = "condition",
                            id_col = "id",
                            success_col = "success"
                          ),
                          verbose = FALSE, times = 5, seeds = seeds, keep_data = TRUE
  )
  future::plan("sequential")

## End(Not run)
# If your on Windows plan needs to be multisession
# If your on Unix (MacOS/Linux) you can use multicore or multisession
# If your running the code on a high performance cluster, look into
# using the future.batchtools API for whichever scheduler is used

# Check the future and furrr documentation for more details on possible options
```

`plot.multiple.mab` *Plot Generic for* `multiple.mab` *objects*

Description

Uses `ggplot2::ggplot()` to summarize the results of multiple Multi-Arm Bandit Trials

Usage

```
## S3 method for class 'multiple.mab'
plot(
  x,
  type,
  estimator = NULL,
  cdf = NULL,
  level = 0.95,
  save = FALSE,
  path = NULL,
  ...
)
```

Arguments

x	multiple.mab class object created by multiple_mab_simulation()
type	String; Type of plot requested; valid types are: • summary: Shows the number of times each arm was selected as the highest chance of being the best. • hist: Shows histograms for each treatment condition's proportion of success across trials. • estimate: Shows proportion of success estimates user specified normal or empirical confidence intervals.
estimator	Estimator to plot; Either "AIPW", "Sample" or "Both"; used by hist and estimate.
cdf	String; specifies the type of CDF to use when analyzing the estimates. valid cdfs are the empirical cdf, the normal cdf. Used when type = estimate.
level	Confidence Interval Width (i.e 0.90, .95, 0.99)
save	Logical; Whether or not to save the plot to disk; FALSE by default.
path	String; File directory to save file.
...	arguments to pass to ggplot2:geom_* function (e.g. color, linewidth, alpha, bins etc.)

Value

Minimal ggplot object, that can be customized and added to with + (To change, scales, labels, legend, theme, etc.)

Examples

```
# Objects returned by `single_mab_simulation()` have a `mab` class.
# This class has a plot generic has several minimal plots to examine the trials
# quickly
#
# # These functions require ggplot2
if (requireNamespace("ggplot2", quietly = TRUE)) {
  data(tanf)
  # Subsetting to make the example faster
  tanf <- tanf[1:20, ]
  # Simulating a few trials

  seeds <- sample.int(100, 5)
  conditions <- as.character(unique(tanf$condition))
  x <- multiple_mab_simulation(
    data = tanf,
    assignment_method = "Batch",
    period_length = 10,
    whole_experiment = TRUE,
    blocking = FALSE,
    perfect_assignment = TRUE,
    algorithm = "Thompson",
    prior_periods = "All",
    control_augment = 0,
    conditions = conditions,
    data_cols = c(
      condition_col = "condition",
      id_col = "id",
```

```
    success_col = "success"
  ),
  verbose = FALSE, times = 5, seeds = seeds, keep_data = FALSE
)

# The plot generic has several options
# Specify type = summary, to get a bar graph showing each time
# a treatment group was selected as the best.
plot(x, type = "summary")

# type = hist, creates a histogram of AIPW, Sample, or Both estimates for each
# treatment over each trial
plot(x, type = "hist", estimator = "AIPW")

# type = estimate creates a similar error bar plot like in plot.mab()
# but here the empirical variance of the estimate can be used instead
plot(x, type = "estimate", estimator = "AIPW", cdf = "empirical")

# These plots can be added to like any ggplot2 object
plot(x, type = "summary") + ggplot2::labs(title = "Your New Title")

# Each only uses 1 geom, so arguments for them can be added in the function call
plot(x, type = "hist", estimator = "AIPW", binwidth = 0.05)
}
```

`print.mab` *Print Generic For* `mab`

Description

Custom Print Display for objects of mab class returned by `single_mab_simulation()`.

Usage

```
## S3 method for class 'mab'
print(x, ...)
```

Arguments

`x`	mab class object created by `single_mab_simulation()`
`...`	further arguments passed to or from other methods

Value

Text summary of settings used for the Multi-Arm Bandit trial.

print.multiple.mab *Print Generic For* multiple.mab

Description

Custom Print Display for 'multiple.mab" objects returned by multiple_mab_simulation().

Usage

```
## S3 method for class 'multiple.mab'
print(x, ...)
```

Arguments

x	multiple.mab class object
...	further arguments passed to or from other methods

Value

Text summary of settings used for the Multi-Arm Bandit trials.

single_mab_simulation *Run One Adaptive Simulation With Inference.*

Description

Performs a single Multi-Arm Bandit (MAB) trial using experimental data from an original randomized controlled trial, and adaptive inference strategies as described in Hadad et al. (2021). Wraps around the internal implementation functions, and performs the full MAB pipeline: preparing inputs, assigning treatments and imputing successes, and adaptively weighted estimation.

Usage

```
single_mab_simulation(
  data,
  assignment_method,
  algorithm,
  conditions,
  prior_periods,
  perfect_assignment,
  whole_experiment,
  blocking,
  data_cols,
  control_augment = 0,
  time_unit = NULL,
  period_length = NULL,
  block_cols = NULL,
  verbose = FALSE
)
```

Arguments

`data`	A data.frame, data.table, or tibble containing input data from the trial. This should be the results of a traditional Randomized Controlled Trial (RCT). data.frames will be converted to tibbles internally.
`assignment_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 time-consuming for larger datasets.
`algorithm`	A character string specifying the MAB algorithm to use. Options are "Thompson" or "UCB1".
`conditions`	A named character vector containing treatment conditions. The elements of this vector should be the names of each treatment as seen in your data, so to create it you can simply call `unique(df$cond)`. The names of each element are used to reference the contents but are not inherently important; choose names that are meaningful and consistent. If `control_augment` > 0, then the control condition of the trial in this vector must have the name "Control".
`prior_periods`	A numeric value of length 1, or the character string "All"; number of previous periods to use in the treatment assignment model. This is used to implement the stationary/non-stationary bandit. For example, a non-stationary bandit assumes the true probability of success for each treatment changes over time, so to account for that, not all prior data should be used when making decisions because it could be "out of date".
`perfect_assignment`	A logical value; if TRUE, assumes perfect information for treatment assignment (i.e., all outcomes are observed regardless of the date). If FALSE, hides outcomes not yet theoretically observed, based on the dates treatments would have been assigned for each wave. This is useful when simulating batch-based assignment where treatments were assigned on a given day whether or not all the information from a prior batch was available and you have exact dates treatments were assigned.
`whole_experiment`	A logical value; 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 runtime.
`blocking`	A logical value; whether or not to use treatment blocking. Treatment blocking is used to ensure an even-enough distribution of treatment conditions across blocks. For example, blocking by gender would mean the randomized assignment should split treatments evenly not just throughout the sample (so for 4 arms, 25-25-25-25), but also within each block, so 25% of men would receive each treatment and 25% of women the same.
`data_cols`	A named character vector containing the names of columns in `data` as strings: • `id_col`: Column in `data`; contains unique ID as a key. • `success_col`: Column in `data`; binary successes from the original experiment. • `condition_col`: Column in `data`; original treatment condition for each observation.

- `date_col`: Column in `data`; contains original date of event/trial. Only necessary when assigning by "Date". Must be of type `Date`, not a character string.
- `month_col`: Column in `data`; contains month of treatment. Only necessary when `time_unit = "Month"`. This can be a string or factor variable containing the names or numbers of months.
- `success_date_col`: Column in `data`; contains original dates each success occurred. Only necessary when `perfect_assignment = FALSE`. Must be of type `Date`, not a character string.
- `assignment_date_col`: Column in `data`; contains original dates treatments were assigned to observations. Only necessary when `perfect_assignment = FALSE`. Used to simulate imperfect information on the part of researchers conducting an adaptive trial. Must be of type `Date`, not a character string.

`control_augment`
: A numeric value in 0, 1; proportion of each wave guaranteed to receive the "Control" treatment. Default is 0.

`time_unit`
: A character string specifying the unit of time for assigning periods when `assignment_method` is "Date". Acceptable values are "Day", "Week", or "Month".

`period_length`
: A numeric value of length 1; represents the length of each treatment period. If assignment method is "Date", this refers to the length of periods by your specified `time_unit` (i.e., if "Day", 10 would be 10 days). If assignment method is "Batch", this refers to the number of people in each batch. This factor contributes most to the computational cost of calling the function, as large batch sizes make each iteration of the simulation run slower, while each additional period adds time because of the extra iterations. If you have a large dataset, consider passing it as a `data.table`.

`block_cols`
: A character vector of variables to block by. This vector should not be named.

`verbose`
: A logical value; whether or not to print intermediate messages. Default is FALSE.

Details

This function simulates a single adaptive Multi-Arm-Bandit trial, using experimental data from a traditional randomized controlled trial. It is intended to help researchers understand how an adaptive design could have performed but it is not a substitute for a real experiment, and for that reason it does not generate the synthetic data for the simulation. The input data should come from a randomized trial to ensure the assumptions made during the simulation are valid.

At each period, the outcomes from the number of previous periods specified in `prior_periods` are aggregated together to calculate Thompson probabilities of each arm being the best using the bandit package, or UCB1 statistic based on the well defined formula. New Treatments are then assigned randomly using the Thompson Probabilities via the randomizr package, implementing any specified treatment blocking, or as the highest UCB1 statistic.

If `perfect_assignment` is FALSE, at this step, some of the successes may be masked, if they occurred after the specified treatment assignment date for that given period, but these will be unmasked in later periods.

After treatments are assigned, observations with new treatments have their outcomes imputed, once again using the randomizr package, with any specified treatment blocking implemented. The probabilities of success used to impute, are estimated via the grouped means of successes from the original data, either from the whole trial, or up to that period, defined by `whole_experiment`. If `perfect_assignment` is FALSE, only for those who switched treatments and changed to success

have their success date imputed using a grouped average from the treatment block in the period. However if an observation changed treatment, but had succeeded under the original experiment, they did not have their date changed.

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.

This procedure has the potential to be computationally expensive and time-consuming. Performance depends on the relative size of each period, number of periods, and overall size of the dataset. This function supports `data.table` objects, which are used when passed, but otherwise a combination of `dplyr` and base R is 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`. An example dataset with 3,520 observations under individual assignment takes 20-30 seconds under Base R and 40-50 seconds under `data.table`

Value

An object of class `mab`, which is a named list containing:

- `final_data`: The processed tibble or data.table, containing new columns pertaining to the results of the trial.
- `bandits`: A tibble or data.table containing the UCB1 statistics or Thompson Sampling posterior distributions for each period.
- `assignment_probs`: A tibble or data.table containing the probability of being assigned each treatment arm at a given period.
- `estimates`: A tibble or data.table containing the AIPW (Augmented Inverse Probability Weighting) treatment effect estimates and variances, and traditional sample means and variances, for each treatment arm.
- `settings`: A named list of the configuration settings used in the trial.
- `original_data`: The original `data` object passed to the function (data.frame, tibble, or data.table).

References

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. https://doi.org/10.1073/pnas.2014602118.

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

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. https://doi.org/10.1111/ajps.12597.

See Also

`multiple_mab_simulation()`, `summary.mab()`, `plot.mab()`.

Examples

```
# Loading Example Data and defining conditions
data(tanf)
conditions <- c("no_letter", "open_appt", "specific_appt")

## Running Thompson Sampling with 500 person large batches,
## with no blocks and imperfect assignment

single_mab_simulation(
  data = tanf,
  assignment_method = "Batch",
  algorithm = "Thompson",
  period_length = 500,
  prior_periods = "All",
  blocking = FALSE,
  whole_experiment = TRUE,
  conditions = conditions,
  perfect_assignment = FALSE,
  data_cols = c(
    condition_col = "condition",
    id_col = "id",
    success_col = "success",
    success_date_col = "date_of_recert",
    assignment_date_col = "letter_sent_date"
  )
)

## Running UCB1 Sampling with 1 Month based batches and
## control augmentation set to 0.25, with perfect_assignment.
## When using control_augment > 0, conditions need to have proper names
names(conditions) <- c("Control", "T1", "T2")
# no_letter is control, the others are treatments

single_mab_simulation(
  data = tanf,
  assignment_method = "Date",
  time_unit = "Month",
  algorithm = "UCB1",
  period_length = 1,
  prior_periods = "All",
  blocking = FALSE,
  whole_experiment = TRUE,
  perfect_assignment = TRUE,
  conditions = conditions,
  control_augment = 0.25,
  data_cols = c(
    condition_col = "condition",
    id_col = "id",
    success_col = "success",
    date_col = "appt_date",
    month_col = "recert_month"
  )
)

## If you misspecify or miss an argument, an appropriate error will be given
## I specified Month assignment but did not provide a month_column in my data
```

```
try(single_mab_simulation(
  data = tanf,
  assignment_method = "Date",
  time_unit = "Month",
  algorithm = "UCB1",
  period_length = 1,
  prior_periods = "All",
  blocking = FALSE,
  whole_experiment = TRUE,
  perfect_assignment = FALSE,
  conditions = conditions,
  data_cols = c(
    condition_col = "condition",
    id_col = "id",
    success_col = "success",
    date_col = "appt_date"
  )
))

# I specified a negative period_length

try(single_mab_simulation(
  data = tanf,
  assignment_method = "Date",
  time_unit = "Month",
  algorithm = "UCB1",
  period_length = -500,
  prior_periods = "All",
  blocking = FALSE,
  whole_experiment = TRUE,
  perfect_assignment = FALSE,
  conditions = conditions,
  control_augment = 0,
  data_cols = c(
    condition_col = "condition",
    id_col = "id",
    success_col = "success",
    date_col = "appt_date"
  )
))

# I forgot to add column containing the successes of the original experiment

try(single_mab_simulation(
  data = tanf,
  assignment_method = "Batch",
  algorithm = "Thompson",
  period_length = 500,
  prior_periods = "All",
  blocking = FALSE,
  whole_experiment = TRUE,
  perfect_assignment = TRUE,
  conditions = conditions,
  control_augment = 0,
  data_cols = c(
    condition_col = "condition",
```

```
    id_col = "service_center"
  )
))
```

---

summary.mab *Summary Generic for "mab" class*

Description

Summarizes the Results of a Single Multi-Arm Bandit Trial.

Usage

```
## S3 method for class 'mab'
summary(object, level = 0.95, ...)
```

Arguments

object	"mab" class object created by single_mab_simulation().
level	Confidence Interval Width (i.e 0.90, .95, 0.99)
...	additional arguments.

Value

data.frame containg each treatment, the final Thompson/UCB1 Statistic, the AIPW estimate and Normal CI based on user supplied level.

Examples

```
# Objects returned by `single_mab_simulation()` have a `mab` class.
# This class has a summary generic that can produce quick results of the trial.

# Loading Data and running a quick simulation
data(tanf)
x <- single_mab_simulation(
  data = tanf,
  algorithm = "Thompson",
  assignment_method = "Batch",
  period_length = 600,
  whole_experiment = TRUE,
  perfect_assignment = TRUE,
  blocking = FALSE,
  prior_periods = "All",
  conditions = c(
    "no_letter",
    "open_appt",
    "specific_appt"
  ),
  data_cols = c(
    condition_col = "condition",
    id_col = "id",
    success_col = "success"
  )
```

```
)

# Calling `summary` Returns a summary table for the trial
# Defaults to 95% Normal Confidence Intervals
# for the Augmented Inverse Probability Estimates
summary(x)

# We can also change the confidence level to anything between 0 and 1
summary(x, level = 0.7)

# Invalid levels throw an error
try(summary(x, level = 5))
```

`summary.multiple.mab` *Summary Generic for "multiple.mab" class*

Description

Summarizes results of of multiple Multi-Arm Bandit Trials

Usage

```
## S3 method for class 'multiple.mab'
summary(object, level = 0.95, ...)
```

Arguments

`object`	`multiple.mab` object created by multiple_mab_simulation
`level`	Confidence Interval Width (i.e 0.90, .95, 0.99)
`...`	additional arguments.

Examples

```
# Objects returned by `multiple_mab_simulation()` have a `multiple.mab` class.
# This class has a summary generic that can produce quick results of the trials
data(tanf)
# Subsetting to make the example faster
tanf <- tanf[1:100, ]
# Simulating a few trials

seeds <- sample.int(10000, 5)
conditions <- c("no_letter", "open_appt", "specific_appt")
x <- multiple_mab_simulation(
  data = tanf,
  assignment_method = "Batch",
  period_length = 20,
  whole_experiment = TRUE,
  blocking = FALSE,
  perfect_assignment = TRUE,
  algorithm = "Thompson",
  prior_periods = "All",
  control_augment = 0,
  conditions = conditions,
```

```
  data_cols = c(
    condition_col = "condition",
    id_col = "id",
    success_col = "success"
  ),
  verbose = FALSE, times = 5, seeds = seeds, keep_data = FALSE
)

# Calling `summary` Returns a summary table for the trial
# Upper and Lower Bounds default to 95% Confidence Intervals
summary(x) |>
  print(width = Inf) # calling width = Inf to so whole table prints

# We can also change the confidence level to anything between 0 and 1
# This only changes the upper and lower bounds that are presented.
summary(x, level = 0.7) |>
  dplyr::select(lower_normal:upper_empirical)

# Invalid levels throw an error
try(summary(x, level = 5))
```

`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 3520 rows and 22 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