Package 'strategize'

January 22, 2026

Type Package

Title Tools for Learning Adversarial or Non-Adversarial Optimal Distributions in High-Dimensional Conjoint Experiments

Version 0.0.1

Date 2025-12-25

Description The 'strategize' package implements methods for learning an optimal or adversarial probability distribution over high-dimensional factors in conjoint (or factorial) experiments. It supports both single-agent (non-adversarial) optimization and adversarial two-player settings, such as multi-stage electoral contexts. Users can estimate the distribution of factor levels that best achieves a chosen objective, optionally under institutional constraints (e.g., two-stage primaries). The package offers a variety of estimation routines, including closed-form solutions for some linear models, gradient-based optimizers for more complex outcome models, and built-in support for inference (via the delta method). It is particularly suitable for exploring and comparing candidate strategies in forced-choice conjoint studies, but is general enough for broader use in high-dimensional policy learning.

URL https://github.com/cjerzak/strategize-software

BugReports https://github.com/cjerzak/strategize-software/issues

Depends R (>= 4.1)

License GPL-3

Encoding UTF-8

Imports stats,
compositions,
graphics,
utils,
glinternet,
matrixStats,
FactorHet,
sandwich,
reticulate

Suggests testthat (>= 3.0.0),
withr,
Matrix,
Rcpp,
devtools,
knitr,
rmarkdown,

ggplot2,
gridExtra,
rlang,
rrapply,
tgp

VignetteBuilder knitr

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Contents

build_backend 2
check_hessian_geometry 3
create_p_list 5
cv_strategize 6
get_final_gradients 11
helpers 12
is_adversarial 12
plot_best_response_curves 13
plot_convergence 15
plot_quadrant_breakdown 17
print.hessian_analysis 18
print.strategize_result 19
strategize 19
strategize.plot 27
strategize_onestep 29
strategize_preset 35
summarize_adversarial 36
summary.strategize_result 37
validate_equilibrium 37

Index **41**

`build_backend` *Build the environment for* `strategize`*. Creates a conda environment in which JAX and NumPy are installed. Users may also create such an environment themselves.*

Description

Build the environment for `strategize`. Creates a conda environment in which JAX and NumPy are installed. Users may also create such an environment themselves.

Usage

```
build_backend(conda_env = "strategize_env", conda = "auto")
```

Arguments

conda_env	Name of the conda environment in which to place the backends. Defaults to "strategize_env".
conda	The path to a conda executable. Using "auto" allows reticulate to attempt to automatically find an appropriate conda binary. Defaults to "auto". If creation fails and a conda binary is found on PATH, the function retries with it.

Value

Invisibly returns NULL. This function is called for its side effects of creating and configuring a conda environment for strategize. This function requires an Internet connection. You can find a list of conda Python paths via: Sys.which("python")

Examples

```
## Not run:
# Create a conda environment named "strategize"
# and install the required Python packages (jax, numpy, etc.)
build_backend(conda_env = "strategize", conda = "auto")

# If you want to specify a particular conda path:
# build_backend(conda_env = "strategize", conda = "/usr/local/bin/conda")

## End(Not run)
```

---

check_hessian_geometry

Check Hessian Geometry at Nash Equilibrium

Description

Computes Hessian eigenvalues to verify proper saddle point structure at the Nash equilibrium. For a valid Nash equilibrium in a zero-sum game:

- AST (maximizer) Hessian should be negative semi-definite (all eigenvalues <= 0)
- DAG (minimizer) Hessian should be positive semi-definite (all eigenvalues >= 0)

Usage

```
check_hessian_geometry(result, tol = 1e-06, verbose = TRUE)
```

Arguments

result	Output from strategize with adversarial = TRUE
tol	Numeric. Tolerance for near-zero eigenvalues (default 1e-6)
verbose	Logical. Whether to print progress messages. Default is TRUE.

Details

Mathematical Foundation

At a Nash equilibrium in a zero-sum game, the Hessian matrices encode local curvature information:

- **AST (maximizes Q)**: The Hessian should be negative semi-definite, meaning all eigenvalues are <= 0. This confirms AST is at a local maximum.
- **DAG (minimizes Q)**: The Hessian should be positive semi-definite, meaning all eigenvalues are >= 0. This confirms DAG is at a local minimum.

The condition number (ratio of largest to smallest eigenvalue magnitude) indicates equilibrium robustness. High condition numbers suggest the equilibrium is sensitive to small perturbations.

Interpretation

- `status = "PASS"`: Valid saddle point with no flat directions
- `status = "WARNING"`: Valid saddle point but has flat directions (weak identification). Consider increasing regularization.
- `status = "FAIL"`: Not a proper saddle point. The solution may not have converged to a true Nash equilibrium.

Value

A list of class `"hessian_analysis"` containing:

status Character: "PASS", "WARNING", or "FAIL"

valid_saddle Logical. TRUE if both Hessians have correct definiteness

eigenvalues_ast Eigenvalues of AST player's Hessian

eigenvalues_dag Eigenvalues of DAG player's Hessian

is_negative_semidefinite_ast Logical. TRUE if AST Hessian is negative semi-definite

is_positive_semidefinite_dag Logical. TRUE if DAG Hessian is positive semi-definite

condition_number_ast Condition number of AST Hessian (stability indicator)

condition_number_dag Condition number of DAG Hessian

flat_directions_ast Number of near-zero eigenvalues (weak identification)

flat_directions_dag Number of near-zero eigenvalues

interpretation Character string with human-readable interpretation

Returns NULL with a warning if Hessian functions are not available.

See Also

`validate_equilibrium` for best-response validation

Examples

```
## Not run:
# Run adversarial strategize
result <- strategize(Y = y, W = w, adversarial = TRUE, nSGD = 500)

# Check Hessian geometry
hess <- check_hessian_geometry(result)
print(hess)
```

```
# Check if it's a valid saddle point
if (hess$valid_saddle) {
  message("Valid Nash equilibrium geometry!")
}

## End(Not run)
```

`create_p_list` *Create Baseline Probability List from Data*

Description

Generates a `p_list` suitable for the `strategize()` function from observed data or using uniform probabilities.

Usage

```
create_p_list(W, uniform = FALSE)
```

Arguments

`W`	Factor matrix (data.frame or matrix) with one column per conjoint factor.
`uniform`	Logical; if `TRUE`, use uniform probabilities across levels; if `FALSE` (default), use observed frequencies from the data.

Details

For conjoint experiments with balanced randomization, use `uniform = TRUE`. For experiments with intentional imbalance or to match observed frequencies, use `uniform = FALSE`.

Value

Named list where each element is a named numeric vector of probabilities. Names correspond to factor levels.

Examples

```
# Create sample factor matrix
W <- data.frame(
  Gender = c("Male", "Female", "Male", "Female"),
  Age = c("Young", "Old", "Young", "Old")
)

# Uniform probabilities (for balanced designs)
p_list_uniform <- create_p_list(W, uniform = TRUE)
print(p_list_uniform)
# $Gender
#   Male Female
#    0.5    0.5
# $Age
#   Young    Old
```

```
#     0.5    0.5

# Observed frequencies
p_list_observed <- create_p_list(W, uniform = FALSE)
print(p_list_observed)
```

---

cv_strategize *Cross-validation for Optimal Stochastic Interventions in Conjoint Analysis*

Description

Performs cross-validation to select the regularization parameter λ (and, if desired, other hyperparameters) for the `strategize` function. This function splits the data by respondent (or user-specified units), trains candidate models under a grid of λ values, and evaluates out-of-sample performance, returning the model that maximizes a chosen criterion (e.g., out-of-sample expected utility or log-likelihood).

Usage

```
cv_strategize(
  Y,
  W,
  X = NULL,
  lambda_seq = NULL,
  lambda = NULL,
  folds = 2L,
  varcov_cluster_variable = NULL,
  competing_group_variable_respondent = NULL,
  competing_group_variable_candidate = NULL,
  competing_group_competition_variable_candidate = NULL,
  pair_id = NULL,
  respondent_id = NULL,
  respondent_task_id = NULL,
  profile_order = NULL,
  p_list = NULL,
  slate_list = NULL,
  use_optax = F,
  K = 1,
  nSGD = 100,
  diff = F,
  adversarial = F,
  adversarial_model_strategy = "four",
  partial_pooling = NULL,
  partial_pooling_strength = 50,
  use_regularization = TRUE,
  force_gaussian = F,
  temperature = 0.5,
  a_init_sd = 0.001,
  learning_rate_max = 0.001,
```

```
  penalty_type = "KL",
  outcome_model_type = "glm",
  neural_mcmc_control = NULL,
  compute_se = T,
  conda_env = "strategize_env",
  conda_env_required = F,
  conf_level = 0.9,
  nFolds_glm = 3L,
  nMonte_adversarial = 5L,
  primary_pushforward = "mc",
  primary_strength = 1,
  primary_n_entrants = 1L,
  primary_n_field = 1L,
  nMonte_Qglm = 100L,
  optim_type = "gd",
  optimism = "extragrad"
)
```

Arguments

`Y`	A numeric or binary response vector. If binary (e.g., 0–1), it should correspond to forced-choice outcomes (1 if candidate `A` is chosen; 0 if candidate `B` is chosen). If numeric, please see details in `strategize` for how outcomes are handled.
`W`	A data frame or matrix representing the randomized conjoint attributes. Each column is a factor or character vector indicating attribute levels for a particular dimension. Multiple columns can be used if the conjoint has multiple attributes.
`X`	Optional covariate matrix or data frame for modeling systematic heterogeneity. If `K > 1`, this is typically required for multi-class or cluster-based models. Otherwise, set `X = NULL`.
`lambda_seq`	A numeric vector of candidate λ values for cross-validation. If `NULL` and `lambda` is also `NULL`, a sequence of values is automatically generated (e.g., via `10^seq(-4, 0, length.out = 5) * sd(Y)`).
`lambda`	A single user-specified λ value. If provided, cross-validation is effectively disabled unless `lambda_seq` is also supplied.
`folds`	An integer or user-specified partitioning indicating the number of cross-validation folds. Defaults to 2. See Details for how data splitting is done.
`varcov_cluster_variable`	An optional clustering variable for robust standard errors. For instance, if the data is from multiple respondents, specify respondent IDs here for cluster-robust inference (via sandwich estimation). If `NULL`, no cluster-based variance correction is used.
`competing_group_variable_respondent`	Optional vector for multi-round or multi-group setups, indicating which respondent belongs to which group. Used for advanced or adversarial designs (e.g., dual-party contexts). If `NULL`, standard usage is assumed.
`competing_group_variable_candidate`	Similar to `competing_group_variable_respondent`, but for candidate-level grouping. If `NULL`, standard usage is assumed.
`competing_group_competition_variable_candidate`	An optional variable for specifying which candidate is in competition with which group. Relevant if multi-step adversarial frameworks are used.

pair_id
: An optional vector (same length as Y) identifying which rows (candidate pairs) belong to the same forced choice. For example, if each respondent evaluates multiple pairs, this ID ensures correct grouping. Required only in certain advanced difference-in-differences or paired analyses.

respondent_id
: A user-specified ID to denote respondent-level grouping, typically used to cluster standard errors or to perform out-of-sample validation by respondent. If NULL, a simple row index is used for splitting.

respondent_task_id
: Another optional ID for tasks (e.g., each respondent might see multiple tasks). Helps in advanced designs. If NULL, ignored.

profile_order
: An optional vector capturing the ordering of candidate profiles within tasks, if multiple profiles are being shown. Used in difference or extended hierarchical modeling.

p_list
: A list of assignment probabilities for each attribute, if known or desired as a baseline. If NULL, each level is assumed to have uniform probability or derived from empirical frequencies in W.

slate_list
: An optional list specifying alternative or restricted sets of attribute levels. Used when a subset of attributes is feasible or when bounding certain strategies in an adversarial design.

use_optax
: Logical. If TRUE, uses the **optax** Python library (via **reticulate**) for gradient-based optimization. If FALSE, uses a default gradient-based approach from **jax**.

K
: An integer specifying the number of mixture components or clusters if X is used (e.g., for multi-class analysis). Defaults to 1 (no mixture).

nSGD
: An integer number of iterations for gradient-based training. Defaults to 100 but can be increased if convergence has not been reached.

diff
: Logical indicating whether a difference-based model (e.g., for forced-choice or difference-in-outcomes) is used. Defaults to FALSE, but set TRUE in certain difference-of-utility designs.

adversarial
: Logical indicating whether to use a two-party or multi-agent *adversarial* approach in the optimization. If TRUE, a min-max (zero-sum) formulation is employed. Defaults to FALSE (single-agent or average-case optimization).

adversarial_model_strategy
: Character string indicating whether to estimate "four" outcome models (primary + general for each group), "two" outcome models (one per group reused for both primary and general), or "neural" (Bayesian Transformer models with party tokens; defaults to a single pooled model across groups and stages. Set neural_mcmc_control$n_bayesian_models = 2 to fit separate AST/DAG models) in adversarial mode.

partial_pooling
: Logical indicating whether to partially pool (shrink) group-specific outcome model coefficients toward a shared average when using the "two" strategy. When NULL, defaults to TRUE in the two-strategy adversarial case.

partial_pooling_strength
: Numeric scalar controlling the amount of shrinkage used for partial pooling in the two-strategy adversarial case.

use_regularization
: Logical; if TRUE, penalty-based regularization is used for the outcome model. Usually set to TRUE for large designs. Defaults to TRUE.

force_gaussian Logical indicating whether a Gaussian family (lm-style) is forced for the outcome model, even if Y is binary. Defaults to FALSE.

temperature Optional numeric temperature controlling the smoothness of Gumbel-Softmax sampling when exploring probability vectors. Smaller values lead to distributions closer to the argmax. Defaults to NULL, which allows internal defaults.

a_init_sd A numeric controlling the random initialization scale for unconstrained parameters in gradient-based optimization. Defaults to 0.001. Larger values can help avoid local minima in complex outcome landscapes.

learning_rate_max
Base learning rate for gradient-based optimizers. Defaults to 0.001.

penalty_type A character string specifying the type of penalty for the *optimal stochastic intervention*, e.g., "KL", "L2", or "LogMaxProb". The default is "KL".

outcome_model_type
Character string indicating the outcome model to use, such as "glm" for generalized linear models or "neural" for a neural-network approximation. Defaults to "glm".

neural_mcmc_control
Optional list overriding default MCMC settings used when outcome_model_type = "neural". Use neural_mcmc_control$uncertainty_scope = "output" to restrict delta-method uncertainty to output-layer parameters. In adversarial neural mode, set neural_mcmc_control$n_bayesian_models = 2 to fit separate AST/DAG models (default is 1 for a single differential model). Use neural_mcmc_control$ModelDi and neural_mcmc_control$ModelDepth to override the Transformer hidden width and depth. Set neural_mcmc_control$cross_candidate_encoder = "term" (or TRUE) to include the opponent-dependent cross-candidate term in pairwise mode, or set neural_mcmc_control$cross_candidate_encoder = "full" to enable a full cross-encoder that jointly encodes both candidates. Use "none" (or FALSE) to disable. For variational inference (subsample_method = "batch_vi"), set neural_mcmc_control$optimizer to "adam" (numpyro.optim) or "adabelief" (optax). Learning-rate decay is controlled by neural_mcmc_control$svi_lr_sche (default "warmup_cosine"), with optional svi_lr_warmup_frac and svi_lr_end_factor.

compute_se Logical; if TRUE, attempts to compute standard errors using M-estimation or the Delta method. Defaults to TRUE.

conda_env A character specifying the name of a Conda environment for **reticulate**. Defaults to "strategize_env".

conda_env_required
Logical. If TRUE, errors if the specified Conda environment conda_env cannot be found. Otherwise tries to fall back gracefully.

conf_level The confidence level (between 0 and 1) for interval estimation, default 0.90.

nFolds_glm An integer specifying the number of folds in internal regression-based cross-validation (if used) for outcome model selection. Defaults to 3.

nMonte_adversarial
A positive integer specifying the number of Monte Carlo draws for the min-max (adversarial) stage, if adversarial = TRUE. Defaults to 5.

primary_pushforward
Character string controlling the primary-stage push-forward estimator. Use "mc" (default) for Monte Carlo sampling with per-draw primary winners, or "linearized" for the faster averaged-weight approximation, or "multi" for multi-candidate primaries.

`primary_strength`
: Numeric scalar controlling primary decisiveness (see `strategize`).

`primary_n_entrants`
: Integer number of entrant candidates per party in multi-candidate primaries.

`primary_n_field`
: Integer number of field candidates per party in multi-candidate primaries.

`nMonte_Qglm`
: An integer specifying the number of Monte Carlo draws for evaluating certain integrals in `glm`-based approximations, default 100.

`optim_type`
: A character describing the optimization routine. Typically `"default"` uses a standard gradient-based approach; set `"tryboth"` or `"SecondOrder"` for testing or advanced routines.

`optimism`
: Character string controlling optimistic / extra-gradient updates for the gradient optimizer. Options: `"extragrad"` (default), `"smp"` (stochastic mirror-prox), `"ogda"`, or `"none"`. Only supported when `use_optax = FALSE`.

Details

`cv_strategize` implements a cross-validation routine for `strategize`. First, the data is split into `folds` parts. For each fold, we train candidate outcome models and compute out-of-sample performance. The best-performing λ is selected. Finally, a refit on the full data is done using the chosen hyperparameters, returning the results of the final `strategize` call with λ set to the best value.

The function supports a wide range of conjoints, including forced-choice (where `diff = TRUE`), multi-cluster outcome modeling (where $K > 1$), and adversarial designs (where `adversarial = TRUE`). Regularization for the outcome model or for the candidate distribution can be enabled via `use_regularization` and `penalty_type`. Cross-validation is particularly helpful when the data is limited or highly dimensional.

Value

A named list with components:

pi_star_point The estimated optimal probability distribution(s) over candidate profiles ($\hat{\boldsymbol{\pi}}^*$).

Q_point_mEst The estimated expected outcome (e.g., vote share) under the selected optimal distribution.

lambda The chosen λ value from cross-validation (and any other relevant hyperparameters).

CVInfo A data frame or matrix summarizing cross-validation results, e.g., in-sample and out-of-sample estimates for each candidate λ.

Other components Various additional objects useful for inference and debugging (e.g., final model fits, standard error estimates, weighting details).

See Also

`strategize` for direct optimization of stochastic interventions in conjoint analysis, including average and adversarial settings.

Examples

```
# ================================================
# Cross-validation to select regularization lambda
# ================================================
set.seed(123)
```

```
n <- 400  # profiles (200 pairs)

# Generate factor matrix
W <- data.frame(
  Gender = sample(c("Male", "Female"), n, replace = TRUE),
  Age = sample(c("35", "50", "65"), n, replace = TRUE),
  Party = sample(c("Dem", "Rep"), n, replace = TRUE)
)

# Simulate outcome with true effects
latent <- 0.2 * (W$Gender == "Female") + 0.15 * (W$Age == "35")
prob <- plogis(latent)

# Create paired forced-choice structure
pair_id <- rep(1:(n/2), each = 2)
Y <- numeric(n)
for (p in unique(pair_id)) {
  idx <- which(pair_id == p)
  winner <- sample(idx, 1, prob = prob[idx])
  Y[idx] <- as.integer(seq_along(idx) == which(idx == winner))
}
profile_order <- rep(1:2, n/2)

# Cross-validate over lambda values
# Lower lambda = less regularization = further from baseline
cv_result <- cv_strategize(
  Y = Y,
  W = W,
  lambda_seq = c(0.01, 0.1, 0.5, 1.0),
  folds = 2,
  pair_id = pair_id,
  respondent_id = pair_id,
  profile_order = profile_order,
  diff = TRUE,
  nSGD = 50,
  compute_se = FALSE
)

# View CV results and selected lambda
print(cv_result$lambda)       # Optimal lambda
print(cv_result$CVInfo)       # Performance at each lambda
print(cv_result$pi_star_point) # Optimal distribution
print(cv_result$Q_point)       # Expected outcome
```

`get_final_gradients` *Get Final Gradient Norms*

Description

Extract the final gradient magnitudes from a strategize result.

Usage

```
get_final_gradients(result)
```

Arguments

result	Output from strategize

Value

Named numeric vector with gradient norms for AST and DAG players.

---

helpers *User-Facing Helper Functions for strategize*

Description

Convenience functions to simplify common tasks when using the strategize package. These functions reduce the complexity of the main API by providing sensible defaults and automatic data processing.

---

is_adversarial *Check if Result is Adversarial*

Description

Utility function to check if a strategize result is from adversarial mode.

Usage

```
is_adversarial(result)
```

Arguments

result	Output from strategize

Value

Logical. TRUE if the result is from adversarial mode.

plot_best_response_curves

Plot Dimension-by-Dimension Best-Response Curves from Adversarial `strategize()` *Output*

Description

`plot_best_response_curves` takes the result of an adversarial `strategize` run (i.e., with `adversarial = TRUE`) and produces dimension-specific best-response curves. Specifically, for a chosen factor dimension d, it plots:

1. The curve of $\pi^*_{\text{dag},d}$ as a function of $\pi_{\text{ast},d}$.
2. The curve of $\pi^*_{\text{ast},d}$ as a function of $\pi_{\text{dag},d}$.

Potential intersection points in this 2D space can indicate approximate equilibria for dimension d, holding the other dimensions fixed at the solution found by `strategize`.

This function is computationally intensive: for each of `nPoints_br` grid values of $\pi_{\text{ast},d}$, it searches over possible $\pi_{\text{dag},d}$ (and vice versa) to find each side's best response, re-running partial objective evaluations. Nonetheless, it provides a direct visualization of how each player (ast or dag) responds to changes in the other's distribution along a single factor dimension.

Usage

```
plot_best_response_curves(
  res,
  d_ = 1,
  nPoints_br = 100L,
  nPoints_heat = 50L,
  title = NULL,
  col_ast = "blue",
  col_dag = "red",
  lwd_ast = 2,
  lwd_dag = 2,
  point_pch = 19,
  silent = FALSE
)
```

Arguments

`res`	A list returned by `strategize`, which must include adversarial references. Internally, `res` should contain items like `res$a_i_ast`, `res$a_i_dag`, the JAX-based functions (`dQ_da_ast`, `dQ_da_dag`, `QFXN`, ...), along with the appropriate unconstrained parameter vectors.
`d_`	(Integer) The dimension of $\pi_{\text{ast}}, \pi_{\text{dag}}$ to examine. For example, if you have multiple factors (dimensions), each is indexed by a positive integer. Defaults to 1.
`nPoints_br`	(Integer) Number of equally spaced grid points in $[0, 1]$ to sample for *the outer* loop. The code does an internal small search for best responses at each grid point. Larger `nPoints_br` => smoother curves but more computation. Defaults to `100L`.

`nPoints_heat` (Integer) Number of grid points for heat map calculations. Defaults to `50L`.

`title` (Character or `NULL`) Main plot title. If `NULL`, an auto-generated title is used, e.g. "Best-Response Curves (Dimension d_=1)".

`col_ast`, `col_dag` (Character) Colors for ast's and dag's best-response curves, respectively. Defaults to `"blue"` (ast) and `"red"` (dag).

`lwd_ast`, `lwd_dag` (Numeric) Line widths for ast and dag curves, respectively. Default is 2.

`point_pch` (Numeric) Symbol for marking the approximate intersection (if found) on the plot. Defaults to 19 (filled circle).

`silent` (Logical) If `TRUE`, suppresses printed messages during the search for intersection. Defaults to `FALSE`.

Details

Mechanics: For each $\pi_{\text{ast},d} \in \{0, \frac{1}{nPoints_br-1}, \ldots, 1\}$, the function temporarily fixes that dimension in the ast player's unconstrained parameter vector. It then does an internal grid search over $\pi_{\text{dag},d}$ to see which value yields the largest dag payoff (lowest ast payoff), consistent with `adversarial=TRUE`. The resulting curve is $\text{BR}_{\text{dag}}(\pi_{\text{ast},d})$.

Likewise, it holds $\pi_{\text{dag},d}$ fixed and searches over $\pi_{\text{ast},d}$ to get $\text{BR}_{\text{ast}}(\pi_{\text{dag},d})$.

The intersection in $(\pi_{\text{ast},d}, \pi_{\text{dag},d})$-space (if one exists in the discretized grid) is a candidate local equilibrium for that factor dimension, *given that the other factor dimensions remain at the solution from* `strategize`.

Performance Caution: This brute force line-search re-runs partial objective evaluations many times, which may be slow for large `nPoints_br` or complex outcome models. Consider using a smaller `nPoints_br` if performance is an issue, or focusing on only a handful of crucial dimensions d.

Value

(Invisibly) A list containing:

`grid_points` A numeric vector of the `nPoints_br` grid values in $[0, 1]$ used for the outer loop.

`br_dag_given_ast` A numeric vector of the same length, giving the dag best-response $\pi_{\text{dag},d}$ at each grid point for $\pi_{\text{ast},d}$.

`br_ast_given_dag` A numeric vector with the ast best-response for each grid point $\pi_{\text{dag},d}$.

See Also

`strategize` for obtaining the result object `res` in adversarial mode. See also `cv_strategize`, and if one-step M-estimation is desired, see `strategize_onestep`.

Examples

```
## Not run:
# ====================================================
# Visualize best-response curves in adversarial mode
# ====================================================
# First, fit an adversarial strategize model
set.seed(42)
n <- 400
```

```
# Generate data with party structure
W <- data.frame(
  Gender = sample(c("Male", "Female"), n, replace = TRUE),
  Age = sample(c("Young", "Middle", "Old"), n, replace = TRUE)
)

# Party affiliations for respondents and candidates
respondent_party <- sample(c("Dem", "Rep"), n/2, replace = TRUE)
candidate_party <- rep(c("Dem", "Rep"), n/2)

Y <- rbinom(n, 1, 0.5)  # Simplified outcome

# Fit adversarial model
adv_result <- strategize(
  Y = Y,
  W = W,
  lambda = 0.1,
  adversarial = TRUE,
  competing_group_variable_respondent = rep(respondent_party, each = 2),
  competing_group_variable_candidate = candidate_party,
  nSGD = 100
)

# Plot best-response curves for Gender dimension (d_ = 1)
# Shows how each party's optimal Gender distribution responds
# to changes in the other party's Gender distribution
plot_best_response_curves(
  res = adv_result,
  d_ = 1,  # Gender is first factor
  nPoints_br = 50,
  title = "Gender: Best-Response Curves",
  col_ast = "blue",   # Democrats
  col_dag = "red"     # Republicans
)

# Intersection point indicates Nash equilibrium for this dimension

## End(Not run)
```

---

plot_convergence *Plot Convergence Diagnostics for Strategize Results*

Description

Visualizes the optimization trajectory from gradient descent, showing gradient magnitudes, loss values, and learning rate adaptation over iterations.

Usage

```
plot_convergence(
  result,
  metrics = c("gradient", "loss"),
```

```
  log_scale = TRUE,
  use_ggplot = FALSE
)
```

Arguments

result	Output from strategize with adversarial = TRUE
metrics	Character vector specifying which metrics to plot. Options are: • "gradient": Gradient magnitude (L2 norm) over iterations • "loss": Objective function value over iterations • "lr": Learning rate adaptation over iterations Default is c("gradient", "loss").
log_scale	Logical. Whether to use log scale for gradient magnitudes. Default is TRUE.
use_ggplot	Logical. If TRUE and ggplot2 is available, uses ggplot2 for visualization. Otherwise uses base R graphics. Default is FALSE.

Details

This function provides diagnostic plots to assess whether the gradient descent optimization has converged to a Nash equilibrium. Key indicators of convergence:

- Gradient magnitudes should decrease toward zero for both players
- Loss values should stabilize (may oscillate slightly in adversarial settings)
- Learning rates should adapt appropriately (decrease as gradients shrink)

In adversarial (two-player) mode, both AST and DAG players' metrics are shown. In non-adversarial mode, only AST metrics are displayed.

Value

If use_ggplot = TRUE and ggplot2 is available, returns a ggplot object. Otherwise, plots are created as side effects and the function returns invisible(NULL).

Examples

```
## Not run:
# Run adversarial strategize
result <- strategize(Y = y, W = w, adversarial = TRUE, nSGD = 500)

# Plot convergence diagnostics
plot_convergence(result)

# Plot only gradient magnitudes with log scale
plot_convergence(result, metrics = "gradient", log_scale = TRUE)

## End(Not run)
```

`plot_quadrant_breakdown` *Plot Four-Quadrant Contribution Breakdown*

Description

Decomposes the equilibrium vote share Q* into contributions from four distinct election scenarios, providing insight into which primary-to-general election pathways drive the Nash equilibrium outcome.

Usage

```
plot_quadrant_breakdown(
  result,
  type = c("bar", "pie"),
  nMonte = 500,
  verbose = TRUE
)
```

Arguments

`result`	Output from `strategize` with `adversarial = TRUE`
`type`	Character string specifying the plot type: • `"bar"`: Stacked bar chart showing contributions • `"pie"`: Pie chart showing proportions Default is `"bar"`.
`nMonte`	Integer. Number of Monte Carlo samples for estimation. Default is 500.
`verbose`	Logical. Whether to print progress messages. Default is `TRUE`.

Details

In adversarial mode, two parties simultaneously optimize their candidate attribute distributions. Each party's "entrant" candidate (drawn from the optimized distribution) competes in a primary against a "field" candidate (drawn from the baseline distribution). The general election outcome depends on which candidates win their respective primaries, creating four possible scenarios. This function visualizes the relative importance of each scenario to the overall equilibrium.

The four scenarios (quadrants) represent different primary election outcomes:

E1: Both Entrants Both parties' "entrant" candidates (sampled from optimized distributions) win their primaries

E2: A Entrant, B Field Party A's entrant wins, Party B's field candidate (sampled from baseline) wins

E3: A Field, B Entrant Party A's field wins, Party B's entrant wins

E4: Both Field Both parties' field candidates win their primaries

The weight of each scenario depends on the primary election probabilities (kappa values), which in turn depend on the voter model.

Value

A list containing:

weights Named vector of four-quadrant weights (sum to 1)

contributions Named vector of contributions to Q* from each quadrant

Q_star Total equilibrium vote share

plot Base R plot (invisible return)

Interpretation

- A dominant E1 contribution suggests entrant vs. entrant matchups are most important for the equilibrium
- Balanced contributions indicate a robust equilibrium across scenarios
- Large E4 contribution suggests field candidates often win primaries

Examples

```
## Not run:
# Run adversarial strategize
result <- strategize(Y = y, W = w, adversarial = TRUE, nSGD = 500)

# Plot quadrant breakdown
breakdown <- plot_quadrant_breakdown(result)
print(breakdown$weights)
print(breakdown$contributions)

## End(Not run)
```

---

print.hessian_analysis

Print method for hessian_analysis objects

Description

Print method for hessian_analysis objects

Usage

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

Arguments

x	A hessian_analysis object from check_hessian_geometry
...	Additional arguments (ignored)

print.strategize_result *Print Method for strategize Results*

Description

Print Method for strategize Results

Usage

```
## S3 method for class 'strategize_result'
print(x, digits = 3, ...)
```

Arguments

x	A strategize result object
digits	Number of digits to display
...	Additional arguments (ignored)

Value

Invisibly returns the input object

strategize *Estimate Optimal (or Adversarial) Stochastic Interventions for Conjoint Experiments*

Description

`strategize` implements the core methods described in the accompanying paper for learning an optimal or adversarial probability distribution over conjoint factor levels. It is specifically designed for forced-choice conjoint settings (e.g., candidate-choice experiments) and can accommodate scenarios in which a single agent optimizes its strategy in isolation, or in which two (potentially adversarial) agents simultaneously optimize against each other.

This function can be used to find the *optimal stochastic intervention* for maximizing an outcome of interest (e.g., vote choice, rating, or utility), possibly subject to a penalty that keeps the learned distribution close to the original design distribution. It can also incorporate institutional rules (e.g., primaries, multiple stages of choice) by specifying additional arguments. Estimation can be done under standard generalized linear modeling assumptions or more advanced approaches. The function returns estimates of the learned distribution and the associated performance quantity ($Q(\boldsymbol{\pi}^*)$) along with optional inference based on the (asymptotic) delta method.

Usage

```
strategize(
  Y,
  W,
  X = NULL,
  lambda,
  varcov_cluster_variable = NULL,
  competing_group_variable_respondent = NULL,
  competing_group_variable_respondent_proportions = NULL,
  competing_group_variable_candidate = NULL,
  competing_group_competition_variable_candidate = NULL,
  pair_id = NULL,
  respondent_id = NULL,
  respondent_task_id = NULL,
  profile_order = NULL,
  p_list = NULL,
  slate_list = NULL,
  K = 1,
  nSGD = 100,
  diff = FALSE,
  adversarial = FALSE,
  adversarial_model_strategy = "four",
  include_stage_interactions = NULL,
  partial_pooling = NULL,
  partial_pooling_strength = 50,
  use_regularization = TRUE,
  force_gaussian = FALSE,
  a_init_sd = 0.001,
  outcome_model_type = "glm",
  neural_mcmc_control = NULL,
  penalty_type = "KL",
  compute_se = FALSE,
  se_method = c("full", "implicit"),
  conda_env = "strategize_env",
  conda_env_required = FALSE,
  conf_level = 0.90,
  nFolds_glm = 3L,
  folds = NULL,
  nMonte_adversarial = 5L,
  primary_pushforward = "mc",
  primary_strength = 1.0,
  primary_n_entrants = 1L,
  primary_n_field = 1L,
  nMonte_Qglm = 100L,
  learning_rate_max = 0.001,
  temperature = 0.5,
  save_outcome_model = FALSE,
  presaved_outcome_model = FALSE,
  outcome_model_key = NULL,
  use_optax = FALSE,
  optim_type = "gd",
  optimism = "extragrad",
```

```
  compute_hessian = TRUE,
  hessian_max_dim = 50L
)
```

Arguments

Y	A numeric or binary vector of observed outcomes, typically in {0,1} for forced-choice conjoint tasks, indicating whether the profile was selected. For instance, Y = 1 if candidate A was chosen over candidate B, and Y = 0 otherwise. The length must match the number of rows in W.
W	A matrix or data frame representing the assigned levels of each factor in a conjoint design (one column per factor). Each row corresponds to a single profile. For forced-choice tasks, a given respondent may have contributed multiple rows if you reshape pairwise choices into long format. If the experiment used multiple factors D, with each factor having L_d levels, W should capture all factor assignments accordingly.
X	An optional matrix or data frame of additional covariates, often respondent-level features (e.g., respondent demographics). If K > 1, X may be used internally to fit multi-cluster or multi-component outcome models, or to allow cluster-specific effect estimation for more granular insights. Defaults to NULL.
lambda	A numeric scalar or vector giving the regularization penalty (e.g., in Kullback-Leibler or L2 sense) used to shrink the learned probability distribution(s) of factor levels toward a baseline distribution p_list. Typically set via either domain knowledge or cross-validation.
varcov_cluster_variable	An optional vector of cluster identifiers (e.g., respondent IDs) used to form a robust variance-covariance estimate of the outcome model. If NULL, the usual IID assumption is made. Defaults to NULL.
competing_group_variable_respondent	Optional variable marking competition group membership of respondents. Particularly relevant in adversarial settings (adversarial = TRUE) or multi-stage electoral settings, e.g., capturing the party of each respondent. Defaults to NULL.
competing_group_variable_respondent_proportions	Optional numeric vector specifying the population proportions of each competing group. If NULL, proportions are estimated from the data. Useful when the sample proportions differ from the target population proportions. Defaults to NULL.
competing_group_variable_candidate	Optional variable marking competition group membership of candidate profiles. Defaults to NULL.
competing_group_competition_variable_candidate	Optional variable indicating whether a candidate profile belongs to the competing group in adversarial settings. Defaults to NULL.
pair_id	A factor or numeric vector identifying the forced-choice pair. If each row of W is a single profile, pair_id groups the rows belonging to the same choice set. Defaults to NULL.
respondent_id, respondent_task_id	Another set of optional identifiers. respondent_id marks each respondent across tasks, while respondent_task_id can define unique IDs for repeated measurements from the same respondent across multiple tasks. Useful for advanced clustering or robust SEs. Defaults to NULL.

profile_order
: If each forced-choice is shown with different ordering (e.g., Candidate A vs. Candidate B), profile_order can label each row accordingly. Helpful for ensuring consistent labeling of reference vs. opposing profiles. Defaults to NULL.

p_list
: An optional list describing the baseline probability distribution over factor levels in W. Typically derived from the initial design distribution or uniform assignment distribution. If NULL, the function may assume uniform or attempt to estimate the distribution from W.

slate_list
: An optional list (or lists) providing custom slates of candidate features (and their associated probabilities). Used in more advanced or adversarial setups where certain combinations must be included or excluded. If NULL, no special constraints beyond the usual factor-level distributions are applied.

K
: Integer specifying the number of latent clusters for multi-component outcome models. If K = 1, no latent clustering is done. Defaults to 1.

nSGD
: Integer specifying the number of stochastic gradient descent (or gradient-based) iterations to use when learning the optimal distributions. Defaults to 100.

diff
: Logical indicating whether the outcome Y represents a first-difference or difference-based metric. In forced-choice contexts, typically diff = FALSE. Defaults to FALSE.

adversarial
: Logical controlling whether to enable the max-min adversarial scenario. When TRUE, the function searches for a pair of distributions (one for each competing party or group) such that each party's distribution is optimal given the other party's distribution. Defaults to FALSE.

adversarial_model_strategy
: Character string indicating whether to estimate "four" outcome models (primary + general for each group), "two" outcome models (one per group reused for both primary and general), or "neural" (Bayesian Transformer models with party tokens; defaults to a single pooled model across groups and stages. Set neural_mcmc_control$n_bayesian_models = 2 to fit separate AST/DAG models). Defaults to "four".

include_stage_interactions
: Logical indicating whether to include stage (primary vs general) indicator and stage-by-factor interactions in the outcome model. When NULL (default), automatically set to TRUE when adversarial_model_strategy = "two" and FALSE otherwise. Including stage interactions allows a single pooled model to learn different response patterns for primary vs general election scenarios, which helps prevent pattern-matching equilibria where both parties converge to identical strategies.

partial_pooling
: Logical indicating whether to partially pool (shrink) group-specific outcome model coefficients toward a shared average when adversarial_model_strategy = "two". When NULL (default), automatically set to TRUE for the two-strategy adversarial case and FALSE otherwise.

partial_pooling_strength
: Numeric scalar controlling the amount of shrinkage used for partial pooling in the two-strategy adversarial case. Interpreted as a pseudo-sample size: larger values increase pooling, smaller values preserve group differentiation. Defaults to 50.

use_regularization
: Logical indicating whether to regularize the outcome model (in addition to any penalty lambda on the distribution shift). This can help avoid overfitting in high-dimensional designs. Defaults to TRUE.

force_gaussian Logical indicating whether to force a Gaussian-based outcome modeling approach, even if Y is binary or forced-choice. If FALSE, the function attempts to choose a more appropriate link (e.g., "binomial"). Defaults to FALSE.

a_init_sd Numeric scalar specifying the standard deviation for random initialization of unconstrained parameters used in the gradient-based search over factor-level probabilities. Defaults to 0.001.

outcome_model_type
Character string specifying the outcome model to use. Currently supports "glm" for generalized linear models or "neural" for a neural-network approximation. Defaults to "glm".

neural_mcmc_control
Optional list overriding default MCMC settings used when outcome_model_type = "neural". Named entries override the defaults in CS_2Step_ModelOutcome_neural.R. Set neural_mcmc_control$uncertainty_scope = "output" to compute delta-method uncertainty using only the output-layer parameters (default is "all"). In adversarial neural mode, set neural_mcmc_control$n_bayesian_models = 2 to fit separate AST/DAG models (default is 1 for a single differential model). Use neural_mcmc_control$ModelDims and neural_mcmc_control$ModelDepth to override the Transformer hidden width and depth. Set neural_mcmc_control$cross_candidate_ = "term" (or TRUE) to include the opponent-dependent cross-candidate term in pairwise mode, or set neural_mcmc_control$cross_candidate_encoder = "full" to enable a full cross-encoder that jointly encodes both candidates. Use "none" (or FALSE) to disable. For variational inference (subsample_method = "batch_vi"), set neural_mcmc_control$optimizer to "adam" (numpyro.optim) or "adabelief" (optax). Learning-rate decay is controlled by neural_mcmc_control$svi_lr_sche (default "warmup_cosine"), with optional svi_lr_warmup_frac and svi_lr_end_factor.

penalty_type A character string specifying the type of penalty (e.g., "KL", "L2", or "LogMaxProb") used in the objective function for shifting the factor-level probabilities away from the baseline p_list. Defaults to "KL".

compute_se Logical indicating whether standard errors should be computed for the final estimates (via the delta method or related expansions). Defaults to FALSE.

se_method Character string specifying the SE computation method when compute_se = TRUE. "full" differentiates through the full optimization trace (default). "implicit" uses implicit differentiation at the solution (adversarial equilibrium or non-adversarial optimum).

conda_env A character string naming a Python conda environment that includes **jax**, **optax**, and other dependencies. If not NULL, the function attempts to activate that environment. Defaults to "strategize_env".

conda_env_required
Logical; if TRUE, raises an error if the environment given by conda_env cannot be activated. Otherwise, the function attempts to proceed with any available installation. Defaults to FALSE.

conf_level Numeric in $(0, 1)$, specifying the confidence level for intervals or credible bounds. Defaults to 0.90.

nFolds_glm Integer specifying the number of folds (default 3L) for internal cross-validation used in certain outcome model or regularization steps. Defaults to 3L.

folds An optional user-supplied partitioning or CV scheme, overriding nFolds_glm. Defaults to NULL.

nMonte_adversarial
: Integer specifying the number of Monte Carlo samples used in adversarial or max-min steps, e.g., sampling from the opposing candidate's distribution to approximate expected payoffs. Defaults to 5L.

primary_pushforward
: Character string controlling the primary-stage push-forward estimator. Use "mc" (default) for Monte Carlo sampling with per-draw primary winners, or "linearized" for the faster averaged-weight approximation, or "multi" for multi-candidate primaries.

primary_strength
: Numeric scalar controlling primary decisiveness. Values > 1 make primary outcomes more deterministic; values in (0, 1) make primaries more noisy. Defaults to 1.0 (neutral scaling).

primary_n_entrants
: Integer number of entrant candidates sampled per party in multi-candidate primaries (primary_pushforward = "multi"). Defaults to 1.

primary_n_field
: Integer number of field candidates sampled per party in multi-candidate primaries (primary_pushforward = "multi"). Defaults to 1.

nMonte_Qglm
: Integer specifying the number of Monte Carlo samples for evaluating or approximating the quantity of interest under certain outcomes or distributions. Defaults to 100L.

learning_rate_max
: Base learning rate for gradient-based optimizers. Defaults to 0.001.

temperature
: Numeric temperature parameter used in Gumbel-Softmax sampling to smooth the exploration of the probability simplex. Smaller values yield distributions closer to the argmax. Defaults to 0.5.

save_outcome_model
: Logical indicating whether to save the fitted outcome model to disk for reuse. Useful for large models or repeated runs. Defaults to FALSE.

presaved_outcome_model
: Logical indicating whether to use a previously saved outcome model instead of re-fitting. Defaults to FALSE.

outcome_model_key
: Optional character string to append to saved outcome model filenames. Useful for distinguishing between different model configurations or experimental runs. When provided, files are saved as main_{group}_{round}_{key}.csv. Defaults to NULL.

use_optax
: Logical indicating whether to use the optax library for gradient-based optimization in JAX (TRUE) or a built-in method (FALSE). Defaults to FALSE.

optim_type
: A character string for choosing which optimizer or approach is used internally (e.g., "gd" for gradient descent). Defaults to "gd".

optimism
: Character string controlling optimistic / extra-gradient updates for the gradient optimizer. Options: "extragrad" (default; classic Korpelevich extra-gradient), "smp" (stochastic mirror-prox: extra-gradient with weighted averaging of look-ahead points), "ogda" (optimistic gradient), or "none" (standard updates). Only supported when use_optax = FALSE.

compute_hessian
: Logical. Whether to compute Hessian functions for equilibrium geometry analysis in adversarial mode. When TRUE (default), Hessian functions are JIT-

compiled to enable check_hessian_geometry analysis. Set to FALSE to skip Hessian computation for faster execution.

hessian_max_dim

Integer. Maximum number of parameters per player before automatically skipping Hessian computation. Defaults to 50L. For problems with more parameters, Hessian computation is skipped to avoid memory/time overhead. The result will have hessian_skipped_reason = "high_dimension" in this case.

Details

Modeling the outcome: Internally, strategize may fit a generalized linear model or a more flexible approach (such as multi-cluster factorization) to learn the mapping from factor-level assignments W (and optional covariates X) onto outcomes Y. Once these outcome coefficients are estimated, the function uses gradient-based or closed-form solutions to find the *optimal stochastic intervention(s)*, i.e., new factor-level probability distributions that maximize an expected outcome (or solve the max-min adversarial problem).

Adversarial or strategic design: When adversarial = TRUE, the function attempts to solve a zero-sum game in which one agent (say, "A") chooses its distribution to maximize vote share, while the other ("B") simultaneously chooses its distribution to minimize "A"'s vote share. In many settings, competing_group_variable_respondent and related arguments help define which respondents belong to the "A" or "B" sub-electorate (e.g., a primary). The final solution is a mixed-strategy Nash equilibrium, if it exists, for the forced-choice environment. This can be used to compare or interpret real-world candidate positioning in multi-stage elections.

Regularization: The argument lambda penalizes how far the learned distribution strays from the baseline distribution p_list. This helps avoid overfitting in high-dimensional designs. Different penalty types can be selected via penalty_type.

Implementation details: Under the hood, this function may rely on **jax** for automatic differentiation. By default, it uses an internal gradient-based approach. If use_optax = TRUE, the optax library is used for optimization. The function can automatically detect or load a **conda** environment if specified, though advanced users can pass conda_env_required = TRUE to enforce that environment activation is mandatory.

Value

A named list containing:

pi_star_point An estimate of the (possibly multi-cluster or adversarial) optimal distribution(s) over the factor levels.

Structure depends on parameters:

- If adversarial = TRUE and K = 1, returns a pair of distributions (e.g., maximin solutions).
- If K > 1, returns a list where each element corresponds to a cluster-optimal distribution.
- Otherwise, returns a single distribution.

pi_star_se Standard errors for entries in pi_star_point. Mirrors the structure of pi_star_point (e.g., a pair of SEs if adversarial = TRUE and K = 1). Only present if compute_se = TRUE.

Q_point_mEst Point estimate(s) of the optimized outcome (e.g., utility/vote share). Matches the structure of pi_star_point.

Q_se_mEst Standard errors for Q_point_mEst. Only present if compute_se = TRUE.

pi_star_lb, pi_star_ub Confidence bounds for pi_star_point (if compute_se = TRUE and a confidence level is provided).

`outcome_model_view` Interpretable summaries of the fitted outcome models (by player and stage). Includes main-effect tables and top interactions for AST/DAG primary/general submodels when available.

`CVInfo` Cross-validation performance data (if applicable). Typically a `data.frame` or list.

`estimationType` String indicating the approach used (e.g., `"TwoStep"` or `"OneStep"`).

`...` Additional internal details (e.g., fitted models, optimization logs).

See Also

`cv_strategize` for cross-validation across candidate values of `lambda`. See also `strategize_onestep` for a function that implements a "one-step" approach to M-estimation of the same target quantity.

Examples

```
# ============================================
# Example 1: Basic single-agent optimization
# ============================================
# Generate synthetic conjoint data
set.seed(42)
n <- 400  # Number of profiles (200 pairs)

# Factor matrix: candidate attributes
W <- data.frame(
  Gender = sample(c("Male", "Female"), n, replace = TRUE),
  Age = sample(c("Young", "Middle", "Old"), n, replace = TRUE),
  Party = sample(c("Dem", "Rep"), n, replace = TRUE)
)

# Simulate outcome: Female + Young candidates preferred
latent <- 0.3 * (W$Gender == "Female") +
          0.2 * (W$Age == "Young") -
          0.1 * (W$Age == "Old")
prob <- plogis(latent)

# Paired forced-choice: within each pair, one wins
pair_id <- rep(1:(n/2), each = 2)
Y <- numeric(n)
for (p in unique(pair_id)) {
  idx <- which(pair_id == p)
  winner <- sample(idx, 1, prob = prob[idx])
  Y[idx] <- as.integer(seq_along(idx) == which(idx == winner))
}
profile_order <- rep(1:2, n/2)

# Baseline probabilities (uniform assignment)
p_list <- list(
  Gender = c(Male = 0.5, Female = 0.5),
  Age = c(Young = 1/3, Middle = 1/3, Old = 1/3),
  Party = c(Dem = 0.5, Rep = 0.5)
)

# Run strategize to find optimal distribution
# (requires conda environment with JAX - see build_backend())
result <- strategize(
  Y = Y,
  W = W,
```

```
  lambda = 0.1,
  pair_id = pair_id,
  respondent_id = pair_id,
  respondent_task_id = pair_id,
  profile_order = profile_order,
  p_list = p_list,
  diff = TRUE,
  nSGD = 50,
  compute_se = FALSE
)

# View optimal distribution
print(result$pi_star_point)

# View expected outcome under optimal strategy
print(result$Q_point)
```

`strategize.plot` *Plot Estimated Probabilities for Hypothetical Scenarios*

Description

This function creates a grid of base R plots to visualize and compare probabilities (and optionally their confidence intervals) across multiple hypothetical or assignment scenarios. By default, it arranges the plots in a 3xN grid, labeling each row according to the factor or condition being displayed.

Usage

```
strategize.plot(
  pi_star_list = NULL,
  pi_star_se_list = NULL,
  p_list = NULL,
  col.main = "black",
  cex.main = 1.5,
  zStar = 1,
  xlim = NULL,
  ticks_type = "assignmentProbs",
  col_vec = NULL,
  plot_names = TRUE,
  plot_ci = TRUE,
  widths_vec,
  heights_vec,
  main_title = "",
  margins_vec = NULL,
  add = FALSE,
  pch = 20,
  factor_name_transformer = function(x) {
     x
 },
```

```
  level_name_transformer = function(x) {
     x
 },
  open_browser = FALSE
)
```

Arguments

pi_star_list	A list of numeric vectors, each corresponding to a set of hypothetical probabilities to be plotted. These are typically model-based or derived values.
pi_star_se_list	A list of numeric vectors of the same structure as pi_star_list, containing standard errors for each probability. Used to plot confidence intervals.
p_list	A list of numeric vectors of "assignment" or baseline probabilities to be overlaid as vertical ticks on each plot (depending on ticks_type).
col.main	Character. Color for the main title in each subplot. Default is "black".
cex.main	Numeric. Character expansion factor for main titles. Default is 1.5.
zStar	Numeric. Multiplier for the standard error bars (e.g., 1.96 for approximately 95 percent confidence intervals). Default is 1.
xlim	Numeric vector of length 2. The x-axis limits for all subplots. Defaults to c(0, 1) if not specified.
ticks_type	Character. Controls the type of reference ticks added. "assignmentProbs": Vertical ticks drawn at the positions from p_list (default). "zero": Vertical ticks drawn at 0. "none": No vertical reference ticks.
col_vec	Optional character vector of colors (one per set of probabilities in pi_star_list). If NULL, uses sequential indexing for color.
plot_names	Logical. If TRUE (default), factor/condition labels will be placed along the y-axis.
plot_ci	Logical. If TRUE (default), error bars will be drawn using pi_star_se_list.
widths_vec, heights_vec	Currently unused. Reserved for future layout expansions.
main_title	Character. An overall title for the plot. Default is an empty string.
margins_vec	Currently unused. Reserved for future layout expansions.
add	Logical. If FALSE (default), a new plot is created. If TRUE, points/error bars are added to an existing plot space.
pch	Numeric or character. The plotting symbol. Default is 20.
factor_name_transformer	Function to transform factor names for display. Should accept and return a character vector. Default is identity function.
level_name_transformer	Function to transform level names for display. Should accept and return a character vector. Default is identity function.
open_browser	Logical. If TRUE, opens a browser for debugging. Default is FALSE. Intended for development use only.

Details

strategize.plot arranges multiple subplots (3 columns by default) in a grid that depends on the number of elements in p_list. Each subplot will show a factor level or condition on the y-axis, with probabilities along the x-axis. If confidence intervals are provided (pi_star_se_list), horizontal error bars around each probability point will be displayed. Additionally, vertical reference ticks can be added, showing values from p_list or zero depending on ticks_type.

Value

Invisibly returns NULL. This function is primarily called for its side effect: producing a multi-panel base R plot.

Examples

```
# ============================================
# Visualize optimal vs baseline distributions
# ============================================
# This function works without JAX - just needs the result structure

# Create mock strategize result for plotting
# (In practice, use output from strategize())
pi_star_list <- list(k1 = list(
  Gender = c(Male = 0.35, Female = 0.65),
  Age = c(Young = 0.45, Middle = 0.30, Old = 0.25),
  Party = c(Dem = 0.40, Rep = 0.60)
))

pi_star_se_list <- list(k1 = list(
  Gender = c(Male = 0.04, Female = 0.04),
  Age = c(Young = 0.03, Middle = 0.03, Old = 0.03),
  Party = c(Dem = 0.05, Rep = 0.05)
))

# Baseline (original assignment) probabilities
p_list <- list(
  Gender = c(Male = 0.5, Female = 0.5),
  Age = c(Young = 0.33, Middle = 0.33, Old = 0.34),
  Party = c(Dem = 0.5, Rep = 0.5)
)

# Plot comparing optimal to baseline
strategize.plot(
  pi_star_list = pi_star_list,
  pi_star_se_list = pi_star_se_list,
  p_list = p_list,
  main_title = "Optimal vs Baseline Distribution",
  ticks_type = "assignmentProbs"  # Show baseline as reference ticks
)
```

---

strategize_onestep *Estimate an Optimal (or Adversarial) Stochastic Intervention for Conjoint Analysis Using a One-Step M-estimation Approach*

Description

strategize_onestep implements a single-step ("one-step") approach to estimating a target quantity of interest in high-dimensional conjoint (or factorial) experiments, such as finding an *optimal stochastic intervention* over factor levels. This method can incorporate adversarial or non-adversarial settings, regularization, multi-stage structures (e.g., primaries followed by a general election), and a variety of user-specified outcome models. It returns estimated distributions of factor levels *(e.g., candidate attributes)* that maximize or minimize an outcome (e.g., vote share), including optional standard errors via M-estimation or the delta method.

Usage

```
strategize_onestep(
  W,
  Y,
  X = NULL,
  K = 1,
  warm_start = FALSE,
  automatic_scaling = TRUE,
  p_list = NULL,
  pi_list = NULL,
  pi_init_vec = NULL,
  constrain_ub = NULL,
  n_lambda = 10,
  penalty_type = "LogMaxProb",
  test_fraction = 0.5,
  log_PrW = NULL,
  learning_rate_max = 0.01,
  cycle_width  = 50,
  cycle_number = 4,
  nSGD = 500,
  nEpoch = NULL,
  X_factorized = NULL,
  momentum = 0.99,
  n_full_cycles = 1,
  optim_method = "tf",
  sg_method = NULL,
  forceSEs = FALSE,
  clip_at = 100000,
  adaptive_momentum = FALSE,
  known_norm_factor = NULL,
  split1_indices = NULL,
  split2_indices = NULL,
  use_hajek = TRUE,
  find_max = TRUE,
  quiet = TRUE,
  lambda_seq = NULL,
  lambda_coef = 0.0001,
  n_folds = 3,
  batch_size = 50,
  confLevel = 0.90,
  conda_env = NULL,
  conda_env_required = FALSE,
```

```
  hypothetical_n = NULL
)
```

Arguments

W	A matrix or data frame of assigned factor levels in the conjoint. For forced-choice designs, each row generally represents a single profile or a single respondent-task-profile combination. Must have integer, factor, or character columns representing factor levels.
Y	A numeric or binary outcome vector. Typically Y is 1 if a profile is chosen over its competitor, and 0 otherwise. If find_max = FALSE, the sign is flipped, effectively minimizing Y.
X	Optional matrix or data frame of covariates (or respondent-level features). Can be used for multi-cluster modeling with K>1.
K	An integer for the number of mixture components or latent clusters if multi-cluster modeling is desired. Defaults to 1 (no clusters).
warm_start	Logical. If TRUE, attempts to re-initialize from previous solutions each time lambda or other hyperparameters change. Defaults to FALSE.
automatic_scaling	Logical indicating whether to center or scale X and Y automatically. Defaults to TRUE.
p_list	A list of baseline factor-level probabilities in the design or assignment mechanism (e.g., the original random assignment distribution). If NULL, the function may assume uniform or empirical distributions.
pi_list	An optional list specifying a counterfactual distribution over factor levels. If provided, strategize_onestep directly computes and returns the performance or value under that distribution instead of estimating a new optimal distribution.
pi_init_vec	A numeric vector for initializing the simplex-based representation of factor-level probabilities to be optimized. If NULL, a random initialization is used internally.
constrain_ub	Optional numeric or vector of upper bounds on factor probabilities. If not NULL, can help to enforce constraints in optimization.
n_lambda	Integer specifying the number of penalty values considered if cross-validation is performed. Defaults to 10.
penalty_type	A character specifying the type of penalty for shifting probabilities (e.g., "LogMaxProb", "L2", or "KL"). This is an additional penalization on top of penalty_type for the outcome model. Defaults to "LogMaxProb".
test_fraction	Fraction of samples used for holdout in cross-validation. Defaults to 0.5 for a basic split. If NULL, no split is performed.
log_PrW	Optional numeric vector of log probabilities for each row in W. If omitted, the function will compute log_PrW from p_list given the assumption of independent factor-level assignments.
learning_rate_max	Base learning rate for gradient-based optimizers. Defaults to 0.01.
cycle_width	Numeric controlling the frequency of restarts or adaptive learning-rate schedules.
cycle_number	Number of cycles used in the learning-rate schedule.
nSGD	Number of gradient-descent updates. If nEpoch is provided, that takes precedence.

nEpoch Number of epochs, each pass including length(availableTrainIndices) / batch_size mini-batches. If provided, overrides nSGD.

X_factorized An optional matrix or data frame representing factorized (dummy-coded) versions of X for advanced modeling. If NULL, the function may factorize internally.

momentum Numeric specifying momentum for stochastic gradient descent. Defaults to 0.99.

n_full_cycles If >1, repeats training cycles without restarts for a total number of gradient steps. Useful for stability checks.

optim_method A character specifying the optimization backend. Defaults to "jax" if available.

sg_method A character controlling the type of gradient updates (e.g., "adanorm", "wngrad"). If NULL, a default method is chosen.

forceSEs Logical. If TRUE, attempts to compute standard errors by M-estimation or the delta method, even if no cross-validation is done.

clip_at A large numeric to clip gradient norms if they exceed clip_at.

adaptive_momentum
Logical. If TRUE, momentum is adapted automatically as the optimization proceeds. Defaults to FALSE.

known_norm_factor
An optional numeric to normalize reweighting for Hajek-based estimators. If NULL, it is inferred from the sum of weights.

split1_indices, split2_indices
Optional vectors of indices partitioning the data for cross-validation or holdout. If NULL, a random partition is done internally.

use_hajek Logical. If TRUE, uses a Hajek-based reweighting in objective functions for computing the expected outcome under counterfactual probability shifts. Defaults to TRUE.

find_max Logical. If TRUE, maximizes Y; if FALSE, treats Y as negative of interest (e.g., adversity minimization).

quiet Logical controlling the verbosity of printed messages.

lambda_seq Optional numeric vector of penalty values for cross-validation. If NULL, the function attempts a default sequence or single value.

lambda_coef Numeric constant controlling the magnitude of the penalty for the outcome model. Defaults to 0.0001.

n_folds Number of folds for cross-validation. Defaults to 3.

batch_size Positive integer specifying the size of mini-batches in each gradient iteration. Defaults to 50.

confLevel Numeric in \((0,1)\), specifying the confidence level for intervals around estimated probabilities or performance measures. Defaults to 0.90.

conda_env Optional name of a Conda environment with **jax**, **optax**, etc. If NULL, attempts a default environment.

conda_env_required
Logical. If TRUE, errors if the environment conda_env cannot be found. Otherwise attempts to proceed gracefully.

hypothetical_n Optional numeric specifying an alternative n for certain variance calculations (e.g., hypothesized population size). If NULL, uses the observed sample size.

Details

This function implements a *one-step M-estimation* approach for directly estimating the "optimal" probability distributions over high-dimensional factors in conjoint or factorial experiments. Rather than a multi-step procedure of (1) outcome modeling followed by (2) re-optimizing factor distributions, the one-step approach can iteratively re-estimate distribution parameters while simultaneously adjusting the outcome model. This allows regularization or advanced modeling to be integrated into the *same* optimization objective, potentially improving finite-sample performance. Support for adversarial or multiple clusters is also available.

By default, `strategize_onestep` attempts to find the distribution(s) $\boldsymbol{\pi}^*$ that maximizes the average outcome if `find_max = TRUE` (e.g., maximizing candidate choice share). In adversarial contexts, each cluster or "player" can simultaneously learn a best response. The function is flexible enough to incorporate sub-populations or multiple stages (e.g., primaries plus general elections).

If a user-supplied `pi_list` is given, the function directly computes $Q(\boldsymbol{\pi})$ for that distribution instead of estimating. This is useful for evaluating the performance of a known or hypothesized distribution (e.g., "status quo").

Most users do not need to call `strategize_onestep` directly, as this is a lower-level routine. The `strategize` or `cv_strategize` functions may suffice in many typical workflows.

Value

A named `list` with components, often including:

- `pi_star_point`: The estimated optimal or learned distribution(s) over factor levels. If `K>1` or if adversarial competition is considered, may return multiple distributions (`k1`, `k2`, etc.).
- `Q_point`: The estimated performance measure under the learned distribution(s). For example, the average or adversarially optimized outcome.
- `Q_se_mEst`: If available, standard errors via M-estimation or the delta method.
- `PiStar_lb`, `PiStar_ub`: Lower and upper confidence intervals for factor-level probabilities, if standard errors are computed.
- `CVInfo`: A data frame or list summarizing cross-validation performance for each candidate `lambda`.
- `ClassProbsXobs`, `VarCov_ProbClust`, `pi_init_next`, `optim_max_hajek_list`: Additional objects storing advanced details of the optimization or M-estimation procedure.
- `Output.Description`: Additional messages describing the run.

Note

Advanced arguments like `X_factorized`, `conda_env`, `optim_method`, or specifying `adaptive_momentum` are only needed for specialized or larger-scale (GPU-based) computations.

References

- Goplerud, M. & Titiunik, R. (2022). *Analysis of High-Dimensional Factorial Experiments: Estimation of Interactive and Non-Interactive Effects.* ArXiv preprint. - Egami, N. & Imai, K. (2019). *Causal Interaction in Factorial Experiments: Application to Conjoint Analysis.* Journal of the American Statistical Association, 114(526), 529–540. - Hainmueller, J., Hopkins, D. J., & Yamamoto, T. (2014). *Causal Inference in Conjoint Analysis: Understanding Multidimensional Choices via Stated Preference Experiments.* Political Analysis, 22(1), 1–30. - (Paper Reference) A forthcoming or accompanying manuscript describing in detail the methods for *optimal* or *adversarial* stochastic interventions in conjoint settings.

See Also

strategize for an approach that first fits an outcome model and then re-optimizes factor-level probabilities. cv_strategize for cross-validation across candidate values of lambda.

Examples

```
## Not run:
# ================================================
# One-step M-estimation approach
# ================================================
# This approach simultaneously estimates the outcome model and
# optimal distribution, rather than the two-step approach

set.seed(123)
n <- 400

# Generate factor matrix
W <- data.frame(
  Gender = sample(c("Male", "Female"), n, replace = TRUE),
  Age = sample(c("Young", "Middle", "Old"), n, replace = TRUE),
  Party = sample(c("Dem", "Rep"), n, replace = TRUE)
)

# Simulate outcome (Female and Young preferred)
latent <- 0.3 * (W$Gender == "Female") + 0.2 * (W$Age == "Young")
Y <- rbinom(n, 1, plogis(latent))

# Baseline probabilities (uniform)
p_list <- list(
  Gender = c(Male = 0.5, Female = 0.5),
  Age = c(Young = 1/3, Middle = 1/3, Old = 1/3),
  Party = c(Dem = 0.5, Rep = 0.5)
)

# Optional respondent covariates
X <- matrix(rnorm(n * 2), n, 2)
colnames(X) <- c("Income", "Education")

# Run one-step estimation
result <- strategize_onestep(
  W = W,
  Y = Y,
  X = X,
  p_list = p_list,
  nSGD = 100,
  penalty_type = "LogMaxProb",
  lambda_seq = c(0.01, 0.1),
  test_fraction = 0.3,
  quiet = TRUE
)

# View optimal distribution
print(result$pi_star_point)

# View expected outcome under optimal strategy
print(result$Q_point)
```

```
## End(Not run)
```

`strategize_preset` *Get Recommended Parameter Settings*

Description

Returns a list of parameter values suitable for passing to `strategize()`. This simplifies the API by providing sensible defaults for common use cases.

Usage

```
strategize_preset(
  preset = c("standard", "quick_test", "publication", "adversarial")
)
```

Arguments

`preset`	One of:
	`"quick_test"` Minimal iterations for testing code (not for inference)
	`"standard"` Reasonable defaults for most analyses (default)
	`"publication"` Higher accuracy for publication-quality results
	`"adversarial"` Settings tuned for adversarial/game-theoretic mode

Details

These presets provide starting points. You should still set data-specific parameters like `Y`, `W`, `p_list`, and potentially `lambda`.

For the "publication" preset, `lambda` is not included because you should use `cv_strategize()` to select it via cross-validation.

Value

Named list of parameters that can be merged with `strategize()` arguments.

Examples

```
# Get standard settings
params <- strategize_preset("standard")
print(params)

# Use with strategize (hypothetically)
# result <- do.call(strategize, c(list(Y = Y, W = W, p_list = p_list), params))
```

summarize_adversarial *Summarize Adversarial Strategize Results*

Description

Prints a comprehensive summary of adversarial equilibrium results, including strategies, equilibrium quality metrics, and four-quadrant breakdown.

Usage

```
summarize_adversarial(
  result,
  validate = TRUE,
  check_hessian = TRUE,
  verbose = TRUE
)
```

Arguments

result	Output from strategize with adversarial = TRUE
validate	Logical. Whether to run equilibrium validation. Default is TRUE. Set to FALSE for faster output without validation.
check_hessian	Logical. Whether to include Hessian geometry analysis. Default is TRUE. Only runs if Hessian functions are available in the result.
verbose	Logical. Whether to print the summary. Default is TRUE.

Details

This function provides a unified view of adversarial equilibrium results:

- Equilibrium vote share Q* with standard error
- Optimized strategies for both parties (AST and DAG)
- Equilibrium quality metrics (best-response errors)
- Four-quadrant contribution breakdown
- Voter proportion information
- Convergence status

Value

Invisibly returns a list containing all summary statistics.

Examples

```
## Not run:
# Run adversarial strategize
result <- strategize(Y = y, W = w, adversarial = TRUE, nSGD = 500)

# Print summary
summarize_adversarial(result)
```

```
# Quick summary without validation
summarize_adversarial(result, validate = FALSE)

## End(Not run)
```

---

summary.strategize_result *Summary Method for strategize Results*

Description

Creates a summary table comparing baseline and optimal distributions.

Usage

```
## S3 method for class 'strategize_result'
summary(object, ...)
```

Arguments

object	A strategize result object
...	Additional arguments (ignored)

Value

Invisibly returns a data.frame with the comparison

---

validate_equilibrium *Validate Nash Equilibrium Quality*

Description

Computes best-response error to verify that the optimized strategies form a Nash equilibrium. At a true Nash equilibrium, neither player can improve their payoff by unilaterally changing strategy.

Usage

```
validate_equilibrium(
  result,
  method = c("grid", "gradient"),
  resolution = 50,
  tolerance = 0.01,
  nMonte = 100,
  plot = TRUE,
  verbose = TRUE,
  seed = 1L
)
```

Arguments

result	Output from strategize with adversarial = TRUE
method	Character string specifying the search method: • "grid": Grid search around the current solution (deterministic for 1-2 parameters; random sampling for higher dimensions) • "gradient": Gradient ascent from current solution (faster) Default is "grid".
resolution	Integer. Number of grid points per dimension for grid search. Default is 50.
tolerance	Numeric. Maximum BR error to consider as equilibrium. Default is 0.01 (i.e., neither player can improve vote share by more than 1%).
nMonte	Integer. Number of Monte Carlo samples for Q evaluation. This overrides the Monte Carlo settings stored in result for the duration of this validation call. Default is 100.
plot	Logical. Whether to generate visualization. Default is TRUE.
verbose	Logical. Whether to print progress messages. Default is TRUE.
seed	Integer seed for reproducible Monte Carlo evaluation and deterministic search behavior. Use NULL for non-deterministic behavior. Default is 1.

Details

What is a Nash Equilibrium?

In the adversarial setting, two parties (AST and DAG) simultaneously choose probability distributions over candidate attributes (e.g., gender, age, policy positions). Each party wants to maximize their expected vote share given the opponent's strategy. A Nash equilibrium is a pair of strategies where:

- AST's strategy is optimal given DAG's strategy
- DAG's strategy is optimal given AST's strategy

At equilibrium, neither party can improve by unilaterally changing their strategy.

What is Best-Response Error?

The best-response error measures how far a player's current strategy is from being optimal. For player p, it is defined as:

$$BR\_error_p = \max_{\pi_p} Q(\pi_p, \pi^*_{-p}) - Q(\pi^*_p, \pi^*_{-p})$$

In words: if we fix the opponent's strategy and search for the best possible response, how much better could we do compared to our current strategy?

- **BR error = 0**: The player is already playing optimally (true equilibrium)
- **BR error > 0**: The player could improve by switching strategies
- **BR error = 0.05**: The player could gain 5 percentage points in vote share

How Validation Works

This function performs the following steps:

1. Evaluates Q (expected vote share) at the current solution
2. For AST: fixes DAG's strategy and searches for AST's best response

3. For DAG: fixes AST's strategy and searches for DAG's best response
4. Computes the improvement each player could achieve (BR error)
5. If both BR errors are below tolerance, declares it a valid equilibrium

Interpretation

- `is_equilibrium = TRUE`: The solution is a valid Nash equilibrium (within numerical tolerance). Both parties are playing optimally.
- `is_equilibrium = FALSE`: At least one party could improve by changing strategy. This may indicate insufficient SGD iterations, a local minimum, or numerical issues.

Search Methods

- `"grid"`: Searches over a discretized grid of strategies around the current solution. More thorough but slower. Recommended for validation.
- `"gradient"`: Runs additional gradient ascent steps from the current solution. Faster but may miss improvements in other directions.

Value

A list containing:

br_error_ast Best-response error for AST player (how much AST could improve by switching to best response)

br_error_dag Best-response error for DAG player

is_equilibrium Logical. TRUE if both errors are below tolerance

Q_current Current objective value at the solution

Q_br_ast Objective value if AST switched to best response

Q_br_dag Objective value if DAG switched to best response

br_strategy_ast The best response strategy for AST (for comparison)

br_strategy_dag The best response strategy for DAG

nMonte Monte Carlo sample count used for validation

seed Seed used for deterministic evaluation (if provided)

plot ggplot object if plot=TRUE and ggplot2 available, else NULL

Examples

```
## Not run:
# Run adversarial strategize
result <- strategize(Y = y, W = w, adversarial = TRUE, nSGD = 500)

# Validate equilibrium
validation <- validate_equilibrium(result)
print(validation$is_equilibrium)
print(validation$br_error_ast)
print(validation$br_error_dag)

# If validation fails, try more SGD iterations
if (!validation$is_equilibrium) {
  result2 <- strategize(Y = y, W = w, adversarial = TRUE, nSGD = 2000)
  validation2 <- validate_equilibrium(result2)
```

```
}

## End(Not run)
```

Index

build_backend, 2

check_hessian_geometry, 3, *18*, *25*
create_p_list, 5
cv_strategize, 6, *14*, *26*, *33*, *34*

get_final_gradients, 11

helpers, 12

is_adversarial, 12

plot_best_response_curves, 13
plot_convergence, 15
plot_quadrant_breakdown, 17
print.hessian_analysis, 18
print.strategize_result, 19

strategize, *3*, *6*, *7*, *10*, *12–14*, *16*, *17*, 19, *33*, *34*, *36*, *38*
strategize.plot, 27
strategize_onestep, *14*, *26*, 29
strategize_preset, 35
summarize_adversarial, 36
summary.strategize_result, 37

validate_equilibrium, *4*, 37