Package: Robyn (via r-universe)

September 26, 2024

Type Package

Title Semi-Automated Marketing Mix Modeling (MMM) from Meta Marketing Science

Version 3.11.1

Maintainer Bernardo Lares <laresbernardo@gmail.com>

Description Semi-Automated Marketing Mix Modeling (MMM) aiming to reduce human bias by means of ridge regression and evolutionary algorithms, enables actionable decision making providing a budget allocation and diminishing returns curves and allows ground-truth calibration to account for causation.

Depends R (>= 4.0.0)

Imports doParallel, doRNG, dplyr, foreach, ggplot2, ggridges, glmnet, jsonlite, lares, lubridate, minpack.lm, nloptr, patchwork, prophet, reticulate, stringr, tidyr

Config/reticulate list(packages = list(list(package = ``nevergrad", pip = TRUE)))

URL https://github.com/facebookexperimental/Robyn, https://facebookexperimental.github.io/Robyn/

BugReports https://github.com/facebookexperimental/Robyn/issues

RoxygenNote 7.2.3

License MIT + file LICENSE

Encoding UTF-8

LazyData true

NeedsCompilation no

Author Gufeng Zhou [aut], Bernardo Lares [cre, aut], Leonel Sentana [aut], Igor Skokan [aut], Meta Platforms, Inc. [cph, fnd]

Repository CRAN

Date/Publication 2024-06-27 07:30:02 UTC

Contents

`adstock_geometric` *Adstocking Transformation (Geometric and Weibull)*

Description

`adstock_geometric()` for Geometric Adstocking is the classic one-parametric adstock function.

`adstock_weibull()` for Weibull Adstocking is a two-parametric adstock function that allows changing decay rate over time, as opposed to the fixed decay rate over time as in Geometric adstock. It has two options, the cumulative density function "CDF" or the probability density function "PDF".

Usage

```
adstock_geometric(x, theta)

adstock_weibull(x, shape, scale, windlen = length(x), type = "cdf")

transform_adstock(
  x,
  adstock,
```

```
  theta = NULL,
  shape = NULL,
  scale = NULL,
  windlen = length(x)
)

plot_adstock(plot = TRUE)
```

Arguments

x	A numeric vector.
theta	Numeric. Theta is the only parameter on Geometric Adstocking and means fixed decay rate. Assuming TV spend on day 1 is 100€ and theta = 0.7, then day 2 has 100 x 0.7 = 70€ worth of effect carried-over from day 1, day 3 has 70 x 0.7 = 49€ from day 2 etc. Rule-of-thumb for common media genre: TV c(0.3, 0.8), OOH/Print/ Radio c(0.1, 0.4), digital c(0, 0.3).
shape, scale	Numeric. Check "Details" section for more details.
windlen	Integer. Length of modelling window. By default, same length as x.
type	Character. Accepts "CDF" or "PDF". CDF, or cumulative density function of the Weibull function allows changing decay rate over time in both C and S shape, while the peak value will always stay at the first period, meaning no lagged effect. PDF, or the probability density function, enables peak value occurring after the first period when shape >=1, allowing lagged effect.
adstock	Character. One of: "geometric", "weibull_cdf", "weibull_pdf".
plot	Boolean. Do you wish to return the plot?

Details

Weibull's CDF (Cumulative Distribution Function) has two parameters, shape & scale, and has flexible decay rate, compared to Geometric adstock with fixed decay rate. The shape parameter controls the shape of the decay curve. Recommended bound is c(0.0001, 2). The larger the shape, the more S-shape. The smaller, the more L-shape. Scale controls the inflexion point of the decay curve. We recommend very conservative bounce of c(0, 0.1), because scale increases the adstock half-life greatly.

Weibull's PDF (Probability Density Function) also shape & scale as parameter and also has flexible decay rate as Weibull CDF. The difference is that Weibull PDF offers lagged effect. When shape > 2, the curve peaks after x = 0 and has NULL slope at x = 0, enabling lagged effect and sharper increase and decrease of adstock, while the scale parameter indicates the limit of the relative position of the peak at x axis; when 1 < shape < 2, the curve peaks after x = 0 and has infinite positive slope at x = 0, enabling lagged effect and slower increase and decrease of adstock, while scale has the same effect as above; when shape = 1, the curve peaks at x = 0 and reduces to exponential decay, while scale controls the inflexion point; when 0 < shape < 1, the curve peaks at x = 0 and has increasing decay, while scale controls the inflexion point. When all possible shapes are relevant, we recommend c(0.0001, 10) as bounds for shape; when only strong lagged effect is of interest, we recommend c(2.0001, 10) as bound for shape. In all cases, we recommend conservative bound of c(0, 0.1) for scale. Due to the great flexibility of

Weibull PDF, meaning more freedom in hyperparameter spaces for Nevergrad to explore, it also requires larger iterations to converge.

Run `plot_adstock()` to see the difference visually.

Value

Numeric values. Transformed values.

See Also

Other Transformations: `mic_men()`, `saturation_hill()`

Examples

```
adstock_geometric(rep(100, 5), theta = 0.5)
adstock_weibull(rep(100, 5), shape = 0.5, scale = 0.5, type = "CDF")
adstock_weibull(rep(100, 5), shape = 0.5, scale = 0.5, type = "PDF")

# Wrapped function for either adstock
transform_adstock(rep(100, 10), "weibull_pdf", shape = 1, scale = 0.5)
```

---

dt_prophet_holidays *Robyn Dataset: Holidays by Country*

Description

Contains `prophet`'s "new" default holidays by country. When using own holidays, please keep the header `c("ds", "holiday", "country", "year")`.

Usage

```
data(dt_prophet_holidays)
```

Format

An object of class `"data.frame"`

ds Date

holiday Name of celebrated holiday

country Code for the country (Alpha-2)

year Year of `ds`

Value

data.frame

Dataframe. Contains `prophet`'s default holidays by country.

See Also

Other Dataset: dt_simulated_weekly

Examples

```
data(dt_prophet_holidays)
head(dt_prophet_holidays)
```

---

dt_simulated_weekly *Robyn Dataset: MMM Demo Data*

Description

Simulated MMM data. Input time series should be daily, weekly or monthly.

Usage

```
data(dt_simulated_weekly)
```

Format

An object of class "data.frame"

DATE Date

revenue Daily total revenue

tv_S Television

ooh_S Out of home

... ...

Value

data.frame

Dataframe. Contains simulated dummy dataset to test and run demo.

See Also

Other Dataset: dt_prophet_holidays

Examples

```
data(dt_simulated_weekly)
head(dt_simulated_weekly)
```

fit_spend_exposure *Fit a nonlinear model for media spend and exposure*

Description

This function is called in `robyn_engineering()`. It uses the Michaelis-Menten function to fit the nonlinear model. Fallback model is the simple linear model `lm()` in case the nonlinear model is fitting worse. A bad fit here might result in unreasonable model results. Two options are recommended: Either splitting the channel into sub-channels to achieve better fit, or just use spend as `paid_media_vars`

Usage

```
fit_spend_exposure(dt_spendModInput, mediaCostFactor, paid_media_var)
```

Arguments

`dt_spendModInput`	data.frame. Containing channel spends and exposure data.
`mediaCostFactor`	Numeric vector. The ratio between raw media exposure and spend metrics.
`paid_media_var`	Character. Paid media variable.

Value

List. Containing the all spend-exposure model results.

hyper_limits *Check hyperparameter limits*

Description

Reference data.frame that shows the upper and lower bounds valid for each hyperparameter.

Usage

```
hyper_limits()
```

Value

Dataframe. Contains upper and lower bounds for each hyperparameter.

Examples

```
hyper_limits()
```

hyper_names *Get correct hyperparameter names*

Description

Output all hyperparameter names and help specifying the list of hyperparameters that is inserted into `robyn_inputs(hyperparameters = ...)`

Usage

```
hyper_names(adstock, all_media, all_vars = NULL)
```

Arguments

`adstock`	Character. Default to `InputCollect$adstock`. Accepts "geometric", "weibull_cdf" or "weibull_pdf"
`all_media`	Character vector. Default to `InputCollect$all_media`. Includes `InputCollect$paid_media_spends` and `InputCollect$organic_vars`.
`all_vars`	Used to check the penalties inputs, especially for refreshing models.

Value

Character vector. Names of hyper-parameters that should be defined.

Guide to setup hyperparameters

1. Get correct hyperparameter names: All variables in `paid_media_vars` or `organic_vars` require hyperprameters and will be transformed by adstock & saturation. Difference between `paid_media_vars` and `organic_vars` is that `paid_media_vars` has spend that needs to be specified in `paid_media_spends` specifically. Run `hyper_names()` to get correct hyperparameter names. All names in hyperparameters must equal names from `hyper_names()`, case sensitive.
2. Get guidance for setting hyperparameter bounds: For geometric adstock, use theta, alpha & gamma. For both weibull adstock options, use shape, scale, alpha, gamma.
   - Theta: In geometric adstock, theta is decay rate. guideline for usual media genre: TV c(0.3, 0.8), OOH/Print/Radio c(0.1, 0.4), digital c(0, 0.3)
   - Shape: In weibull adstock, shape controls the decay shape. Recommended c(0.0001, 2). The larger, the more S-shape. The smaller, the more L-shape. Channel-type specific values still to be investigated
   - Scale: In weibull adstock, scale controls the decay inflexion point. Very conservative recommended bounce c(0, 0.1), because scale can increase adstocking half-life greatly. Channel-type specific values still to be investigated
   - Gamma: In s-curve transformation with hill function, gamma controls the inflexion point. Recommended bounce c(0.3, 1). The larger the gamma, the later the inflection point in the response curve
3. Set each hyperparameter bounds. They either contains two values e.g. c(0, 0.5), or only one value (in which case you've "fixed" that hyperparameter)

Helper plots

plot_adstock Get adstock transformation example plot, helping you understand geometric/theta and weibull/shape/scale transformation

plot_saturation Get saturation curve transformation example plot, helping you understand hill/alpha/gamma transformation

Examples

```
media <- c("facebook_S", "print_S", "tv_S")
hyper_names(adstock = "geometric", all_media = media)

hyperparameters <- list(
  facebook_S_alphas = c(0.5, 3), # example bounds for alpha
  facebook_S_gammas = c(0.3, 1), # example bounds for gamma
  facebook_S_thetas = c(0, 0.3), # example bounds for theta
  print_S_alphas = c(0.5, 3),
  print_S_gammas = c(0.3, 1),
  print_S_thetas = c(0.1, 0.4),
  tv_S_alphas = c(0.5, 3),
  tv_S_gammas = c(0.3, 1),
  tv_S_thetas = c(0.3, 0.8)
)

# Define hyper_names for weibull adstock
hyper_names(adstock = "weibull", all_media = media)

hyperparameters <- list(
  facebook_S_alphas = c(0.5, 3), # example bounds for alpha
  facebook_S_gammas = c(0.3, 1), # example bounds for gamma
  facebook_S_shapes = c(0.0001, 2), # example bounds for shape
  facebook_S_scales = c(0, 0.1), # example bounds for scale
  print_S_alphas = c(0.5, 3),
  print_S_gammas = c(0.3, 1),
  print_S_shapes = c(0.0001, 2),
  print_S_scales = c(0, 0.1),
  tv_S_alphas = c(0.5, 3),
  tv_S_gammas = c(0.3, 1),
  tv_S_shapes = c(0.0001, 2),
  tv_S_scales = c(0, 0.1)
)
```

---

mic_men *Michaelis-Menten Transformation*

Description

The Michaelis-Menten `mic_men()` function is used to fit the spend exposure relationship for paid media variables, when exposure metrics like impressions, clicks or GRPs are provided in `paid_media_vars` instead of spend metric.

Usage

```
mic_men(x, Vmax, Km, reverse = FALSE)
```

Arguments

x	Numeric value or vector. Input media spend when reverse = FALSE. Input media exposure metrics (impression, clicks, GRPs, etc.) when reverse = TRUE.
Vmax	Numeric Indicates maximum rate achieved by the system.
Km	Numeric. The Michaelis constant.
reverse	Boolean. Input media spend when reverse = FALSE. Input media exposure metrics (impression, clicks, GRPs etc.) when reverse = TRUE.

Value

Numeric values. Transformed values.

See Also

Other Transformations: adstock_geometric(), saturation_hill()

Examples

```
mic_men(x = 5:10, Vmax = 5, Km = 0.5)
```

---

prophet_decomp *Conduct prophet decomposition*

Description

When prophet_vars in robyn_inputs() is specified, this function decomposes trend, season, holiday and weekday from the dependent variable.

Usage

```
prophet_decomp(
  dt_transform,
  dt_holidays,
  prophet_country,
  prophet_vars,
  prophet_signs,
  factor_vars,
  context_vars,
  organic_vars,
  paid_media_spends,
  intervalType,
  dayInterval,
  custom_params
)
```

Arguments

dt_transform	A data.frame with all model features. Must contain ds column for time variable values and dep_var column for dependent variable values.
dt_holidays	data.frame. Raw input holiday data. Load standard Prophet holidays using data("dt_prophet_holidays")
context_vars, paid_media_spends, intervalType, dayInterval, prophet_country, prophet_vars, prophet_signs, factor_vars	As included in InputCollect
organic_vars	Character vector. Typically newsletter sendings, push-notifications, social media posts etc. Compared to paid_media_vars organic_vars are often marketing activities without clear spends.
custom_params	List. Custom parameters passed to prophet()

Value

A list containing all prophet decomposition output.

Robyn *Robyn MMM Project from Meta Marketing Science*

Description

Robyn is an automated Marketing Mix Modeling (MMM) code. It aims to reduce human bias by means of ridge regression and evolutionary algorithms, enables actionable decision making providing a budget allocator and diminishing returns curves and allows ground-truth calibration to account for causation.

Author(s)

Gufeng Zhou (gufeng@meta.com)

Leonel Sentana (leonelsentana@meta.com)

Igor Skokan (igorskokan@meta.com)

Bernardo Lares (bernardolares@meta.com)

See Also

Useful links:

- https://github.com/facebookexperimental/Robyn
- https://facebookexperimental.github.io/Robyn/
- Report bugs at https://github.com/facebookexperimental/Robyn/issues

robyn_allocator *Budget Allocator*

Description

robyn_allocator() function returns a new split of media variable spends that maximizes the total media response.

Usage

```
robyn_allocator(
  robyn_object = NULL,
  select_build = 0,
  InputCollect = NULL,
  OutputCollect = NULL,
  select_model = NULL,
  json_file = NULL,
  scenario = "max_response",
  total_budget = NULL,
  target_value = NULL,
  date_range = "all",
  channel_constr_low = NULL,
  channel_constr_up = NULL,
  channel_constr_multiplier = 3,
  optim_algo = "SLSQP_AUGLAG",
  maxeval = 1e+05,
  constr_mode = "eq",
  plots = TRUE,
  plot_folder = NULL,
  plot_folder_sub = NULL,
  export = TRUE,
  quiet = FALSE,
  ui = FALSE,
  ...
)

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

## S3 method for class 'robyn_allocator'
plot(x, ...)
```

Arguments

robyn_object	Character or List. Path of the Robyn.RDS object that contains all previous modeling information or the imported list.

`select_build` Integer. Default to the latest model build. `select_build = 0` selects the initial model. `select_build = 1` selects the first refresh model.

`InputCollect` List. Contains all input parameters for the model. Required when `robyn_object` is not provided.

`OutputCollect` List. Containing all model result. Required when `robyn_object` is not provided.

`select_model` Character. A model `SolID`. When `robyn_object` is provided, `select_model` defaults to the already selected `SolID`. When `robyn_object` is not provided, `select_model` must be provided with `InputCollect` and `OutputCollect`, and must be one of `OutputCollect$allSolutions`.

`json_file` Character. JSON file to import previously exported inputs or recreate a model. To generate this file, use `robyn_write()`. If you didn't export your data in the json file as "raw_data", `dt_input` must be provided; `dt_holidays` input is optional.

`scenario` Character. Accepted options are: `"max_response"`, `"target_efficiency"`. Scenario `"max_response"` answers the question: "What's the potential revenue/conversions lift with the same (or custom) spend level in `date_range` and what is the allocation and expected response mix?" Scenario `"target_efficiency"` optimizes ROAS or CPA and answers the question: "What's the potential revenue/conversions lift and spend levels based on a `target_value` for CPA/ROAS and what is the allocation and expected response mix?" Deprecated scenario: `"max_response_expected_spend"`.

`total_budget` Numeric. Total marketing budget for all paid channels for the period in `date_range`.

`target_value` Numeric. When using the scenario `"target_efficiency"`, target_value is the desired ROAS or CPA with no upper spend limit. Default is set to 80% of initial ROAS or 120% of initial CPA, when `"target_value = NULL"`.

`date_range` Character. Date(s) to apply adstocked transformations and pick mean spends per channel. Set one of: "all", "last", or "last_n" (where n is the last N dates available), date (i.e. "2022-03-27"), or date range (i.e. `c("2022-01-01", "2022-12-31")`). Default to "all".

`channel_constr_low`, `channel_constr_up`

Numeric vectors. The lower and upper bounds for each paid media variable when maximizing total media response. For example, `channel_constr_low = 0.7` means minimum spend of the variable is 70 average, using non-zero spend values, within `date_min` and `date_max` date range. Both constrains must be length 1 (same for all values) OR same length and order as `paid_media_spends`. It's not recommended to 'exaggerate' upper bounds, especially if the new level is way higher than historical level. Lower bound must be >=0.01, and upper bound should be < 5.

`channel_constr_multiplier`

Numeric. Default to 3. For example, if `channel_constr_low` and `channel_constr_up` are 0.8 to 1.2, the range is 0.4. The allocator will also show the optimum solution for a larger constraint range of 0.4 x 3 = 1.2, or 0.4 to 1.6, to show the optimization potential to support allocation interpretation and decision.

`optim_algo` Character. Default to `"SLSQP_AUGLAG"`, short for "Sequential Least-Squares Quadratic Programming" and "Augmented Lagrangian". Alternatively, `""MMA_AUGLAG"`,

	short for "Methods of Moving Asymptotes". More details see the documentation of NLopt here.
`maxeval`	Integer. The maximum iteration of the global optimization algorithm. Defaults to 100000.
`constr_mode`	Character. Options are `"eq"` or `"ineq"`, indicating constraints with equality or inequality.
`plots`	Boolean. Generate plots?
`plot_folder`	Character. Path for saving plots and files. Default to `robyn_object` and saves plot in the same directory as `robyn_object`.
`plot_folder_sub`	Character. Sub path for saving plots. Will overwrite the default path with timestamp or, for refresh and allocator, simply overwrite files.
`export`	Boolean. Export outcomes into local files?
`quiet`	Boolean. Keep messages off?
`ui`	Boolean. Save additional outputs for UI usage. List outcome.
`...`	Additional parameters passed to `robyn_outputs()`.
`x`	`robyn_allocator()` output.

Value

A list object containing allocator result.

List. Contains optimized allocation results and plots.

Examples

```
## Not run:
# Having InputCollect and OutputCollect results
AllocatorCollect <- robyn_allocator(
  InputCollect = InputCollect,
  OutputCollect = OutputCollect,
  select_model = "1_2_3",
  scenario = "max_response",
  channel_constr_low = 0.7,
  channel_constr_up = c(1.2, 1.5, 1.5, 1.5, 1.5),
  channel_constr_multiplier = 4,
  date_range = "last_26",
  export = FALSE
)
# Print a summary
print(AllocatorCollect)
# Plot the allocator one-pager
plot(AllocatorCollect)

## End(Not run)
```

robyn_clusters *Clustering to Reduce Number of Models based on ROI and Errors*

Description

robyn_clusters() uses output from robyn_run(), to reduce the number of models and create bootstrapped confidence interval and help the user pick up the best (lowest combined error) of the most different kinds (clusters) of models.

Usage

```
robyn_clusters(
  input,
  dep_var_type,
  cluster_by = "hyperparameters",
  all_media = NULL,
  k = "auto",
  wss_var = 0.06,
  max_clusters = 10,
  limit = 1,
  weights = rep(1, 3),
  dim_red = "PCA",
  quiet = FALSE,
  export = FALSE,
  seed = 123,
  ...
)
```

Arguments

input	robyn_export()'s output or pareto_aggregated.csv results.
dep_var_type	Character. For dep_var_type 'revenue', ROI is used for clustering. For conversion', CPA is used for clustering.
cluster_by	Character. Any of: "performance" or "hyperparameters".
all_media	Character vector. Default to InputCollect$all_media. Includes InputCollect$paid_media_spends and InputCollect$organic_vars.
k	Integer. Number of clusters
wss_var	Numeric. Used to pick automatic k value, when k is NULL based on WSS variance while considering limit clusters. Values between (0, 1). Default value could be 0.05 to consider convergence.
max_clusters	Integer. Maximum number of clusters.
limit	Integer. Top N results per cluster. If kept in "auto", will select k as the cluster in which the WSS variance was less than 5%.

weights	Vector, size 3. How much should each error weight? Order: nrmse, decomp.rssd, mape. The highest the value, the closer it will be scaled to origin. Each value will be normalized so they all sum 1.
dim_red	Character. Select dimensionality reduction technique. Pass any of: `c("PCA", "tSNE", "all", "none")`.
quiet	Boolean. Keep quiet? If not, print messages.
export	Export plots into local files?
seed	Numeric. Seed for reproducibility
...	Additional parameters passed to `lares::clusterKmeans()`.

Value

List. Clustering results as labeled data.frames and plots.

Author(s)

Bernardo Lares (bernardolares@meta.com)

Examples

```
## Not run:
# Having InputCollect and OutputCollect results
cls <- robyn_clusters(
  input = OutputCollect,
  all_media = InputCollect$all_media,
  k = 3, limit = 2,
  weights = c(1, 1, 1.5)
)

## End(Not run)
```

---

robyn_converge *Check Models Convergence*

Description

`robyn_converge()` consumes `robyn_run()` outputs and calculate convergence status and builds convergence plots. Convergence is calculated by default using the following criteria (having kept the default parameters: sd_qtref = 3 and med_lowb = 2):

Criteria #1: Last quantile's standard deviation < first 3 quantiles' mean standard deviation

Criteria #2: Last quantile's absolute median < absolute first quantile's absolute median - 2 * first 3 quantiles' mean standard deviation

Both mentioned criteria have to be satisfied to consider MOO convergence.

Usage

```
robyn_converge(
  OutputModels,
  n_cuts = 20,
  sd_qtref = 3,
  med_lowb = 2,
  nrmse_win = c(0, 0.998),
  ...
)
```

Arguments

OutputModels	List. Output from robyn_run().
n_cuts	Integer. Default to 20 (5% cuts each).
sd_qtref	Integer. Reference quantile of the error convergence rule for standard deviation (Criteria #1). Defaults to 3.
med_lowb	Integer. Lower bound distance of the error convergence rule for median. (Criteria #2). Default to 3.
nrmse_win	Numeric vector. Lower and upper quantiles thresholds to winsorize NRMSE. Set values within [0,1]; default: c(0, 0.998) which is 1/500.
...	Additional parameters

Value

List. Plots and MOO convergence results.

Examples

```
## Not run:
# Having OutputModels results
MOO <- robyn_converge(
  OutputModels,
  n_cuts = 10,
  sd_qtref = 3,
  med_lowb = 3
)

## End(Not run)
```

---

robyn_inputs *Input Data Check & Transformation*

Description

robyn_inputs() is the function to input all model parameters and check input correctness for the initial model build. It includes the engineering process results that conducts trend, season, holiday & weekday decomposition using Facebook's time-series forecasting library prophet and fit a non-linear model to spend and exposure metrics in case exposure metrics are used in paid_media_vars.

Usage

```
robyn_inputs(
  dt_input = NULL,
  dep_var = NULL,
  dep_var_type = NULL,
  date_var = "auto",
  paid_media_spends = NULL,
  paid_media_vars = NULL,
  paid_media_signs = NULL,
  organic_vars = NULL,
  organic_signs = NULL,
  context_vars = NULL,
  context_signs = NULL,
  factor_vars = NULL,
  dt_holidays = Robyn::dt_prophet_holidays,
  prophet_vars = NULL,
  prophet_signs = NULL,
  prophet_country = NULL,
  adstock = NULL,
  hyperparameters = NULL,
  window_start = NULL,
  window_end = NULL,
  calibration_input = NULL,
  json_file = NULL,
  InputCollect = NULL,
  ...
)

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

Arguments

dt_input	data.frame. Raw input data. Load simulated dataset using `data("dt_simulated_weekly")`
dep_var	Character. Name of dependent variable. Only one allowed
dep_var_type	Character. Type of dependent variable as "revenue" or "conversion". Will be used to calculate ROI or CPI, respectively. Only one allowed and case sensitive.
date_var	Character. Name of date variable. Daily, weekly and monthly data supported. `date_var` must have format "2020-01-01" (YYY-MM-DD). Default to automatic date detection.
paid_media_spends	Character vector. Names of the paid media variables. The values on each of these variables must be numeric. Also, `paid_media_spends` must have same order and length as `paid_media_vars` respectively.
paid_media_vars	Character vector. Names of the paid media variables' exposure level metrics (impressions, clicks, GRP etc) other than spend. The values on each of these

variables must be numeric. These variables are not being used to train the model but to check relationship and recommend to split media channels into sub-channels (e.g. fb_retargeting, fb_prospecting, etc.) to gain more variance. `paid_media_vars` must have same order and length as `paid_media_spends` respectively and is not required.

`paid_media_signs`
: Character vector. Choose any of `c("default", "positive", "negative")`. Control the signs of coefficients for `paid_media_vars`. Must have same order and same length as `paid_media_vars`. By default, all values are set to 'positive'.

`organic_vars`
: Character vector. Typically newsletter sendings, push-notifications, social media posts etc. Compared to `paid_media_vars` `organic_vars` are often marketing activities without clear spends.

`organic_signs`
: Character vector. Choose any of "default", "positive", "negative". Control the signs of coefficients for `organic_vars` Must have same order and same length as `organic_vars`. By default, all values are set to "positive".

`context_vars`
: Character vector. Typically competitors, price & promotion, temperature, unemployment rate, etc.

`context_signs`
: Character vector. Choose any of `c("default", "positive", "negative")`. Control the signs of coefficients for context_vars. Must have same order and same length as `context_vars`. By default it's set to 'defualt'.

`factor_vars`
: Character vector. Specify which of the provided variables in organic_vars or context_vars should be forced as a factor.

`dt_holidays`
: data.frame. Raw input holiday data. Load standard Prophet holidays using `data("dt_prophet_holidays")`

`prophet_vars`
: Character vector. Include any of "trend", "season", "weekday", "monthly", "holiday" or NULL. Highly recommended to use all for daily data and "trend", "season", "holiday" for weekly and above cadence. Set to NULL to skip prophet's functionality.

`prophet_signs`
: Character vector. Choose any of "default", "positive", "negative". Control the signs of coefficients for `prophet_vars`. Must have same order and same length as `prophet_vars`. By default, all values are set to "default".

`prophet_country`
: Character. Only one country allowed. Includes national holidays for all countries, whose list can be found loading `data("dt_prophet_holidays")`.

`adstock`
: Character. Choose any of "geometric", "weibull_cdf", "weibull_pdf". Weibull adstock is a two-parametric function and thus more flexible, but takes longer time than the traditional geometric one-parametric function. CDF, or cumulative density function of the Weibull function allows changing decay rate over time in both C and S shape, while the peak value will always stay at the first period, meaning no lagged effect. PDF, or the probability density function, enables peak value occurring after the first period when shape >=1, allowing lagged effect. Run `plot_adstock()` to see the difference visually. Time estimation: with geometric adstock, 2000 iterations * 5 trials on 8 cores, it takes less than 30 minutes. Both Weibull options take up to twice as much time.

`hyperparameters`
: List. Contains hyperparameter lower and upper bounds. Names of elements in list must be identical to output of `hyper_names()`. To fix hyperparameter values, provide only one value.

`window_start`, `window_end`
: Character. Set start and end dates of modelling period. Recommended to not start in the first date in dataset to gain adstock effect from previous periods. Also, columns to rows ratio in the input data to be >=10:1, or in other words at least 10 observations to 1 independent variable. This window will determine the date range of the data period within your dataset you will be using to specifically regress the effects of media, organic and context variables on your dependent variable. We recommend using a full `dt_input` dataset with a minimum of 1 year of history, as it will be used in full for the model calculation of trend, seasonality and holidays effects. Whereas the window period will determine how much of the full data set will be used for media, organic and context variables.

`calibration_input`
: data.frame. Optional. Provide experimental results to calibrate. Your input should include the following values for each experiment: channel, liftStartDate, liftEndDate, liftAbs, spend, confidence, metric. You can calibrate any spend or organic variable with a well designed experiment. You can also use experimental results from multiple channels; to do so, provide concatenated channel value, i.e. "channel_A+channel_B". Check "Guide for calibration source" section.

`json_file`
: Character. JSON file to import previously exported inputs or recreate a model. To generate this file, use `robyn_write()`. If you didn't export your data in the json file as "raw_data", `dt_input` must be provided; `dt_holidays` input is optional.

`InputCollect`
: Default to NULL. `robyn_inputs`'s output when `hyperparameters` are not yet set.

`...`
: Additional parameters passed to `prophet` functions.

`x`
: `robyn_inputs()` output.

Value

List. Contains all input parameters and modified results using `Robyn:::robyn_engineering()`. This list is ready to be used on other functions like `robyn_run()` and `print()`. Class: `robyn_inputs`.

Guide for calibration source

1. We strongly recommend to use experimental and causal results that are considered ground truth to calibrate MMM. Usual experiment types are people-based (e.g. Facebook conversion lift) and geo-based (e.g. Facebook GeoLift).
2. Currently, Robyn only accepts point-estimate as calibration input. For example, if 10k$ spend is tested against a hold-out for channel A, then input the incremental return as point-estimate as the example below.
3. The point-estimate has to always match the spend in the variable. For example, if channel A usually has 100k$ weekly spend and the experimental HO is 70

Examples

```
# Using dummy simulated data
InputCollect <- robyn_inputs(
  dt_input = Robyn::dt_simulated_weekly,
  dt_holidays = Robyn::dt_prophet_holidays,
  date_var = "DATE",
  dep_var = "revenue",
  dep_var_type = "revenue",
  prophet_vars = c("trend", "season", "holiday"),
  prophet_country = "DE",
  context_vars = c("competitor_sales_B", "events"),
  paid_media_spends = c("tv_S", "ooh_S", "print_S", "facebook_S", "search_S"),
  paid_media_vars = c("tv_S", "ooh_S", "print_S", "facebook_I", "search_clicks_P"),
  organic_vars = "newsletter",
  factor_vars = "events",
  window_start = "2016-11-23",
  window_end = "2018-08-22",
  adstock = "geometric",
  # To be defined separately
  hyperparameters = NULL,
  calibration_input = NULL
)
print(InputCollect)
```

---

robyn_mmm *Core MMM Function*

Description

`robyn_mmm()` function activates Nevergrad to generate samples of hyperparameters, conducts media transformation within each loop, fits the Ridge regression, calibrates the model optionally, decomposes responses and collects the result. It's an inner function within `robyn_run()`.

Usage

```
robyn_mmm(
  InputCollect,
  hyper_collect,
  iterations,
  cores,
  nevergrad_algo,
  intercept = TRUE,
  intercept_sign,
  ts_validation = TRUE,
  add_penalty_factor = FALSE,
  objective_weights = NULL,
  dt_hyper_fixed = NULL,
  rssd_zero_penalty = TRUE,
```

```
  refresh = FALSE,
  trial = 1L,
  seed = 123L,
  quiet = FALSE,
  ...
)
```

Arguments

InputCollect	List. Contains all input parameters for the model. Required when robyn_object is not provided.
hyper_collect	List. Containing hyperparameter bounds. Defaults to InputCollect$hyperparameters.
iterations	Integer. Number of iterations to run.
cores	Integer. Default to parallel::detectCores() - 1 (all cores except one). Set to 1 if you want to turn parallel computing off.
nevergrad_algo	Character. Default to "TwoPointsDE". Options are c("DE","TwoPointsDE", "OnePlusOne", "DoubleFastGADiscreteOnePlusOne","DiscreteOnePlusOne", "PortfolioDiscreteOnePlusOne", "NaiveTBPSA","cGA", "RandomSearch").
intercept	Boolean. Should intercept(s) be fitted (default=TRUE) or set to zero (FALSE).
intercept_sign	Character. Choose one of "non_negative" (default) or "unconstrained". By default, if intercept is negative, Robyn will drop intercept and refit the model. Consider changing intercept_sign to "unconstrained" when there are context_vars with large positive values.
ts_validation	Boolean. When set to TRUE, Robyn will split data by test, train, and validation partitions to validate the time series. By default the "train_size" range is set to c(0.5, 0.8), but it can be customized or set to a fixed value using the hyperparameters input. For example, if train_size = 0.7, validation size and test size will both be 0.15 and 0.15. When ts_validation = FALSE, nrmse_train is the objective function; when ts_validation = TRUE, nrmse_val is the objective function.
add_penalty_factor	Boolean. Add penalty factor hyperparameters to glmnet's penalty.factor to be optimized by nevergrad. Use with caution, because this feature might add too much hyperparameter space and probably requires more iterations to converge.
objective_weights	Numeric vector. Default to NULL to give equal weights to all objective functions. Order: NRMSE, DECOMP.RSSD, MAPE (when calibration data is provided). When you are not calibrating, only the first 2 values for objective_weights must be defined, i.e. set c(2, 1) to give double weight to the 1st (NRMSE). This is an experimental feature. There's no research on optimal weight setting. Subjective weights might strongly bias modeling results.
dt_hyper_fixed	data.frame or named list. Only provide when loading old model results. It consumes hyperparameters from saved csv pareto_hyperparameters.csv or JSON file to replicate a model.
rssd_zero_penalty	Boolean. When TRUE, the objective function DECOMP.RSSD will penalize models with more 0 media effects additionally. In other words, given the same

	DECOMP.RSSD score, a model with 50% 0-coef variables will get penalized by DECOMP.RSSD * 1.5 (larger error), while another model with no 0-coef variables gets un-penalized with DECOMP.RSSD * 1.
`refresh`	Boolean. Set to `TRUE` when used in `robyn_refresh()`.
`trial`	Integer. Which trial are we running? Used to ID each model.
`seed`	Integer. For reproducible results when running nevergrad.
`quiet`	Boolean. Keep messages off?
`...`	Additional parameters passed to `robyn_outputs()`.

Value

List. MMM results with hyperparameters values.

`robyn_outputs` *Evaluate Models and Output Results into Local Files*

Description

Pack `robyn_plots()`, `robyn_csv()`, and `robyn_clusters()` outcomes on `robyn_run()` results. When `UI=TRUE`, enriched `OutputModels` results with additional plots and objects.

Create a plot to visualize the convergence for each of the datasets when running `robyn_run()`, especially useful for when using ts_validation. As a reference, the closer the test and validation convergence points are, the better, given the time-series wasn't overfitted.

Usage

```
robyn_outputs(
  InputCollect,
  OutputModels,
  pareto_fronts = "auto",
  calibration_constraint = 0.1,
  plot_folder = NULL,
  plot_folder_sub = NULL,
  plot_pareto = TRUE,
  csv_out = "pareto",
  clusters = TRUE,
  select_model = "clusters",
  ui = FALSE,
  export = TRUE,
  all_sol_json = FALSE,
  quiet = FALSE,
  refresh = FALSE,
  ...
)
```

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

robyn_csv(
  InputCollect,
  OutputCollect,
  csv_out = NULL,
  export = TRUE,
  calibrated = FALSE
)

robyn_plots(
  InputCollect,
  OutputCollect,
  export = TRUE,
  plot_folder = OutputCollect$plot_folder,
  ...
)

robyn_onepagers(
  InputCollect,
  OutputCollect,
  select_model = NULL,
  quiet = FALSE,
  export = TRUE,
  plot_folder = OutputCollect$plot_folder,
  baseline_level = 0,
  ...
)

ts_validation(OutputModels, quiet = FALSE, ...)

decomp_plot(
  InputCollect,
  OutputCollect,
  solID = NULL,
  exclude = NULL,
  baseline_level = 0
)
```

Arguments

InputCollect, OutputModels	robyn_inputs() and robyn_run() outcomes.
pareto_fronts	Integer. Number of Pareto fronts for the output. pareto_fronts = 1 returns the best models trading off NRMSE & DECOMP.RSSD. Increase pareto_fronts to get more model choices. pareto_fronts = "auto" selects the min fronts that include at least 100 candidates. To customize this threshold, set value with min_candidates.

`calibration_constraint`
: Numeric. Default to 0.1 and allows 0.01-0.1. When calibrating, 0.1 means top 10 selection. Lower `calibration_constraint` increases calibration accuracy.

`plot_folder`
: Character. Path for saving plots and files. Default to `robyn_object` and saves plot in the same directory as `robyn_object`.

`plot_folder_sub`
: Character. Sub path for saving plots. Will overwrite the default path with timestamp or, for refresh and allocator, simply overwrite files.

`plot_pareto`
: Boolean. Set to `FALSE` to deactivate plotting and saving model one-pagers. Used when testing models.

`csv_out`
: Character. Accepts "pareto" or "all". Default to "pareto". Set to "all" will output all iterations as csv. Set NULL to skip exports into CSVs.

`clusters`
: Boolean. Apply `robyn_clusters()` to output models?

`select_model`
: Character vector. Which models (by `solID`) do you wish to plot the one-pagers and export? Default will take top `robyn_clusters()` results.

`ui`
: Boolean. Save additional outputs for UI usage. List outcome.

`export`
: Boolean. Export outcomes into local files?

`all_sol_json`
: Logical. Add all pareto solutions to json export?

`quiet`
: Boolean. Keep messages off?

`refresh`
: Boolean. Refresh mode

`...`
: Additional parameters passed to `robyn_clusters()`

`x`
: `robyn_outputs()` output.

`OutputCollect`
: `robyn_run(..., export = FALSE)` output.

`calibrated`
: Logical

`baseline_level`
: Integer, from 0 to 5. Aggregate baseline variables, depending on the level of aggregation you need. Default is 0 for no aggregation. 1 for Intercept only. 2 adding trend. 3 adding all prophet decomposition variables. 4. Adding contextual variables. 5 Adding organic variables. Results will be reflected on the waterfall chart.

`solID`
: Character vector. Model IDs to plot.

`exclude`
: Character vector. Manually exclude variables from plot.

Value

(Invisible) list. Class: `robyn_outputs`. Contains processed results based on `robyn_run()` results.

Invisible `NULL`.

Invisible list with `ggplot` plots.

Invisible list with `patchwork` plot(s).

Invisible list with `ggplot` plots.

robyn_refresh *Build Refresh Model*

Description

robyn_refresh() builds updated models based on the previously built models saved in the Robyn.RDS object specified in robyn_object. For example, when updating the initial build with 4 weeks of new data, robyn_refresh() consumes the selected model of the initial build, sets lower and upper bounds of hyperparameters for the new build around the selected hyperparameters of the previous build, stabilizes the effect of baseline variables across old and new builds, and regulates the new effect share of media variables towards the latest spend level. It returns the aggregated results with all previous builds for reporting purposes and produces reporting plots.

You must run robyn_save() to select and save an initial model first, before refreshing.

When should robyn_refresh() **NOT be used:** The robyn_refresh() function is suitable for updating within "reasonable periods". Two situations are considered better to rebuild model instead of refreshing:

1. Most data is new: If initial model was trained with 100 weeks worth of data but we add +50 weeks of new data.

2. New variables are added: If initial model had less variables than the ones we want to start using on new refresh model.

Usage

```
robyn_refresh(
  json_file = NULL,
  robyn_object = NULL,
  dt_input = NULL,
  dt_holidays = Robyn::dt_prophet_holidays,
  refresh_steps = 4,
  refresh_mode = "manual",
  refresh_iters = 1000,
  refresh_trials = 3,
  bounds_freedom = NULL,
  plot_folder = NULL,
  plot_pareto = TRUE,
  version_prompt = FALSE,
  export = TRUE,
  calibration_input = NULL,
  objective_weights = NULL,
  ...
)

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

```
## S3 method for class 'robyn_refresh'
plot(x, ...)
```

Arguments

json_file	Character. JSON file to import previously exported inputs or recreate a model. To generate this file, use robyn_write(). If you didn't export your data in the json file as "raw_data", dt_input must be provided; dt_holidays input is optional.
robyn_object	Character or List. Path of the Robyn.RDS object that contains all previous modeling information or the imported list.
dt_input	data.frame. Should include all previous data and newly added data for the refresh.
dt_holidays	data.frame. Raw input holiday data. Load standard Prophet holidays using data("dt_prophet_holidays").
refresh_steps	Integer. It controls how many time units the refresh model build move forward. For example, refresh_steps = 4 on weekly data means the InputCollect$window_start & InputCollect$window_end move forward 4 weeks. If refresh_steps is smaller than the number of newly provided data points, then Robyn would only use the first N steps of the new data.
refresh_mode	Character. Options are "auto" and "manual". In auto mode, the robyn_refresh() function builds refresh models with given refresh_steps repeatedly until there's no more data available. I manual mode, the robyn_refresh() only moves forward refresh_steps only once. "auto" mode has been deprecated when using json_file input.
refresh_iters	Integer. Iterations per refresh. Rule of thumb is, the more new data added, the more iterations needed. More reliable recommendation still needs to be investigated.
refresh_trials	Integer. Trials per refresh. Defaults to 5 trials. More reliable recommendation still needs to be investigated.
bounds_freedom	Numeric. Percentage of freedom we'd like to allow for the new hyperparameters values compared with the model to be refreshed. If set to NULL (default) the value will be calculated as refresh_steps / rollingWindowLength. Applies to all hyperparameters.
plot_folder	Character. Path for saving plots and files. Default to robyn_object and saves plot in the same directory as robyn_object.
plot_pareto	Boolean. Set to FALSE to deactivate plotting and saving model one-pagers. Used when testing models.
version_prompt	Logical. If FALSE, the model refresh version will be selected based on the smallest combined error of normalized NRMSE, DECOMP.RSSD, MAPE. If TRUE, a prompt will be presented to the user to select one of the refreshed models (one-pagers and Pareto CSV files will already be generated).
export	Boolean. Export outcomes into local files?

`calibration_input`
: data.frame. Optional. Provide experimental results to calibrate. Your input should include the following values for each experiment: channel, liftStartDate, liftEndDate, liftAbs, spend, confidence, metric. You can calibrate any spend or organic variable with a well designed experiment. You can also use experimental results from multiple channels; to do so, provide concatenated channel value, i.e. "channel_A+channel_B". Check "Guide for calibration source" section.

`objective_weights`
: Numeric vector. Default to NULL to give equal weights to all objective functions. Order: NRMSE, DECOMP.RSSD, MAPE (when calibration data is provided). When you are not calibrating, only the first 2 values for `objective_weights` must be defined, i.e. set c(2, 1) to give double weight to the 1st (NRMSE). This is an experimental feature. There's no research on optimal weight setting. Subjective weights might strongly bias modeling results.

`...`
: Additional parameters to overwrite original custom parameters passed into initial model.

`x`
: `robyn_refresh()` output.

Value

List. The Robyn object, class `robyn_refresh`.

List. Same as `robyn_run()` but with refreshed models.

Examples

```
## Not run:
# Loading dummy data
data("dt_simulated_weekly")
data("dt_prophet_holidays")
# Set the (pre-trained and exported) Robyn model JSON file
json_file <- "~/Robyn_202208081444_init/RobynModel-2_55_4.json"

# Run \code{robyn_refresh()} with 13 weeks cadence in auto mode
Robyn <- robyn_refresh(
  json_file = json_file,
  dt_input = dt_simulated_weekly,
  dt_holidays = Robyn::dt_prophet_holidays,
  refresh_steps = 13,
  refresh_mode = "auto",
  refresh_iters = 200,
  refresh_trials = 5
)

# Run \code{robyn_refresh()} with 4 weeks cadence in manual mode
json_file2 <- "~/Robyn_202208081444_init/Robyn_202208090847_rf/RobynModel-1_2_3.json"
Robyn <- robyn_refresh(
  json_file = json_file2,
  dt_input = dt_simulated_weekly,
  dt_holidays = Robyn::dt_prophet_holidays,
  refresh_steps = 4,
```

```
  refresh_mode = "manual",
  refresh_iters = 200,
  refresh_trials = 5
)

## End(Not run)
```

`robyn_response` *Response and Saturation Curves*

Description

`robyn_response()` returns the response for a given spend level of a given `paid_media_vars` from a selected model result and selected model build (initial model, refresh model, etc.).

Usage

```
robyn_response(
  InputCollect = NULL,
  OutputCollect = NULL,
  json_file = NULL,
  robyn_object = NULL,
  select_build = NULL,
  select_model = NULL,
  metric_name = NULL,
  metric_value = NULL,
  date_range = NULL,
  dt_hyppar = NULL,
  dt_coef = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

`InputCollect`	List. Contains all input parameters for the model. Required when `robyn_object` is not provided.
`OutputCollect`	List. Containing all model result. Required when `robyn_object` is not provided.
`json_file`	Character. JSON file to import previously exported inputs or recreate a model. To generate this file, use `robyn_write()`. If you didn't export your data in the json file as "raw_data", `dt_input` must be provided; `dt_holidays` input is optional.
`robyn_object`	Character or List. Path of the `Robyn.RDS` object that contains all previous modeling information or the imported list.
`select_build`	Integer. Default to the latest model build. `select_build = 0` selects the initial model. `select_build = 1` selects the first refresh model.

select_model	Character. A model SolID. When robyn_object is provided, select_model defaults to the already selected SolID. When robyn_object is not provided, select_model must be provided with InputCollect and OutputCollect, and must be one of OutputCollect$allSolutions.
metric_name	A character. Selected media variable for the response. Must be one value from paid_media_spends, paid_media_vars or organic_vars
metric_value	Numeric. Desired metric value to return a response for.
date_range	Character. Date(s) to apply adstocked transformations and pick mean spends per channel. Set one of: "all", "last", or "last_n" (where n is the last N dates available), date (i.e. "2022-03-27"), or date range (i.e. c("2022-01-01", "2022-12-31")). Default to "all".
dt_hyppar	A data.frame. When robyn_object is not provided, use dt_hyppar = OutputCollect$resultHypParam It must be provided along select_model, dt_coef and InputCollect.
dt_coef	A data.frame. When robyn_object is not provided, use dt_coef = OutputCollect$xDecompAgg. It must be provided along select_model, dt_hyppar and InputCollect.
quiet	Boolean. Keep messages off?
...	Additional parameters passed to robyn_outputs().

Value

List. Response value and plot. Class: robyn_response.

Examples

```
## Not run:
# Having InputCollect and OutputCollect objects
## Recreate original saturation curve
Response <- robyn_response(
  InputCollect = InputCollect,
  OutputCollect = OutputCollect,
  select_model = select_model,
  metric_name = "facebook_S"
)
Response$plot

## Or you can call a JSON file directly (a bit slower)
# Response <- robyn_response(
#   json_file = "your_json_path.json",
#   dt_input = dt_simulated_weekly,
#   dt_holidays = dt_prophet_holidays,
#   metric_name = "facebook_S"
# )

## Get the "next 100 dollar" marginal response on Spend1
Spend1 <- 20000
Response1 <- robyn_response(
  InputCollect = InputCollect,
  OutputCollect = OutputCollect,
  select_model = select_model,
```

```
  metric_name = "facebook_S",
  metric_value = Spend1, # total budget for date_range
  date_range = "last_1" # last two periods
)
Response1$plot

Spend2 <- Spend1 + 100
Response2 <- robyn_response(
  InputCollect = InputCollect,
  OutputCollect = OutputCollect,
  select_model = select_model,
  metric_name = "facebook_S",
  metric_value = Spend2,
  date_range = "last_1"
)
# ROAS for the 100$ from Spend1 level
(Response2$response_total - Response1$response_total) / (Spend2 - Spend1)

## Get response from for a given budget and date_range
Spend3 <- 100000
Response3 <- robyn_response(
  InputCollect = InputCollect,
  OutputCollect = OutputCollect,
  select_model = select_model,
  metric_name = "facebook_S",
  metric_value = Spend3, # total budget for date_range
  date_range = "last_5" # last 5 periods
)
Response3$plot

## Example of getting paid media exposure response curves
imps <- 10000000
response_imps <- robyn_response(
  InputCollect = InputCollect,
  OutputCollect = OutputCollect,
  select_model = select_model,
  metric_name = "facebook_I",
  metric_value = imps
)
response_imps$response_total / imps * 1000
response_imps$plot

## Example of getting organic media exposure response curves
sendings <- 30000
response_sending <- robyn_response(
  InputCollect = InputCollect,
  OutputCollect = OutputCollect,
  select_model = select_model,
  metric_name = "newsletter",
  metric_value = sendings
)
# response per 1000 sendings
response_sending$response_total / sendings * 1000
```

```
response_sending$plot

## End(Not run)
```

---

robyn_run *Robyn Modelling Function*

Description

robyn_run() consumes robyn_input() outputs, runs robyn_mmm(), and collects all modeling results.

Usage

```
robyn_run(
  InputCollect = NULL,
  dt_hyper_fixed = NULL,
  json_file = NULL,
  ts_validation = FALSE,
  add_penalty_factor = FALSE,
  refresh = FALSE,
  seed = 123L,
  quiet = FALSE,
  cores = NULL,
  trials = 5,
  iterations = 2000,
  rssd_zero_penalty = TRUE,
  objective_weights = NULL,
  nevergrad_algo = "TwoPointsDE",
  intercept = TRUE,
  intercept_sign = "non_negative",
  lambda_control = NULL,
  outputs = FALSE,
  ...
)

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

Arguments

InputCollect	List. Contains all input parameters for the model. Required when robyn_object is not provided.
dt_hyper_fixed	data.frame or named list. Only provide when loading old model results. It consumes hyperparameters from saved csv pareto_hyperparameters.csv or JSON file to replicate a model.

`json_file` Character. JSON file to import previously exported inputs or recreate a model. To generate this file, use `robyn_write()`. If you didn't export your data in the json file as "raw_data", `dt_input` must be provided; `dt_holidays` input is optional.

`ts_validation` Boolean. When set to `TRUE`, Robyn will split data by test, train, and validation partitions to validate the time series. By default the "train_size" range is set to `c(0.5, 0.8)`, but it can be customized or set to a fixed value using the hyperparameters input. For example, if `train_size = 0.7`, validation size and test size will both be 0.15 and 0.15. When `ts_validation = FALSE`, nrmse_train is the objective function; when `ts_validation = TRUE`, nrmse_val is the objective function.

`add_penalty_factor` Boolean. Add penalty factor hyperparameters to glmnet's penalty.factor to be optimized by nevergrad. Use with caution, because this feature might add too much hyperparameter space and probably requires more iterations to converge.

`refresh` Boolean. Set to `TRUE` when used in `robyn_refresh()`.

`seed` Integer. For reproducible results when running nevergrad.

`quiet` Boolean. Keep messages off?

`cores` Integer. Default to `parallel::detectCores() - 1` (all cores except one). Set to 1 if you want to turn parallel computing off.

`trials` Integer. Recommended 5 for default `nevergrad_algo = "TwoPointsDE"`.

`iterations` Integer. Recommended 2000 for default when using `nevergrad_algo = "TwoPointsDE"`.

`rssd_zero_penalty` Boolean. When TRUE, the objective function DECOMP.RSSD will penalize models with more 0 media effects additionally. In other words, given the same DECOMP.RSSD score, a model with 50% 0-coef variables will get penalized by DECOMP.RSSD * 1.5 (larger error), while another model with no 0-coef variables gets un-penalized with DECOMP.RSSD * 1.

`objective_weights` Numeric vector. Default to NULL to give equal weights to all objective functions. Order: NRMSE, DECOMP.RSSD, MAPE (when calibration data is provided). When you are not calibrating, only the first 2 values for `objective_weights` must be defined, i.e. set c(2, 1) to give double weight to the 1st (NRMSE). This is an experimental feature. There's no research on optimal weight setting. Subjective weights might strongly bias modeling results.

`nevergrad_algo` Character. Default to "TwoPointsDE". Options are `c("DE","TwoPointsDE", "OnePlusOne", "DoubleFastGADiscreteOnePlusOne","DiscreteOnePlusOne", "PortfolioDiscreteOnePlusOne", "NaiveTBPSA","cGA", "RandomSearch")`.

`intercept` Boolean. Should intercept(s) be fitted (default=TRUE) or set to zero (FALSE).

`intercept_sign` Character. Choose one of "non_negative" (default) or "unconstrained". By default, if intercept is negative, Robyn will drop intercept and refit the model. Consider changing intercept_sign to "unconstrained" when there are `context_vars` with large positive values.

`lambda_control` Deprecated in v3.6.0.

outputs	Boolean. If set to TRUE, will run robyn_run() and robyn_outputs(), returning a list with OutputModels and OutputCollect results.
...	Additional parameters passed to robyn_outputs().
x	robyn_models() output.

Value

List. Class: robyn_models. Contains the results of all trials and iterations modeled.

List. Contains all trained models. Class: robyn_models.

Examples

```
## Not run:
# Having InputCollect results
OutputModels <- robyn_run(
  InputCollect = InputCollect,
  cores = 2,
  iterations = 200,
  trials = 1
)

## End(Not run)
```

---

robyn_save *Export Robyn Model to Local File [DEPRECATED]*

Description

Use robyn_save() to select and save as .RDS file the initial model.

Usage

```
robyn_save(
  InputCollect,
  OutputCollect,
  robyn_object = NULL,
  select_model = NULL,
  dir = OutputCollect$plot_folder,
  quiet = FALSE,
  ...
)

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

## S3 method for class 'robyn_save'
plot(x, ...)

robyn_load(robyn_object, select_build = NULL, quiet = FALSE)
```

Arguments

InputCollect	List. Contains all input parameters for the model. Required when robyn_object is not provided.
OutputCollect	List. Containing all model result. Required when robyn_object is not provided.
robyn_object	Character or List. Path of the Robyn.RDS object that contains all previous modeling information or the imported list.
select_model	Character. A model SolID. When robyn_object is provided, select_model defaults to the already selected SolID. When robyn_object is not provided, select_model must be provided with InputCollect and OutputCollect, and must be one of OutputCollect$allSolutions.
dir	Character. Existing directory to export JSON file to.
quiet	Boolean. Keep messages off?
...	Additional parameters passed to robyn_outputs().
x	robyn_save() output.
select_build	Integer. Default to the latest model build. select_build = 0 selects the initial model. select_build = 1 selects the first refresh model.

Value

(Invisible) list with filename and summary. Class: robyn_save.

(Invisible) list with imported results

---

robyn_train *Train Robyn Models*

Description

robyn_train() consumes output from robyn_input() and runs the robyn_mmm() on each trial.

Usage

```
robyn_train(
  InputCollect,
  hyper_collect,
  cores,
  iterations,
  trials,
  intercept_sign,
  intercept,
  nevergrad_algo,
  dt_hyper_fixed = NULL,
  ts_validation = TRUE,
  add_penalty_factor = FALSE,
```

```
  objective_weights = NULL,
  rssd_zero_penalty = TRUE,
  refresh = FALSE,
  seed = 123,
  quiet = FALSE
)
```

Arguments

InputCollect	List. Contains all input parameters for the model. Required when robyn_object is not provided.
hyper_collect	List. Containing hyperparameter bounds. Defaults to InputCollect$hyperparameters.
cores	Integer. Default to parallel::detectCores() - 1 (all cores except one). Set to 1 if you want to turn parallel computing off.
iterations	Integer. Recommended 2000 for default when using nevergrad_algo = "TwoPointsDE".
trials	Integer. Recommended 5 for default nevergrad_algo = "TwoPointsDE".
intercept_sign	Character. Choose one of "non_negative" (default) or "unconstrained". By default, if intercept is negative, Robyn will drop intercept and refit the model. Consider changing intercept_sign to "unconstrained" when there are context_vars with large positive values.
intercept	Boolean. Should intercept(s) be fitted (default=TRUE) or set to zero (FALSE).
nevergrad_algo	Character. Default to "TwoPointsDE". Options are c("DE","TwoPointsDE", "OnePlusOne", "DoubleFastGADiscreteOnePlusOne","DiscreteOnePlusOne", "PortfolioDiscreteOnePlusOne", "NaiveTBPSA","cGA", "RandomSearch").
dt_hyper_fixed	data.frame or named list. Only provide when loading old model results. It consumes hyperparameters from saved csv pareto_hyperparameters.csv or JSON file to replicate a model.
ts_validation	Boolean. When set to TRUE, Robyn will split data by test, train, and validation partitions to validate the time series. By default the "train_size" range is set to c(0.5, 0.8), but it can be customized or set to a fixed value using the hyperparameters input. For example, if train_size = 0.7, validation size and test size will both be 0.15 and 0.15. When ts_validation = FALSE, nrmse_train is the objective function; when ts_validation = TRUE, nrmse_val is the objective function.
add_penalty_factor	Boolean. Add penalty factor hyperparameters to glmnet's penalty.factor to be optimized by nevergrad. Use with caution, because this feature might add too much hyperparameter space and probably requires more iterations to converge.
objective_weights	Numeric vector. Default to NULL to give equal weights to all objective functions. Order: NRMSE, DECOMP.RSSD, MAPE (when calibration data is provided). When you are not calibrating, only the first 2 values for objective_weights must be defined, i.e. set c(2, 1) to give double weight to the 1st (NRMSE). This is an experimental feature. There's no research on optimal weight setting. Subjective weights might strongly bias modeling results.

`rssd_zero_penalty`
: Boolean. When TRUE, the objective function DECOMP.RSSD will penalize models with more 0 media effects additionally. In other words, given the same DECOMP.RSSD score, a model with 50% 0-coef variables will get penalized by DECOMP.RSSD * 1.5 (larger error), while another model with no 0-coef variables gets un-penalized with DECOMP.RSSD * 1.

`refresh`
: Boolean. Set to `TRUE` when used in `robyn_refresh()`.

`seed`
: Integer. For reproducible results when running nevergrad.

`quiet`
: Boolean. Keep messages off?

Value

List. Iteration results to include in `robyn_run()` results.

`robyn_update` *Update Robyn Version*

Description

Update Robyn version from Github repository for latest "dev" version or from CRAN for latest "stable" version.

Usage

```
robyn_update(dev = TRUE, ...)
```

Arguments

`dev`
: Boolean. Dev version? If not, CRAN version.

`...`
: Parameters to pass to `remotes::install_github` or `utils::install.packages`, depending on `dev` parameter.

Value

Invisible `NULL`.

robyn_write *Import and Export Robyn JSON files*

Description

robyn_write() generates light JSON files with all the information required to replicate Robyn models. Depending on user inputs, there are 3 use cases: only the inputs data, input data + modeling results data, and input data, modeling results + specifics of a single selected model. To replicate a model, you must provide InputCollect, OutputCollect, and, if OutputCollect contains more than one model, the select_model.

Usage

```
robyn_write(
  InputCollect,
  OutputCollect = NULL,
  select_model = NULL,
  dir = OutputCollect$plot_folder,
  add_data = TRUE,
  export = TRUE,
  quiet = FALSE,
  pareto_df = NULL,
  ...
)

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

robyn_read(json_file = NULL, step = 1, quiet = FALSE, ...)

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

robyn_recreate(json_file, quiet = FALSE, ...)
```

Arguments

InputCollect	robyn_inputs() output.
OutputCollect	robyn_run(..., export = FALSE) output.
select_model	Character. Which model ID do you want to export into the JSON file?
dir	Character. Existing directory to export JSON file to.
add_data	Boolean. Include raw dataset. Useful to recreate models with a single file containing all the required information (no need of CSV).
export	Boolean. Export outcomes into local files?
quiet	Boolean. Keep messages off?

pareto_df	Dataframe. Save all pareto solutions to json file.
...	Additional parameters to export into a custom Extras element.
x	robyn_read() or robyn_write() output.
json_file	Character. JSON file name to read and import.
step	Integer. 1 for import only and 2 for import and output.

Value

(invisible) List. Contains all inputs and outputs of exported model. Class: robyn_write.

Examples

```
## Not run:
InputCollectJSON <- robyn_inputs(
  dt_input = Robyn::dt_simulated_weekly,
  json_file = "~/Desktop/RobynModel-1_29_12.json"
)
print(InputCollectJSON)

## End(Not run)
```

---

saturation_hill *Hill Saturation Transformation*

Description

saturation_hill is a two-parametric version of the Hill function that allows the saturation curve to flip between S and C shape.

Produce example plots for the Hill saturation curve.

Usage

```
saturation_hill(x, alpha, gamma, x_marginal = NULL)

plot_saturation(plot = TRUE)
```

Arguments

x	Numeric vector.
alpha	Numeric. Alpha controls the shape of the saturation curve. The larger the alpha, the more S-shape. The smaller, the more C-shape.
gamma	Numeric. Gamma controls the inflexion point of the saturation curve. The larger the gamma, the later the inflexion point occurs.
x_marginal	Numeric. When provided, the function returns the Hill-transformed value of the x_marginal input.
plot	Boolean. Do you wish to return the plot?

Value

Numeric values. Transformed values.

See Also

Other Transformations: `adstock_geometric()`, `mic_men()`

Examples

```
saturation_hill(c(100, 150, 170, 190, 200), alpha = 3, gamma = 0.5)
```

`set_holidays` *Detect and set date variable interval*

Description

Robyn only accepts daily, weekly and monthly data. This function is only called in `robyn_engineering()`.

Usage

```
set_holidays(dt_transform, dt_holidays, intervalType)
```

Arguments

`dt_transform`	A data.frame. Transformed input data.
`dt_holidays`	A data.frame. Raw input holiday data.
`intervalType`	A character. Accepts one of the values: `c("day","week","month")`

Value

List. Containing the all spend-exposure model results.

Index