Package ‘grizbayr’

December 23, 2022

Type Package
Title Bayesian Inference for A/B and Bandit Marketing Tests
Version 1.3.3
Author Ryan Angi
Maintainer Ryan Angi <angi.ryan@gmail.com>
Description Uses simple Bayesian conjugate prior update rules to calculate the win probability of each option, value remaining in the test, and percent lift over the baseline for various marketing objectives.
References:
Depends R (>= 2.10)
License MIT + file LICENSE
Encoding UTF-8
RoxygenNote 7.2.1
Imports purrr, dplyr, tidyr (>= 1.0.0), magrittr, tibble, rlang
Suggests spelling, knitr, testthat (>= 2.1.0), rmarkdown
Language en-US
VignetteBuilder knitr
NeedsCompilation no
Repository CRAN
Date/Publication 2022-12-23 11:10:06 UTC

R topics documented:
calculate_multi_rev_per_session ............................... 2
calculate_total_cm ................................. 3
distribution_column_mapping ............................... 4
**Description**

Calculate Multi Rev Per Session
Usage

    calculate_multi_rev_per_session(conv_rates, inverse_rev_A, inverse_rev_B)

Arguments

    conv_rates      Dirichlet samples containing a tibble with columns alpha_1, alpha_2, and alpha_0
    inverse_rev_A   Vector of inverse revenue samples from A conversion type
    inverse_rev_B   Vector of inverse revenue samples from B conversion type

Value

    Vector of samples (dbl)

---

calculate_total_cm  Calculate Total CM

description

Calculate Total CM

Usage

    calculate_total_cm(rev_per_click, cost_per_click, expected_clicks)

Arguments

    rev_per_click   vector of rev per click samples
    cost_per_click  vector of cost per click (cpc) samples
    expected_clicks vector of expected clicks (expected CTR * fixed impressions)

Value

    vector of CM estimates (dbl)
distribution_column_mapping

A Mapping from distribution names (inputs) to columns. This allows the package to dynamically select column names for different distribution types.

Description

A Mapping from distribution names (inputs) to columns. This allows the package to dynamically select column names for different distribution types.

Usage

distribution_column_mapping

Format

A data frame with 11 rows and 12 columns

estimate_all_values Estimate All Values

Description

Efficiently estimates all values at once so the posterior only need to be sampled one time. This function will return as a list win probability, value remaining, estimated percent lift with respect to the provided option, and the win probability of the best option vs the provided option.

Usage

```r
estimate_all_values(
  input_df,  # Required
  distribution,  # Required
  wrt_option_lift,  # Required
  priors = list(),  # Optional
  wrt_option_vr = NULL,  # Optional
  loss_threshold = 0.95,  # Optional
  lift_threshold = 0.7,  # Optional
  metric = "lift"  # Optional
)
```
Arguments

input_df  Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.

distribution  String of the distribution name

wrt_option_lift  String: the option lift and win probability is calculated with respect to (wrt). Required.

priors  Optional list of priors. Defaults will be use otherwise.

wrt_option_vr  String: the option against which loss (value remaining) is calculated. If NULL the best option will be used. (optional)

loss_threshold  The confidence interval specifying what the "worst case scenario" should be. Defaults to 95%. (optional)

lift_threshold  The confidence interval specifying how likely the lift is to be true. Defaults to 70%. (optional)

metric  string the type of loss. absolute will be the difference, on the outcome scale. 0 when best = wrt_option lift will be the (best - wrt_option) / wrt_option, 0 when best = wrt_option relative_risk will be the ratio best/wrt_option, 1 when best = wrt_option

Details

TODO: Add high density credible intervals to this output for each option.

Value

A list with 4 named items: Win Probability, Value Remaining, Lift vs Baseline, and Win Probability vs Baseline.

Examples

```r
## Not run:
input_df <- data.frame(option_name = c("A", "B", "C"),
                         sum_clicks = c(1000, 1000, 1000),
                         sum_conversions = c(100, 120, 110), stringsAsFactors = FALSE)
estimate_all_values(input_df, distribution = "conversion_rate", wrt_option_lift = "A")
## End(Not run)
```
**estimate_lift**

*Estimate Lift Distribution*

**Description**

Estimates lift distribution vector from posterior samples.

**Usage**

```r
estimate_lift(posterior_samples, distribution, wrt_option, metric = "lift")
```

**Arguments**

- `posterior_samples`: Tibble returned from sample_from_posterior with 3 columns 'option_name', 'samples', and 'sample_id'.
- `distribution`: String of the distribution name
- `wrt_option`: string the option lift is calculated with respect to (wrt). Required.
- `metric`: string the type of lift. 'absolute' will be the difference, on the outcome scale. 0 when best = wrt_option 'lift' will be the (best - wrt_option) / wrt_option, 0 when best = wrt_option 'relative_risk' will be the ratio best/wrt_option, 1 when best = wrt_option

**Value**

numeric, the lift distribution

**Examples**

```r
# Requires posterior_samples dataframe. See `sample_from_posterior()`
# for an example.

## Not run:
estimate_lift(posterior_samples = posterior_samples,
distribution = "conversion_rate",
wrt_option = "A",
metric = "lift")

## End(Not run)
```
Estimate Lift vs Baseline

Description
Estimate Lift vs Baseline

Usage

```r
estimate_lift_vs_baseline(
  input_df,
  distribution,
  priors = list(),
  wrt_option,
  metric = "lift",
  threshold = 0.7
)
```

Arguments

- `input_df`: Dataframe containing `option_name` (str) and various other columns depending on the distribution type. See vignette for more details.
- `distribution`: String of the distribution name.
- `priors`: Optional list of priors. Defaults will be used otherwise.
- `wrt_option`: String the option loss is calculated with respect to (wrt). Required.
- `metric`: String the type of loss. `absolute` will be the difference, on the outcome scale. 0 when `best = wrt_option` `lift` will be the `(best - wrt_option) / wrt_option`, 0 when `best = wrt_option` `relative_risk` will be the ratio `best/wrt_option`, 1 when `best = wrt_option`.
- `threshold`: Lift percentage threshold between 0 and 1. (0.7 threshold is "at least 70% lift"). Defaults to 0.7.

Value

numeric value remaining at the specified threshold

Examples

```r
input_df <- tibble::tibble(option_name = c("A", "B", "C"),
  sum_clicks = c(1000, 1000, 1000),
  sum_conversions = c(100, 120, 110))
estimate_lift_vs_baseline(input_df, distribution = "conversion_rate", wrt_option = "A")
```
estimate_loss  Estimate Loss

Description
Estimate Loss

Usage

estimate_loss(
    posterior_samples,
    distribution,
    wrt_option = NULL,
    metric = c("absolute", "lift", "relative_risk")
)

Arguments

posterior_samples  Tibble: returned from sample_from_posterior with 3 columns ‘option_name’, ‘samples’, and ‘sample_id’.
distribution       String: the name of the distribution
wrt_option         String: the option loss is calculated with respect to (wrt). If NULL, the best option will be chosen.
metric             String: the type of loss. absolute will be the difference, on the outcome scale. 0 when best = wrt_option lift will be the (best - wrt_option) / wrt_option, 0 when best = wrt_option relative_risk will be the ratio best/wrt_option, 1 when best = wrt_option

Value

numeric, the loss distribution

Examples

# Requires posterior_samples dataframe. See `sample_from_posterior()`
# for an example.

## Not run:
estimate_loss(posterior_samples = posterior_samples, distribution = "conversion_rate")

## End(Not run)
estimate_value_remaining

Estimate Value Remaining

Description

Estimates value remaining or loss (in terms of percent lift, absolute, or relative).

Usage

```r
estimate_value_remaining(
    input_df,
    distribution,
    priors = list(),
    wrt_option = NULL,
    metric = "lift",
    threshold = 0.95
)
```

Arguments

- **input_df**: Dataframe containing `option_name` (str) and various other columns depending on the distribution type. See vignette for more details.
- **distribution**: String of the distribution name
- **priors**: Optional list of priors. Defaults will be used otherwise.
- **wrt_option**: String the option loss is calculated with respect to (wrt). If NULL, the best option will be chosen.
- **metric**: String the type of loss. `absolute` will be the difference, on the outcome scale. 0 when `best = wrt_option` `lift` will be the `(best - wrt_option) / wrt_option`, 0 when `best = wrt_option` `relative_risk` will be the ratio `best/wrt_option`, 1 when `best = wrt_option`
- **threshold**: The confidence interval specifying what the 'worst case scenario should be. Defaults to 95%. (optional)

Value

numeric value remaining at the specified threshold

Examples

```r
input_df <- tibble::tibble(option_name = c("A", "B", "C"),
    sum_clicks = c(1000, 1000, 1000),
    sum_conversions = c(100, 120, 110))
estimate_value_remaining(input_df, distribution = "conversion_rate")
estimate_value_remaining(input_df, distribution = "conversion_rate")
```
estimate_win_prob

Description

Creates a tibble of win probabilities for each option based on the data observed.

Usage

estimate_win_prob(input_df, distribution, priors = list())

Arguments

input_df        Dataframe containing option_name (str) and various other columns depending
                on the distribution type. See vignette for more details.
distribution    String of the distribution name
priors          Optional list of priors. Defaults will be use otherwise.

Value

tibble object with 2 columns: 'option_name' and 'win_probability' formatted as a percent

Examples

input_df <- tibble::tibble(
  option_name = c("A", "B"),
  sum_clicks = c(1000, 1000),
  sum_conversions = c(100, 120)
)
estimate_win_prob(input_df, "conversion_rate")
estimate_win_prob_given_posterior

Estimate Win Probability Given Posterior Distribution

Description

Estimate Win Probability Given Posterior Distribution

Usage

estimate_win_prob_given_posterior(posterior_samples, winner_is_max = TRUE)

Arguments

- **posterior_samples**: Tibble of data in long form with 2 columns 'option_name' and 'samples'
- **winner_is_max**: Boolean. This should almost always be TRUE. If a larger number is better then this should be TRUE. This should be FALSE for metrics such as CPA or CPC where a higher cost is not necessarily better.

Value

Tibble of each option_name and the win probability expressed as a percentage and a decimal 'raw'

Examples

```r
# Requires posterior_samples dataframe. See `sample_from_posterior()`
# for an example.
## Not run:
estimate_win_prob_given_posterior(posterior_samples = posterior_samples)
estimate_win_prob_given_posterior(
    posterior_samples = posterior_samples,
    winner_is_max = TRUE
)
## End(Not run)
```

estimate_win_prob_vs_baseline

Estimate Win Probability vs. Baseline

Description

Calculates the win probability of the best option compared to a single other option given an input_df
Usage

```r
estimate_win_prob_vs_baseline(
    input_df,
    distribution,
    priors = list(),
    wrt_option
)
```

Arguments

- `input_df`: Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
- `distribution`: String of the distribution name.
- `priors`: Optional list of priors. Defaults will be use otherwise.
- `wrt_option`: String the option win prob is calculated with respect to (wrt). Required.

Value

Tibble of each option_name and the win probability expressed as a percentage and a decimal 'raw'

Examples

```r
input_df <- tibble::tibble(
    option_name = c("A", "B", "C"),
    sum_clicks = c(1000, 1000, 1000),
    sum_conversions = c(100, 120, 110)
)

estimate_win_prob_vs_baseline(input_df = input_df,
    distribution = "conversion_rate",
    wrt_option = "B")
```

---

**estimate_win_prob_vs_baseline_given_posterior**

_Estimate Win Probability vs. Baseline Given Posterior_

Description

Calculates the win probability of the best option compared to a single other option given a posterior distribution.

Usage

```r
estimate_win_prob_vs_baseline_given_posterior(
    posterior_samples,
    distribution,
    wrt_option
)
```
find_best_option

Arguments

posterior_samples
Tibble returned from sample_from_posterior with 3 columns ‘option_name’, ‘samples’, and ‘sample_id’.
distribution String: the distribution name
wrt_option String: the option to compare against the best option.

Value
Tibble of each option_name and the win probability expressed as a percentage and a decimal ‘raw’

Examples

# Requires posterior_samples dataframe. See ‘sample_from_posterior()’
# for an example.
## Not run:
estimate_win_prob_vs_baseline_given_posterior(
    posterior_samples = posterior_samples,
    distribution = "conversion_rate",
    wrt_option = "A")
## End(Not run)

find_best_option Find Best Option

Description
Samples from posterior, calculates win probability, and selects the best option. Note: this can be inefficient if you already have the win probability dataframe. Only use this if that has not already been calculated.

Usage

find_best_option(posterior_samples, distribution)

Arguments

posterior_samples
Tibble returned from sample_from_posterior with 3 columns ‘option_name’, ‘samples’, and ‘sample_id’.
distribution String: name of the distribution

Value
String: the best option name
Examples

```r
# Requires posterior distribution
## Not run:
find_best_option(posterior_samples = posterior_samples, distribution = "conversion_rate")
## End(Not run)
```

### grizbayr package

#### Description
Grizzly Bear - Bayesian Inference Package for A/B and Bandit Marketing Tests

#### Details
See the README on GitHub or the ‘intro’ vignette contained in the package.

### impute_missing_options

#### Description
When win probability is calculated

#### Usage

```r
impute_missing_options(posterior_samples, wp_raw)
```

#### Arguments

- `posterior_samples`:
  - Tibble of data in long form with 2 columns ‘option_name’ and ‘samples’

- `wp_raw`:
  - Tibble of win probabilities with the columns: ‘option_name’ and ‘win_prob_raw’

#### Value

- `wp_raw` table with new rows if option names were missing.
**is_prior_valid**

**Is Prior Valid**

**Description**

Checks if a single valid prior name is in the list of prior values and if that prior value from the list is greater than 0.

**Usage**

```r
is_prior_valid(priors_list, valid_prior)
```

**Arguments**

- **priors_list**  
  A list of valid priors

- **valid_prior**  
  A character string

**Value**

Boolean (TRUE/FALSE)

---

**is_winner_max**

**Is Winner Max**

**Description**

Determines if the max or min function should be used for win probability. If CPA or CPC distribution, lower is better, else higher number is better.

**Usage**

```r
is_winner_max(distribution)
```

**Arguments**

- **distribution**  
  String: the name of the distribution

**Value**

Boolean TRUE/FALSE
**rdirichlet**  
*Random Dirichlet*

**Description**
Randomly samples a vector of length n from a dirichlet distribution parameterized by a vector of alphas PDF of Gamma with scale = 1: \( f(x) = \frac{1}{\text{Gamma}(a)} x^{(a-1)} e^{-x} \)

**Usage**

```r
dirichlet(n, alphas_list)
```

**Arguments**

- `n`: integer, the number of samples
- `alphas_list`: Named List of Integers: parameters of the dirichlet, interpreted as the number of success of each outcome

**Value**

`n x length(alphas)` named tibble representing the probability of observing each outcome

**Examples**

```r
dirichlet(100, list(a = 20, b = 15, c = 60))
```

---

**sample_cm_per_click**  
*Sample CM Per Click*

**Description**
Adds 4 new nested columns to the input_df: 'beta_params', 'gamma_params_rev', 'gamma_params_cost' and 'samples'

**Usage**

```r
sample_cm_per_click(input_df, priors, n_samples = 50000)
```

**Arguments**

- `input_df`: Dataframe containing option_name (str), sum_conversions (dbl), sum_revenue (dbl), and sum_clicks (dbl).
- `priors`: Optional list of priors alpha0, beta0 for Beta, k0, theta0 for Gamma Inverse Revenue, and k01, theta01 for Gamma Cost (uses alternate priors so they can be different from Revenue). Default Beta(1,1) and Gamma(1,250) will be used otherwise.
- `n_samples`: Optional integer value. Defaults to 50,000 samples.
Details

`beta_params' and `gamma_params_rev' in each row should be a tibble of length 2 (α and β parameters and k and θ parameters) `samples' in each row should be a tibble of length `n_samples`

See update_rules vignette for a mathematical representation.

\[ CMPerClick = ConversionsPerClick \times RevPerConversion - CostPerClick \]

Value

input_df with 4 new nested columns `beta_params', `gamma_params_rev', `gamma_params_cost', and `samples'

Description

Adds 2 new nested columns to the input_df: `beta_params' and `samples' `beta_params' in each row should be a tibble of length 2 (α and β parameters) `samples' in each row should be a tibble of length `n_samples'

Usage

sample_conv_rate(input_df, priors, n_samples = 50000)

Arguments

input_df Dataframe containing option_name (str), sum_conversions (dbl), and sum_clicks (dbl).
priors Optional list of priors alpha0 and beta0. Default Beta(1, 1) will be used otherwise.
n_samples Optional integer value. Defaults to 50,000 samples.

Details

See update_rules vignette for a mathematical representation.

\[ \text{Conversion}_i \sim Bernoulli(\phi) \]

\[ \phi \sim Beta(\alpha, \beta) \]

Conversion Rate is sampled from a Beta distribution with a Binomial likelihood of an individual converting.

Value

input_df with 2 new nested columns `beta_params' and `samples'
sample_cpa: Sample Cost Per Activation (CPA)

Description

Adds 3 new nested columns to the input_df: ‘beta_params’, ‘gamma_params’, and ‘samples’ ‘beta_params’ and ‘gamma_params’ in each row should be a tibble of length 2 (α and β parameters and κ and θ parameters) ‘samples’ in each row should be a tibble of length ‘n_samples’

Usage

```r
sample_cpa(input_df, priors, n_samples = 50000)
```

Arguments

- `input_df`: Dataframe containing option_name (str), sum_conversions (dbl), sum_cost (dbl), and sum_clicks (dbl).
- `priors`: Optional list of priors alpha0, beta0 for Beta and k0, theta0 for Gamma. Default Beta(1,1) and Gamma(1,250) will be use otherwise.
- `n_samples`: Optional integer value. Defaults to 50,000 samples.

Details

See update_rules vignette for a mathematical representation. This is a combination of a Beta-Bernoulli update and a Gamma-Exponential update.

\[
\begin{align*}
    &\text{conversion}_i \sim \text{Bernoulli}(\phi) \\
    &\text{cpc}_i \sim \text{Exponential}(\lambda) \\
    &\phi \sim \text{Beta}(\alpha, \beta) \\
    &\lambda \sim \text{Gamma}(k, \theta)
\end{align*}
\]

\[
\text{cpa}_i = \frac{1}{(\text{Bernoulli}(\phi) \ast \text{Exponential}(\lambda))}
\]

\[
\text{averageCPA} = \frac{1}{(\phi \lambda)}
\]

Conversion Rate is sampled from a Beta distribution with a Binomial likelihood of an individual converting.
Average CPC is sampled from a Gamma distribution with an Exponential likelihood of an individual cost.

Value

input_df with 3 new nested columns ‘beta_params’, ‘gamma_params’, and ‘samples’
sample_cpc  

**Sample Cost Per Click**

**Description**

Adds 2 new nested columns to the input_df: ‘gamma_params’ and ‘samples’ ‘gamma_params’ in each row should be a tibble of length 2 (\(k\) and \(\theta\) parameters) ‘samples’ in each row should be a tibble of length ‘n_samples’

**Usage**

```r
taxample_cpc(input_df, priors, n_samples = 50000)
```

**Arguments**

- **input_df**: Dataframe containing option_name (str), sum_clicks (dbl), sum_cost (dbl).
- **priors**: Optional list of priors k0, theta0 for Gamma. Default \(\text{Gamma}(1, 250)\) will be use otherwise.
- **n_samples**: Optional integer value. Defaults to 50,000 samples.

**Details**

See update_rules vignette for a mathematical representation.

\[
cpc \sim \text{Exponential}(\lambda)
\]

\[
\lambda \sim \text{Gamma}(k, \theta)
\]

Average CPC is sampled from a Gamma distribution with an Exponential likelihood of an individual cost.

**Value**

input_df with 2 new nested columns ‘gamma_params’ and ‘samples’

---

sample_ctr  

**Sample Click Through Rate**

**Description**

This is an alias for sample_conv_rate with 2 different input columns. This function calculates posterior samples of \(CTR = \text{clicks/impressions}\). Adds 2 new nested columns to the input_df: ‘beta_params’ and ‘samples’. ‘beta_params’ in each row should be a tibble of length 2 (\(\alpha\) and \(\beta\) parameters) ‘samples’ in each row should be a tibble of length ‘n_samples’
Usage
sample_crt(input_df, priors, n_samples = 50000)

Arguments
input_df    Dataframe containing option_name (str), sum_clicks (dbl), and sum_impressions (dbl).
priors      Optional list of priors alpha0 and beta0. Default Beta(1,1) will be use otherwise.
n_samples   Optional integer value. Defaults to 50,000 samples.

Details
See update_rules vignette for a mathematical representation.

\begin{align*}
\text{Click}_i & \sim \text{Bernoulli}(\phi) \\
\phi & \sim \text{Beta}(\alpha, \beta)
\end{align*}

Click Through Rate is sampled from a Beta distribution with a Binomial likelihood of an individual Clicking

Value
input_df with 2 new nested columns ‘beta_params’ and ‘samples’

---

sample_from_posterior    Sample From Posterior

---

Description
Selects which function to use to sample from the posterior distribution

Usage
sample_from_posterior(
    input_df, 
    distribution, 
    priors = list(), 
    n_samples = 50000
)

Arguments
input_df    Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
distribution String of the distribution name
priors      Optional list of priors. Defaults will be use otherwise.
n_samples   Optional integer value. Defaults to 50,000 samples.
sample_multi_rev_per_session

Value

A tibble with 2 columns: option_name (chr) and samples (dbl) [long form data].

Examples

```r
input_df <- tibble::tibble(
  option_name = c("A", "B"),
  sum_clicks = c(1000, 1000),
  sum_conversions = c(100, 120),
  sum_sessions = c(1000, 1000),
  sum_revenue = c(1000, 1500)
)
sample_from_posterior(input_df, "conversion_rate")
sample_from_posterior(input_df, "rev_per_session")
```

Sample Multi Revenue Per Session

Description

Adds 5 new nested columns to the input_df: ‘dirichlet_params’, ‘gamma_params_A’, ‘gamma_params_B’, and ‘samples’. This samples from multiple revenue per session distributions at once.

Usage

```r
sample_multi_rev_per_session(input_df, priors, n_samples = 50000)
```

Arguments

- **input_df**: Dataframe containing option_name (str), sum_conversions (dbl), sum_sessions (dbl), sum_revenue (dbl), sum_conversion_2 (dbl), sum_sessions_2 (dbl), sum_revenue_2 (dbl).
- **priors**: Optional list of priors alpha0 and beta0. Default Beta(1, 1) will be otherwise.
- **n_samples**: Optional integer value. Defaults to 50,000 samples.

Details

See update_rules vignette for a mathematical representation.

\[
\text{conversion}_i \sim \text{MultiNomial}(\phi_1, \phi_2, \ldots, \phi_k)
\]

\[
\phi_k \sim \text{Dirichlet}(\alpha, \beta)
\]

Conversion Rate is sampled from a Dirichlet distribution with a Multinomial likelihood of an individual converting.
Sample Page Views Per Session (Visit)

**Description**

Adds 2 new nested columns to the input_df: ‘gamma_params’ and ‘samples’ ‘gamma_params’ in each row should be a tibble of length 2 (k and θ parameters) ‘samples’ in each row should be a tibble of length ‘n_samples’

**Usage**

```r
code
```

**Arguments**

- `input_df` Dataframe containing option_name (str), sum_sessions (dbl), and sum_page_views_per_session (dbl).
- `priors` Optional list of priors k0 and theta0. Default Gamma(1,250) will be used otherwise. Gamma(1,1) might also be a good choice for this distribution if you only have a few page views per session.
- `n_samples` Optional integer value. Defaults to 50,000 samples.

**Details**

See update_rules vignette for a mathematical representation.

\[ \text{page_views}_i \sim \text{Poisson}(\lambda) \]

\[ \lambda \sim \text{Gamma}(k, \theta) \]

Page Views Per Visit is sampled from a Gamma distribution with a Poisson likelihood of an individual having n page views by the end of their session.

This is not always the case, so verify your data follows the shape of an Poisson distribution before using this.

**Value**

`input_df` with 2 new nested columns ‘gamma_params’ and ‘samples’
**sample_response_rate**

*Sample Response Rate*

**Description**

This is an alias for `sample_conv_rate` with a different input column. Adds 2 new nested columns to the `input_df`: ‘beta_params’ and ‘samples’ ‘beta_params’ in each row should be a tibble of length 2 (α and β parameters) ‘samples’ in each row should be a tibble of length ‘n_samples’

**Usage**

```r
sample_response_rate(input_df, priors, n_samples = 50000)
```

**Arguments**

- `input_df` Dataframe containing `option_name` (str), `sum_conversions` (dbl), and `sum_sessions` (dbl).
- `priors` Optional list of priors alpha0 and beta0. Default Beta(1, 1) will be otherwise.
- `n_samples` Optional integer value. Defaults to 50,000 samples.

**Details**

See `update_rules` vignette for a mathematical representation.

\[
conversion \sim \text{Bernoulli}(\phi)
\]

\[
\phi \sim \text{Beta}(\alpha, \beta)
\]

Response Rate is sampled from a Beta distribution with a Binomial likelihood of an individual converting.

**Value**

`input_df` with 2 new nested columns ‘beta_params’ and ‘samples’

---

**sample_rev_per_session**

*Sample Rev Per Session*

**Description**

Adds 3 new nested columns to the `input_df`: ‘beta_params’, ‘gamma_params’, and ‘samples’ ‘beta_params’ and ‘gamma_params’ in each row should be a tibble of length 2 (α and β parameters and k and θ parameters) ‘samples’ in each row should be a tibble of length ‘n_samples’
sample_rev_per_session

Usage

sample_rev_per_session(input_df, priors, n_samples = 50000)

Arguments

input_df  Dataframe containing option_name (str), sum_conversions (dbl), sum_revenue (dbl), and sum_clicks (dbl).
priors    Optional list of priors alpha0, beta0 for Beta and k0, theta0 for Gamma. Default Beta(1, 1) and Gamma(1, 250) will be used otherwise.
n_samples Optional integer value. Defaults to 50,000 samples.

Details

See update_rules vignette for a mathematical representation.

\[
\text{RevPerSession} = \text{RevPerOrder} \times \text{OrdersPerClick}
\]

This is a combination of a Beta-Bernoulli update and a Gamma-Exponential update.

\[
\begin{align*}
\text{conversion}_i & \sim \text{Bernoulli}(\phi) \\
\text{revenue}_i & \sim \text{Exponential}(\lambda) \\
\phi & \sim \text{Beta}(\alpha, \beta) \\
\lambda & \sim \text{Gamma}(k, \theta)
\end{align*}
\]

\[
\text{revenue}_i \sim \text{Bernoulli}(\phi) \times \text{Exponential}(\lambda^{-1})
\]

\[
\text{RevPerSession} \sim \phi / \lambda
\]

Conversion Rate is sampled from a Beta distribution with a Binomial likelihood of an individual converting.

Average Rev Per Order is sampled from a Gamma distribution with an Exponential likelihood of Revenue from an individual order. This function makes sense to use if there is a distribution of possible revenue values that can be produced from a single order or conversion.

Value

input_df with 3 new nested columns ‘beta_params’, ‘gamma_params’, and ‘samples’
Description

Adds 2 new nested columns to the input_df: ‘gamma_params’ and ‘samples’ ‘gamma_params’ in each row should be a tibble of length 2 (k and θ parameters) ‘samples’ in each row should be a tibble of length ‘n_samples’

Usage

sample_session_duration(input_df, priors, n_samples = 50000)

Arguments

input_df Dataframe containing option_name (str), sum_sessions (dbl), and sum_duration (dbl).
priors Optional list of priors k0 and theta0. Default Gamma(1, 250) will be use otherwise.
n_samples Optional integer value. Defaults to 50,000 samples.

Details

See update_rules vignette for a mathematical representation.

\[ \text{duration} \sim \text{Exponential}(\lambda) \]
\[ \lambda \sim \text{Gamma}(k, \theta) \]

Session Duration is sampled from a Gamma distribution with a Exponential likelihood of an individual leaving the site or ending a session at time t.

This is not always the case, so verify your data follows the shape of an exponential distribution before using this.

Value

input_df with 2 new nested columns ‘gamma_params’ and ‘samples’
Sample Total CM (Given Impression Count)

Description


Usage

```r
sample_total_cm(input_df, priors, n_samples = 50000)
```

Arguments

- **input_df**: Dataframe containing option_name (str), sum_conversions (dbl), sum_revenue (dbl), and sum_clicks (dbl).
- **priors**: Optional list of priors alpha0, beta0 for Beta, k0, theta0 for Gamma Inverse Revenue, and k01, theta01 for Gamma Cost (uses alternate priors so they can be different from Revenue). Default Beta(1,1) and Gamma(1,250) will be used otherwise.
- **n_samples**: Optional integer value. Defaults to 50,000 samples.

Details

‘beta_params’ and ‘gamma_params’ in each row should be a tibble of length 2 (α and β params and k and θ params). ‘samples’ in each row should be a tibble of length ‘n_samples’.

One assumption in this model is that sum_impressions is not stochastic. This assumes that Clicks are stochastically generated from a set number of Impressions. It does not require that the number of impressions are equal on either side. Generally this assumption holds true in marketing tests where traffic is split 50/50 and very little variance is observed in the number of impressions on either side.

See update_rules vignette for a mathematical representation.

\[
TotalCM = Impr \times ExpectedCTR \times (RevPerOrder \times OrdersPerClick - ExpectedCPC)
\]

Value

input_df with 5 new nested columns ‘beta_params_conv’, ‘beta_params_ctr’, ‘gamma_params_rev’, ‘gamma_params_cost’, and ‘samples’
**update_beta**  
*Update Beta*

**Description**
Updates Beta Distribution with the Beta-Bernoulli conjugate prior update rule

**Usage**
```r
update_beta(alpha, beta, priors = list())
```

**Arguments**
- `alpha`: Double value for alpha (count of successes). Must be 0 or greater.
- `beta`: Double value for beta (count of failures). Must be 0 or greater.
- `priors`: An optional list object that contains alpha0 and beta0. Otherwise the function will use Beta(1,1) as the prior distribution.

**Value**
A tibble object that contains ‘alpha’ and ‘beta’

**Examples**
```r
update_beta(alpha = 1, beta = 5, priors = list(alpha0 = 2, beta0 = 2))
update_beta(alpha = 20000, beta = 50000)
```

---

**update_dirichlet**  
*Update Dirichlet Distribution*

**Description**
This function updates the Dirichlet distribution with the Dirichlet-Multinomial conjugate prior update rule.

**Usage**
```r
update_dirichlet(alpha_0, alpha_1, alpha_2, priors = list())
```

**Arguments**
- `alpha_0`: Double value for alpha_0 (count of failures). Must be 0 or greater.
- `alpha_1`: Double value for alpha_1 (count of successes side 1). Must be 0 or greater.
- `alpha_2`: Double value for alpha_2 (count of successes side 2). Must be 0 or greater.
- `priors`: An optional list object that contains alpha0, alpha1, and alpha2. Otherwise the function will use Dirichlet(1, 1, 1) as the prior distribution.
Details

TODO: This function currently only works in 3 dimensions. Should be extended into N dimensions in the future. Can use ... notation.

Value

tibble with columns alpha_0, alpha_1, and alpha_2

Examples

update_dirichlet(alpha_0 = 20, alpha_1 = 5, alpha_2 = 2)
sample_priors_list <- list(alpha00 = 2, alpha01 = 3, alpha02 = 5)
update_dirichlet(alpha_0 = 20, alpha_1 = 5, alpha_2 = 2, priors = sample_priors_list)

update_gamma

Update Gamma

Description

Updates Gamma Distribution with the Gamma-Exponential conjugate prior update rule. Parameterized by $k$ and $\theta$ (not $\alpha, \beta$)

Usage

update_gamma(k, theta, priors = list(), alternate_priors = FALSE)

Arguments

k Double value for $k$ (total revenue generating events). Must be 0 or greater.
theta Double value for $\theta$ (sum of revenue). Must be 0 or greater.
priors An optional list object that contains k0 and theta0. Otherwise the function will use $\text{Gamma}(1, 250)$ as the prior distribution. If a second gamma distribution is used k01 and theta01 can be defined as separate priors when alternate_priors is set to TRUE.
alternate_priors Boolean Defaults to FALSE. Allows a user to specify alternate prior names so the same prior isn’t required when multiple gamma distributions are used.

Value

A list object that contains ’k’ and ’theta’

Examples

update_gamma(k = 1, theta = 100, priors = list(k0 = 2, theta0 = 1000))
update_gamma(k = 10, theta = 200)
**validate_data_values**  

**Validate Data Values**

**Description**
Validates data values are all greater than 0.

**Usage**
```
validate_data_values(data_values)
```

**Arguments**
- `data_values`  
  List of named data values

**Value**
None

**validate_input_column**  

**Validate Input Column**

**Description**
Validates the input column exists in the dataframe, is of the correct type, and that all values are greater than or equal to 0.

**Usage**
```
validate_input_column(column_name, input_df, greater_than_zero = TRUE)
```

**Arguments**
- `column_name`  
  String value of the column name
- `input_df`  
  Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
- `greater_than_zero`  
  Boolean: Do all values in the column have to be greater than zero?

**Value**
None
validate_input_df  Validate Input DataFrame

Description
Validates the input dataframe has the correct type, correct required column names, that the distribution is valid, that the column types are correct, and that the column values are greater than or equal to 0 when they are numeric.

Usage
validate_input_df(input_df, distribution)

Arguments
input_df Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
distribution String of the distribution name

Value
Bool TRUE if all checks pass.

Examples
input_df <- tibble::tibble(
  option_name = c("A", "B"),
  sum_clicks = c(1000, 1000),
  sum_conversions = c(100, 120)
)
validate_input_df(input_df, "conversion_rate")

validate_posterior_samples  Validate Posterior Samples DataFrame

Description
Function fails if posterior is not shaped correctly.

Usage
validate_posterior_samples(posterior_samples)
**Argument**

- `posterior_samples`  
  Tibble of data in long form with 2 columns `option_name` and `samples`

**Value**

None

---

**validate_priors**  
**Validate Priors**

**Description**

Validates list of priors against a vector of valid priors and if the values are not valid, default priors are returned.

**Usage**

`validate_priors(priors, valid_priors, default_priors)`

**Arguments**

- `priors`  
  List of named priors with double values.
- `valid_priors`  
  A character vector of valid prior names.
- `default_priors`  
  A list of default priors for the distribution.

**Value**

A named list of valid priors for the distribution.

---

**validate_wrt_option**  
**Validate With Respect To Option**

**Description**

Verify that the option provided is in the `poster_samples` dataframe `option_name` column. Raises error if not TRUE.

**Usage**

`validate_wrt_option(wrt_option, posterior_samples)`
validate_wrt_option

Arguments

wrt_option       string name of the option
posterior_samples
    Tibble returned from sample_from_posterior with 3 columns ‘option_name’,
    ‘samples’, and ‘sample_id’.

Value

None
Index

* datasets
  - distribution_column_mapping, 4
  - calculate_multi_rev_per_session, 2
  - calculate_total_cm, 3
  - distribution_column_mapping, 4
  - estimate_all_values, 4
  - estimate_lift, 6
  - estimate_lift_vs_baseline, 7
  - estimate_loss, 8
  - estimate_value_remaining, 9
  - estimate_win_prob, 10
  - estimate_win_prob_given_posterior, 11
  - estimate_win_prob_vs_baseline, 11
  - estimate_win_prob_vs_baseline_given_posterior, 12
  - find_best_option, 13
  - grizbayr, 14
  - impute_missing_options, 14
  - is_prior_valid, 15
  - is_winner_max, 15
  - rdirichlet, 16
  - sample_cm_per_click, 16
  - sample_conv_rate, 17
  - sample_cpa, 18
  - sample_cpc, 19
  - sample_ctr, 19
  - sample_from_posterior, 20
  - sample_multi_rev_per_session, 21
  - sample_page_views_per_session, 22
  - sample_response_rate, 23
  - sample_rev_per_session, 23
  - sample_session_duration, 25
  - sample_total_cm, 26
  - update_beta, 27
  - update_dirichlet, 27
  - update_gamma, 28
  - validate_data_values, 29
  - validate_input_column, 29
  - validate_input_df, 30
  - validate_posterior_samples, 30
  - validate_priors, 31
  - validate_wrt_option, 31