Package ‘cramR’

May 14, 2025

Type Package

Title Cram Method for Efficient Simultaneous Learning and Evaluation

Version 0.1.0

Date 2025-05-11

Maintainer Yanis Vandecasteele <yanisvdc.ensae@gmail.com>

Description Performs the Cram method, a general and efficient approach to simultaneous learning and evaluation using a generic machine learning algorithm. In a single pass of batched data, the proposed method repeatedly trains a machine learning algorithm and tests its empirical performance. Because it utilizes the entire sample for both learning and evaluation, cramming is significantly more data-efficient than sample-splitting. Unlike cross-validation, Cram evaluates the final learned model directly, providing sharper inference aligned with real-world deployment. The method naturally applies to both policy learning and contextual bandits, where decisions are based on individual features to maximize outcomes. The package includes cram_policy() for learning and evaluating individualized binary treatment rules, cram_ml() to train and assess the population-level performance of machine learning models, and cram_bandit() for on-policy evaluation of contextual bandit algorithms. For all three functions, the package provides estimates of the average outcome that would result if the model were deployed, along with standard errors and confidence intervals for these estimates. Details of the method are described in Jia, Imai, and Li (2024) <https://www.hbs.edu/ris/Publication%20Files/2403.07031v1_a83462e0-145b-4675-99d5-9754aa65d786.pdf> and Jia et al. (2025) <doi:10.48550/arXiv.2403.07031>.

License GPL-3

URL https://github.com/yanisvdc/cramR,
https://yanisvdc.github.io/cramR/

BugReports https://github.com/yanisvdc/cramR/issues

Depends R (>= 3.5.0)

Imports caret (>= 7.0-1), grf (>= 2.4.0), glmnet (>= 4.1.8), stats (>= 4.3.3), magrittr (>= 2.0.3), doParallel (>= 1.0.17), foreach (>= 1.5.2), DT (>= 0.33), data.table (>= 1.16.4), keras (>= 2.15.0), dplyr (>= 1.1.4), purrr, R6, rjson, R.devices, itertools, iterators

Suggests testthat (>= 3.0.0), covr (>= 3.5.1), kableExtra (>= 1.4.0), profvis (>= 0.4.0), devtools, waldo, knitr, rmarkdown, randomForest, gbm, nnet, withr

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Config/Needs/citation yes

NeedsCompilation no

Author Yanis Vandecasteele [cre, aut],
Michael Lingzhi Li [ctb],
Kosuke Imai [ctb],
Zeyang Jia [ctb],
Longlin Wang [ctb]

Repository CRAN

Date/Publication 2025-05-14 08:00:02 UTC

Contents

BatchContextualEpsilonGreedyPolicy . . . 3
BatchContextualLinTSPolicy . . . 5
BatchLinUCBDisjointPolicyEpsilon . . . 7
ContextualLinearBandit . . . 9
cram_bandit . . . 11
cram_bandit_est . . . 12
cram_bandit_sim . . . 13
cram_bandit_var . . . 15
cram_estimator . . . 15
cram_expected_loss . . . 16
cram_learning . . . 17
cram_ml . . . 18
cram_policy . . . 20
cram_policy_value_estimator . . . 22
cram_simulation . . . 23
cram_variance_estimator . . . 26
cram_variance_estimator_policy_value . . . 27
cram_var_expected_loss . . . 28
fit_model . . . 28
fit_model_ml . . . 29
get_betas . . . 29
LinUCBDisjointPolicyEpsilon . . . 30
ml_learning . . . 32
model_predict . . . 33
model_predict_ml . . . 34
set_model . . . 34

test_baseline_policy 35
test_batch 36
validate_params 37
validate_params_fnn 37

Index **39**

`BatchContextualEpsilonGreedyPolicy`
Batch Contextual Epsilon-Greedy Policy

Description

Batch Contextual Epsilon-Greedy Policy

Batch Contextual Epsilon-Greedy Policy

Details

Implements an epsilon-greedy exploration strategy for contextual bandits with batched updates.

Super class

`cramR::NA`

Public fields

`epsilon` Probability of selecting a random arm (exploration rate).

`batch_size` Number of rounds per batch before updating model parameters.

`A_cc` List of Gram matrices (one per arm), used to accumulate sufficient statistics across batches.

`b_cc` List of reward-weighted context sums (one per arm), updated batch-wise.

`class_name` Internal class name identifier.

Methods

Public methods:

- `BatchContextualEpsilonGreedyPolicy$new()`
- `BatchContextualEpsilonGreedyPolicy$set_parameters()`
- `BatchContextualEpsilonGreedyPolicy$get_action()`
- `BatchContextualEpsilonGreedyPolicy$set_reward()`
- `BatchContextualEpsilonGreedyPolicy$clone()`

Method `new()`**:** Constructor for the Batch Epsilon-Greedy policy.

Usage:

```
BatchContextualEpsilonGreedyPolicy$new(epsilon = 0.1, batch_size = 1)
```

Arguments:

`epsilon` Numeric between 0 and 1. Probability of random arm selection.

`batch_size` Integer. Number of observations between parameter updates.

Method `set_parameters():` Initializes the parameter structures for each arm.

Usage:

```
BatchContextualEpsilonGreedyPolicy$set_parameters(context_params)
```

Arguments:

`context_params` A list with at least ‘d‘ (number of features) and ‘k‘ (number of arms).

Method `get_action():` Chooses an arm based on epsilon-greedy logic and the current estimates.

Usage:

```
BatchContextualEpsilonGreedyPolicy$get_action(t, context)
```

Arguments:

`t` Integer time step.

`context` A list with contextual features and arm count.

Returns: A list with the selected action.

Method `set_reward():` Updates model statistics based on observed reward. Updates occur once per batch.

Usage:

```
BatchContextualEpsilonGreedyPolicy$set_reward(t, context, action, reward)
```

Arguments:

`t` Integer time step.

`context` List of contextual features used for the action.

`action` A list with the chosen arm.

`reward` A list with the observed reward.

Returns: Updated parameter estimates.

Method `clone():` The objects of this class are cloneable with this method.

Usage:

```
BatchContextualEpsilonGreedyPolicy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`BatchContextualLinTSPolicy` *Batch Contextual Thompson Sampling Policy*

Description

Batch Contextual Thompson Sampling Policy

Batch Contextual Thompson Sampling Policy

Details

Implements Thompson Sampling for linear contextual bandits with batch updates.

Methods

- ‘initialize(v = 0.2, batch_size = 1)‘: Constructor, sets variance and batch size. - ‘set_parameters(context_params)‘: Initializes arm-level matrices. - ‘get_action(t, context)‘: Samples from the posterior and selects action. - ‘set_reward(t, context, action, reward)‘: Updates posterior statistics using observed feedback.

Super class

`cramR::NA`

Public fields

`sigma` Numeric, posterior variance scale parameter.

`batch_size` Integer, size of mini-batches before parameter updates.

`A_cc` List of accumulated Gram matrices per arm.

`b_cc` List of reward-weighted context sums per arm.

`class_name` Internal name of the class.

Methods

Public methods:

- `BatchContextualLinTSPolicy$new()`
- `BatchContextualLinTSPolicy$set_parameters()`
- `BatchContextualLinTSPolicy$get_action()`
- `BatchContextualLinTSPolicy$set_reward()`
- `BatchContextualLinTSPolicy$clone()`

Method `new()`**:** Constructor for the batch-based Thompson Sampling policy.

Usage:

```
BatchContextualLinTSPolicy$new(v = 0.2, batch_size = 1)
```

Arguments:

`v` Numeric. Standard deviation scaling parameter for posterior sampling.

`batch_size` Integer. Number of rounds before parameters are updated.

Method `set_parameters()`**:** Initializes per-arm sufficient statistics.

Usage:

```
BatchContextualLinTSPolicy$set_parameters(context_params)
```

Arguments:

`context_params` List with entries: ‘unique‘ (feature vector), ‘k‘ (number of arms).

Method `get_action()`**:** Samples from the posterior distribution of expected rewards and selects an action.

Usage:

```
BatchContextualLinTSPolicy$get_action(t, context)
```

Arguments:

`t` Integer. Time step.

`context` List containing the current context and arm information.

Returns: A list with the chosen arm (‘choice‘).

Method `set_reward()`**:** Updates Gram matrix and response vector for the chosen arm. Parameters are refreshed every ‘batch_size‘ rounds.

Usage:

```
BatchContextualLinTSPolicy$set_reward(t, context, action, reward)
```

Arguments:

`t` Integer. Time step.

`context` Context object containing feature info.

`action` Chosen action (arm index).

`reward` Observed reward for the action.

Returns: Updated internal parameters.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
BatchContextualLinTSPolicy$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`BatchLinUCBDisjointPolicyEpsilon` *Batch Disjoint LinUCB Policy with Epsilon-Greedy*

Description

Batch Disjoint LinUCB Policy with Epsilon-Greedy

Batch Disjoint LinUCB Policy with Epsilon-Greedy

Details

Implements the disjoint LinUCB algorithm with upper confidence bounds and epsilon-greedy exploration, using batched updates.

Methods

- ‘initialize(alpha = 1.0, epsilon = 0.1, batch_size = 1)‘: Constructor. - ‘set_parameters(context_params)‘: Initializes sufficient statistics for each arm. - ‘get_action(t, context)‘: Selects an arm using UCB scores and epsilon-greedy rule. - ‘set_reward(t, context, action, reward)‘: Updates statistics and refreshes model at batch intervals.

Super class

`cramR::NA`

Public fields

`alpha` Numeric, UCB exploration strength parameter.

`epsilon` Numeric, probability of taking a random exploratory action.

`batch_size` Integer, number of rounds per batch update.

`A_cc` List of Gram matrices per arm, accumulated across batch.

`b_cc` List of reward-weighted context vectors per arm.

`class_name` Internal class name identifier.

Methods

Public methods:

- `BatchLinUCBDisjointPolicyEpsilon$new()`
- `BatchLinUCBDisjointPolicyEpsilon$set_parameters()`
- `BatchLinUCBDisjointPolicyEpsilon$get_action()`
- `BatchLinUCBDisjointPolicyEpsilon$set_reward()`
- `BatchLinUCBDisjointPolicyEpsilon$clone()`

Method `new()`**:** Constructor for batched LinUCB with epsilon-greedy exploration.

Usage:

```
BatchLinUCBDisjointPolicyEpsilon$new(alpha = 1, epsilon = 0.1, batch_size = 1)
```

Arguments:

`alpha` Numeric. UCB width parameter (exploration strength).

`epsilon` Numeric. Probability of selecting a random arm.

`batch_size` Integer. Number of rounds before updating parameters.

Method `set_parameters()`**:** Initialize arm-specific parameter containers.

Usage:

```
BatchLinUCBDisjointPolicyEpsilon$set_parameters(context_params)
```

Arguments:

`context_params` List containing at least ‘unique‘ (feature size) and ‘k‘ (number of arms).

Method `get_action()`**:** Chooses an arm based on UCB and epsilon-greedy sampling.

Usage:

```
BatchLinUCBDisjointPolicyEpsilon$get_action(t, context)
```

Arguments:

`t` Integer timestep.

`context` List containing the context for the decision.

Returns: A list with the selected action.

Method `set_reward()`**:** Updates arm-specific sufficient statistics based on observed reward. Parameter updates occur only at the end of a batch.

Usage:

```
BatchLinUCBDisjointPolicyEpsilon$set_reward(t, context, action, reward)
```

Arguments:

`t` Integer timestep.

`context` Context object used for decision-making.

`action` List containing the chosen action.

`reward` List containing the observed reward.

Returns: Updated internal model parameters.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
BatchLinUCBDisjointPolicyEpsilon$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ContextualLinearBandit` *Contextual Linear Bandit Environment*

Description

Contextual Linear Bandit Environment

Contextual Linear Bandit Environment

Details

An R6 class for simulating a contextual linear bandit environment with normally distributed rewards.

Methods

- ‘initialize(k, d, list_betas, sigma = 0.1, binary_rewards = FALSE)‘: Constructor. - ‘post_initialization()‘: Loads correct coefficients based on ‘sim_id‘. - ‘get_context(t)‘: Returns context and sets internal reward vector. - ‘get_reward(t, context_common, action)‘: Returns observed reward for an action.

Super class

`cramR::NA` -> `ContextualLinearBandit`

Public fields

`rewards` A vector of rewards for each arm in the current round.

`betas` Coefficient matrix of the linear reward model (one column per arm).

`sigma` Standard deviation of the Gaussian noise added to rewards.

`binary` Logical, indicating whether to convert rewards into binary outcomes.

`weights` The latent reward scores before noise and/or binarization.

`list_betas` A list of coefficient matrices, one per simulation.

`sim_id` Index for selecting which simulation’s coefficients to use.

`class_name` Name of the class for internal tracking.

Methods

Public methods:

- `ContextualLinearBandit$new()`
- `ContextualLinearBandit$post_initialization()`
- `ContextualLinearBandit$get_context()`
- `ContextualLinearBandit$get_reward()`
- `ContextualLinearBandit$clone()`

Method `new()`**:**

Usage:

```
ContextualLinearBandit$new(
  k,
  d,
  list_betas,
  sigma = 0.1,
  binary_rewards = FALSE
)
```

Arguments:

`k` Number of arms

`d` Number of features

`list_betas` A list of true beta matrices for each simulation

`sigma` Standard deviation of Gaussian noise

`binary_rewards` Logical, use binary rewards or not

Method `post_initialization()`**:** Set the simulation-specific coefficients for the current simulation.

Usage:

```
ContextualLinearBandit$post_initialization()
```

Returns: No return value; modifies the internal state of the object.

Method `get_context()`**:**

Usage:

```
ContextualLinearBandit$get_context(t)
```

Arguments:

`t` Current time step

Returns: A list containing context vector ‘X‘ and arm count ‘k‘

Method `get_reward()`**:**

Usage:

```
ContextualLinearBandit$get_reward(t, context_common, action)
```

Arguments:

`t` Current time step

`context_common` Context shared across arms

`action` Action taken by the policy

Returns: A list with reward and optimal arm/reward info

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ContextualLinearBandit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`cram_bandit` *Cram Bandit: On-policy Statistical Evaluation in Contextual Bandits*

Description

Performs the Cram method for On-policy Statistical Evaluation in Contextual Bandits

Usage

```
cram_bandit(pi, arm, reward, batch = 1, alpha = 0.05)
```

Arguments

`pi`	An array of shape ($T \times B$, T, K) or ($T \times B$, T), where T is the number of learning steps (or policy updates), B is the batch size, K is the number of arms, T x B is the total number of contexts. If 3D, pi[j, t, a] gives the probability that the policy pi_t assigns arm a to context X_j. If 2D, pi[j, t] gives the probability that the policy pi_t assigns arm A_j (arm actually chosen under X_j in the history) to context X_j. Please see vignette for more details.
`arm`	A vector of length T x B indicating which arm was selected in each context
`reward`	A vector of observed rewards of length T x B
`batch`	(Optional) A vector or integer. If a vector, gives the batch assignment for each context. If an integer, interpreted as the batch size and contexts are assigned to a batch in the order of the dataset. Default is 1.
`alpha`	Significance level for confidence intervals for calculating the empirical coverage. Default is 0.05 (95% confidence).

Value

A list containing:

`raw_results`	A data frame summarizing key metrics: Empirical Bias on Policy Value, Average relative error on Policy Value, RMSE using relative errors on Policy Value, Empirical Coverage of Confidence Intervals.
`interactive_table`	An interactive table summarizing the same key metrics in a user-friendly interface.

Examples

```
# Example with batch size of 1

# Set random seed for reproducibility
set.seed(42)

# Define parameters
T <- 100  # Number of timesteps
```

```
K <- 4    # Number of arms

# Simulate a 3D array pi of shape (T, T, K)
# - First dimension: Individuals (context Xj)
# - Second dimension: Time steps (pi_t)
# - Third dimension: Arms (depth)
pi <- array(runif(T * T * K, 0.1, 1), dim = c(T, T, K))

# Normalize probabilities so that each row sums to 1 across arms
for (t in 1:T) {
  for (j in 1:T) {
    pi[j, t, ] <- pi[j, t, ] / sum(pi[j, t, ])
    }
 }

# Simulate arm selections (randomly choosing an arm)
arm <- sample(1:K, T, replace = TRUE)

# Simulate rewards (assume normally distributed rewards)
reward <- rnorm(T, mean = 1, sd = 0.5)

result <- cram_bandit(pi, arm, reward, batch=1, alpha=0.05)
result$raw_results
result$interactive_table
```

`cram_bandit_est` *Cram Bandit Policy Value Estimate*

Description

This function implements the contextual armed bandit on-policy evaluation by providing the policy value estimate.

Usage

```
cram_bandit_est(pi, reward, arm, batch = 1)
```

Arguments

`pi`	An array of shape (T × B, T, K) or (T × B, T), where T is the number of learning steps (or policy updates), B is the batch size, K is the number of arms, T x B is the total number of contexts. If 3D, pi[j, t, a] gives the probability that the policy pi_t assigns arm a to context X_j. If 2D, pi[j, t] gives the probability that the policy pi_t assigns arm A_j (arm actually chosen under X_j in the history) to context X_j. Please see vignette for more details.
`reward`	A vector of observed rewards of length T x B
`arm`	A vector of length T x B indicating which arm was selected in each context

`batch`	(Optional) A vector or integer. If a vector, gives the batch assignment for each context. If an integer, interpreted as the batch size and contexts are assigned to a batch in the order of the dataset. Default is 1.

Value

The estimated policy value.

`cram_bandit_sim` *Cram Bandit Simulation*

Description

This function runs on-policy simulation for contextual bandit algorithms using the Cram method. It evaluates the statistical properties of policy value estimates.

Usage

```
cram_bandit_sim(
  horizon,
  simulations,
  bandit,
  policy,
  alpha = 0.05,
  do_parallel = FALSE,
  seed = 42
)
```

Arguments

`horizon`	An integer specifying the number of timesteps (rounds) per simulation.
`simulations`	An integer specifying the number of independent Monte Carlo simulations to perform.
`bandit`	A contextual bandit environment object that generates contexts (feature vectors) and observed rewards for each arm chosen.
`policy`	A policy object that takes in a context and selects an arm (action) at each timestep.
`alpha`	Significance level for confidence intervals for calculating the empirical coverage. Default is 0.05 (95% confidence).
`do_parallel`	Whether to parallelize the simulations. Default to FALSE. We recommend keeping to FALSE unless necessary, please see vignette.
`seed`	An optional integer to set the random seed for reproducibility. If NULL, no seed is set.

Value

A list containing:

`estimates`	A table containing the detailed history of estimates and errors for each simulation.
`raw_results`	A data frame summarizing key metrics: Empirical Bias on Policy Value, Average relative error on Policy Value, RMSE using relative errors on Policy Value, Empirical Coverage of Confidence Intervals.
`interactive_table`	An interactive table summarizing the same key metrics in a user-friendly interface.

Examples

```
# Number of time steps
horizon       <- 500L

# Number of simulations
simulations   <- 100L

# Number of arms
k = 4

# Number of context features
d= 3

# Reward beta parameters of linear model (the outcome generation models,
# one for each arm, are linear with arm-specific parameters betas)
list_betas <- cramR::get_betas(simulations, d, k)

# Define the contextual linear bandit, where sigma is the scale
# of the noise in the outcome linear model
bandit        <- cramR::ContextualLinearBandit$new(k = k,
                                                    d = d,
                                                    list_betas = list_betas,
                                                    sigma = 0.3)

# Define the policy object (choose between Contextual Epsilon Greedy,
# UCB Disjoint and Thompson Sampling)
policy <- cramR::BatchContextualEpsilonGreedyPolicy$new(epsilon=0.1,
                                                         batch_size=5)
# policy <- cramR::BatchLinUCBDisjointPolicyEpsilon$new(alpha=1.0,epsilon=0.1,batch_size=1)
# policy <- cramR::BatchContextualLinTSPolicy$new(v = 0.1, batch_size=1)


sim <- cram_bandit_sim(horizon, simulations,
                       bandit, policy,
                       alpha=0.05, do_parallel = FALSE)
sim$summary_table
```

`cram_bandit_var` *Cram Bandit Variance of the Policy Value Estimate*

Description

This function implements the crammed variance estimate of the policy value estimate for the contextual armed bandit on-policy evaluation setting.

Usage

```
cram_bandit_var(pi, reward, arm, batch = 1)
```

Arguments

`pi`	An array of shape (T × B, T, K) or (T × B, T), where T is the number of learning steps (or policy updates), B is the batch size, K is the number of arms, T x B is the total number of contexts. If 3D, pi[j, t, a] gives the probability that the policy pi_t assigns arm a to context X_j. If 2D, pi[j, t] gives the probability that the policy pi_t assigns arm A_j (arm actually chosen under X_j in the history) to context X_j. Please see vignette for more details.
`reward`	A vector of observed rewards of length T x B
`arm`	A vector of length T x B indicating which arm was selected in each context
`batch`	(Optional) A vector or integer. If a vector, gives the batch assignment for each context. If an integer, interpreted as the batch size and contexts are assigned to a batch in the order of the dataset. Default is 1.

Value

The crammed variance estimate of the policy value estimate.

`cram_estimator` *Cram Policy Estimator for Policy Value Difference (Delta)*

Description

This function returns the cram policy estimator for the policy value difference (delta).

Usage

```
cram_estimator(X, Y, D, pi, batch_indices, propensity = NULL)
```

Arguments

`X`	A matrix or data frame of covariates for each sample.
`Y`	A vector of outcomes for the n individuals.
`D`	A vector of binary treatments for the n individuals.
`pi`	A matrix of n rows and (nb_batch + 1) columns, where n is the sample size and nb_batch is the number of batches, containing the policy assignment for each individual for each policy. The first column represents the baseline policy.
`batch_indices`	A list where each element is a vector of indices corresponding to the individuals in each batch.
`propensity`	The propensity score function

Value

The estimated policy value difference (Delta).

`cram_expected_loss` *Cram ML Expected Loss Estimate*

Description

This function computes the Cram ML expected loss estimator based on the given loss matrix and batch indices.

Usage

```
cram_expected_loss(loss, batch_indices)
```

Arguments

`loss`	A matrix of loss values with N rows (data points) and K+1 columns (batches). We assume that the first column of the loss matrix contains only zeros. The following nb_batch columns contain the losses of each trained model for each individual.
`batch_indices`	A list where each element is a vector of indices corresponding to a batch.

Value

The Cram ML expected loss estimate

cram_learning *Cram Policy Learning*

Description

This function performs the learning part of the Cram Policy method.

Usage

```
cram_learning(
  X,
  D,
  Y,
  batch,
  model_type = "causal_forest",
  learner_type = "ridge",
  baseline_policy = NULL,
  parallelize_batch = FALSE,
  model_params = NULL,
  custom_fit = NULL,
  custom_predict = NULL,
  n_cores = detectCores() - 1,
  propensity = NULL
)
```

Arguments

`X`	A matrix or data frame of covariates for each sample.
`D`	A vector of binary treatment indicators (1 for treated, 0 for untreated).
`Y`	A vector of outcome values for each sample.
`batch`	Either an integer specifying the number of batches (which will be created by random sampling) or a vector of length equal to the sample size providing the batch assignment (index) for each individual in the sample.
`model_type`	The model type for policy learning. Options include `"causal_forest"`, `"s_learner"`, and `"m_learner"`. Default is `"causal_forest"`. Note: you can also set model_type to NULL and specify custom_fit and custom_predict to use your custom model.
`learner_type`	The learner type for the chosen model. Options include `"ridge"` for Ridge Regression, `"fnn"` for Feedforward Neural Network and `"caret"` for Caret. Default is `"ridge"`. if model_type is 'causal_forest', choose NULL, if model_type is 's_learner' or 'm_learner', choose between 'ridge', 'fnn' and 'caret'.
`baseline_policy`	A list providing the baseline policy (binary 0 or 1) for each sample. If `NULL`, the baseline policy defaults to a list of zeros with the same length as the number of rows in `X`.

`parallelize_batch`	Logical. Whether to parallelize batch processing (i.e. the cram method learns T policies, with T the number of batches. They are learned in parallel when parallelize_batch is TRUE vs. learned sequentially using the efficient data.table structure when parallelize_batch is FALSE, recommended for light weight training). Defaults to `FALSE`.
`model_params`	A list of additional parameters to pass to the model, which can be any parameter defined in the model reference package. Defaults to `NULL`.
`custom_fit`	A custom, user-defined, function that outputs a fitted model given training data (allows flexibility). Defaults to `NULL`.
`custom_predict`	A custom, user-defined, function for making predictions given a fitted model and test data (allow flexibility). Defaults to `NULL`.
`n_cores`	Number of cores to use for parallelization when parallelize_batch is set to TRUE. Defaults to detectCores() - 1.
`propensity`	The propensity score

Value

A list containing:

`final_policy_model`	The final fitted policy model, depending on `model_type` and `learner_type`.
`policies`	A matrix of learned policies, where each column represents a batch's learned policy and the first column is the baseline policy.
`batch_indices`	The indices for each batch, either as generated (if `batch` is an integer) or as provided by the user.

See Also

`causal_forest`, `cv.glmnet`, `keras_model_sequential`

`cram_ml` *Cram ML: Simultaneous Machine Learning and Evaluation*

Description

Performs the Cram method for simultaneous machine learning and evaluation.

Usage

```
cram_ml(
  data,
  batch,
  formula = NULL,
  caret_params = NULL,
  parallelize_batch = FALSE,
  loss_name = NULL,
  custom_fit = NULL,
  custom_predict = NULL,
  custom_loss = NULL,
  alpha = 0.05,
  classify = FALSE
)
```

Arguments

`data`	A matrix or data frame of covariates. For supervised learning, must include the target variable specified in formula.
`batch`	Integer specifying number of batches or vector of pre-defined batch assignments.
`formula`	Formula for supervised learning (e.g., y ~ .).
`caret_params`	List of parameters for caret::train() containing: • method: Model type (e.g., "rf", "glm", "xgbTree" for supervised learning) • Additional method-specific parameters
`parallelize_batch`	Logical indicating whether to parallelize batch processing (default = FALSE).
`loss_name`	Name of loss metric (supported: "se", "logloss", "accuracy").
`custom_fit`	Optional custom model training function.
`custom_predict`	Optional custom prediction function.
`custom_loss`	Optional custom loss function.
`alpha`	Confidence level for intervals (default = 0.05).
`classify`	Indicate if this is a classification problem. Defaults to FALSE.

Value

A list containing:

- raw_results: Data frame with performance metrics
- interactive_table: The same performance metrics in a user-friendly interface
- final_ml_model: Trained model object

See Also

`train` for model training parameters

Examples

```
# Load necessary libraries
library(caret)

# Set seed for reproducibility
set.seed(42)
```

```
# Generate example dataset
X_data <- data.frame(x1 = rnorm(100), x2 = rnorm(100), x3 = rnorm(100))
Y_data <- rnorm(100)  # Continuous target variable for regression
data_df <- data.frame(X_data, Y = Y_data)  # Ensure target variable is included

# Define caret parameters for simple linear regression (no cross-validation)
caret_params_lm <- list(
  method = "lm",
  trControl = trainControl(method = "none")
)

nb_batch <- 5

# Run ML learning function
result <- cram_ml(
  data = data_df,
  formula = Y ~ .,  # Linear regression model
  batch = nb_batch,
  loss_name = 'se',
  caret_params = caret_params_lm
)

result$raw_results
result$interactive_table
result$final_ml_model
```

cram_policy — *Cram Policy: Efficient Simultaneous Policy Learning and Evaluation*

Description

This function performs the cram method (simultaneous policy learning and evaluation) for binary policies on data including covariates (X), binary treatment indicator (D) and outcomes (Y).

Usage

```
cram_policy(
  X,
  D,
  Y,
  batch,
  model_type = "causal_forest",
  learner_type = "ridge",
  baseline_policy = NULL,
  parallelize_batch = FALSE,
  model_params = NULL,
  custom_fit = NULL,
  custom_predict = NULL,
  alpha = 0.05,
  propensity = NULL
)
```

Arguments

`X`	A matrix or data frame of covariates for each sample.
`D`	A vector of binary treatment indicators (1 for treated, 0 for non-treated).
`Y`	A vector of outcome values for each sample.
`batch`	Either an integer specifying the number of batches (which will be created by random sampling) or a vector of length equal to the sample size providing the batch assignment (index) for each individual in the sample.
`model_type`	The model type for policy learning. Options include `"causal_forest"`, `"s_learner"`, and `"m_learner"`. Default is `"causal_forest"`. Note: you can also set model_type to NULL and specify custom_fit and custom_predict to use your custom model.
`learner_type`	The learner type for the chosen model. Options include `"ridge"` for Ridge Regression, `"fnn"` for Feedforward Neural Network and `"caret"` for Caret. Default is `"ridge"`. if model_type is 'causal_forest', choose NULL, if model_type is 's_learner' or 'm_learner', choose between 'ridge', 'fnn' and 'caret'.
`baseline_policy`	A list providing the baseline policy (binary 0 or 1) for each sample. If `NULL`, defaults to a list of zeros with the same length as the number of rows in `X`.
`parallelize_batch`	Logical. Whether to parallelize batch processing (i.e. the cram method learns T policies, with T the number of batches. They are learned in parallel when parallelize_batch is TRUE vs. learned sequentially using the efficient data.table structure when parallelize_batch is FALSE, recommended for light weight training). Defaults to `FALSE`.
`model_params`	A list of additional parameters to pass to the model, which can be any parameter defined in the model reference package. Defaults to `NULL`.
`custom_fit`	A custom, user-defined, function that outputs a fitted model given training data (allows flexibility). Defaults to `NULL`.
`custom_predict`	A custom, user-defined, function for making predictions given a fitted model and test data (allow flexibility). Defaults to `NULL`.
`alpha`	Significance level for confidence intervals. Default is 0.05 (95% confidence).
`propensity`	The propensity score function for binary treatment indicator (D) (probability for each unit to receive treatment). Defaults to 0.5 (random assignment).

Value

A list containing:

- `raw_results`: A data frame summarizing key metrics with truncated decimals:
  - `Delta Estimate`: The estimated treatment effect (delta).
  - `Delta Standard Error`: The standard error of the delta estimate.
  - `Delta CI Lower`: The lower bound of the confidence interval for delta.

  - Delta CI Upper: The upper bound of the confidence interval for delta.
  - Policy Value Estimate: The estimated policy value.
  - Policy Value Standard Error: The standard error of the policy value estimate.
  - Policy Value CI Lower: The lower bound of the confidence interval for policy value.
  - Policy Value CI Upper: The upper bound of the confidence interval for policy value.
  - Proportion Treated: The proportion of individuals treated under the final policy.
- interactive_table: An interactive table summarizing key metrics for detailed exploration.
- final_policy_model: The final fitted policy model based on model_type and learner_type or custom_fit.

See Also

causal_forest, cv.glmnet, keras_model_sequential

Examples

```
# Example data
X_data <- matrix(rnorm(100 * 5), nrow = 100, ncol = 5)
D_data <- as.integer(sample(c(0, 1), 100, replace = TRUE))
Y_data <- rnorm(100)
nb_batch <- 5

# Perform CRAM policy
result <- cram_policy(X = X_data,
                          D = D_data,
                          Y = Y_data,
                          batch = nb_batch)

# Access results
result$raw_results
result$interactive_table
result$final_policy_model
```

---

cram_policy_value_estimator

Cram Policy: Estimator for Policy Value (Psi)

Description

This function returns the cram estimator for the policy value (psi).

Usage

```
cram_policy_value_estimator(X, Y, D, pi, batch_indices, propensity = NULL)
```

Arguments

`X`	A matrix or data frame of covariates for each sample.
`Y`	A vector of outcomes for the n individuals.
`D`	A vector of binary treatments for the n individuals.
`pi`	A matrix of n rows and (nb_batch + 1) columns, where n is the sample size and nb_batch is the number of batches, containing the policy assignment for each individual for each policy. The first column represents the baseline policy.
`batch_indices`	A list where each element is a vector of indices corresponding to the individuals in each batch.
`propensity`	Propensity score function

Value

The estimated policy value.

`cram_simulation` *Cram Policy Simulation*

Description

This function performs the cram method (simultaneous learning and evaluation) on simulation data, for which the data generation process (DGP) is known. The data generation process for X can be given directly as a function or induced by a provided dataset via row-wise bootstrapping. Results are averaged across Monte Carlo replicates for the given DGP.

Usage

```
cram_simulation(
  X = NULL,
  dgp_X = NULL,
  dgp_D,
  dgp_Y,
  batch,
  nb_simulations,
  nb_simulations_truth = NULL,
  sample_size,
  model_type = "causal_forest",
  learner_type = "ridge",
  alpha = 0.05,
  baseline_policy = NULL,
  parallelize_batch = FALSE,
  model_params = NULL,
  custom_fit = NULL,
  custom_predict = NULL,
  propensity = NULL
)
```

Arguments

`X`	Optional. A matrix or data frame of covariates for each sample inducing empirically the DGP for covariates.
`dgp_X`	Optional. A function to generate covariate data for simulations.
`dgp_D`	A vectorized function to generate binary treatment assignments for each sample.
`dgp_Y`	A vectorized function to generate the outcome variable for each sample given the treatment and covariates.
`batch`	Either an integer specifying the number of batches (which will be created by random sampling) or a vector of length equal to the sample size providing the batch assignment (index) for each individual in the sample.
`nb_simulations`	The number of simulations (Monte Carlo replicates) to run.
`nb_simulations_truth`	Optional. The number of additional simmulations (Monte Carlo replicates) beyond nb_simulations to use when calculating the true policy value difference (delta) and the true policy value (psi)
`sample_size`	The number of samples in each simulation.
`model_type`	The model type for policy learning. Options include `"causal_forest"`, `"s_learner"`, and `"m_learner"`. Default is `"causal_forest"`. Note: you can also set model_type to NULL and specify custom_fit and custom_predict to use your custom model.
`learner_type`	The learner type for the chosen model. Options include `"ridge"` for Ridge Regression, `"fnn"` for Feedforward Neural Network and `"caret"` for Caret. Default is `"ridge"`. if model_type is 'causal_forest', choose NULL, if model_type is 's_learner' or 'm_learner', choose between 'ridge', 'fnn' and 'caret'.
`alpha`	Significance level for confidence intervals. Default is 0.05 (95% confidence).
`baseline_policy`	A list providing the baseline policy (binary 0 or 1) for each sample. If `NULL`, defaults to a list of zeros with the same length as the number of rows in `X`.
`parallelize_batch`	Logical. Whether to parallelize batch processing (i.e. the cram method learns T policies, with T the number of batches. They are learned in parallel when parallelize_batch is TRUE vs. learned sequentially using the efficient data.table structure when parallelize_batch is FALSE, recommended for light weight training). Defaults to `FALSE`.
`model_params`	A list of additional parameters to pass to the model, which can be any parameter defined in the model reference package. Defaults to `NULL`.
`custom_fit`	A custom, user-defined, function that outputs a fitted model given training data (allows flexibility). Defaults to `NULL`.
`custom_predict`	A custom, user-defined, function for making predictions given a fitted model and test data (allow flexibility). Defaults to `NULL`.
`propensity`	The propensity score model

Value

A list containing:

`avg_proportion_treated`	The average proportion of treated individuals across simulations.
`avg_delta_estimate`	The average delta estimate across simulations.
`avg_delta_standard_error`	The average standard error of delta estimates.
`delta_empirical_bias`	The empirical bias of delta estimates.
`delta_empirical_coverage`	The empirical coverage of delta confidence intervals.
`avg_policy_value_estimate`	The average policy value estimate across simulations.
`avg_policy_value_standard_error`	The average standard error of policy value estimates.
`policy_value_empirical_bias`	The empirical bias of policy value estimates.
`policy_value_empirical_coverage`	The empirical coverage of policy value confidence intervals.

See Also

`causal_forest`, `cv.glmnet`, `keras_model_sequential`

Examples

```
set.seed(123)

# dgp_X <- function(n) {
#   data.table::data.table(
#     binary     = rbinom(n, 1, 0.5),
#     discrete   = sample(1:5, n, replace = TRUE),
#     continuous = rnorm(n)
#   )
# }

n <- 100

X_data <- data.table::data.table(
  binary     = rbinom(n, 1, 0.5),
  discrete   = sample(1:5, n, replace = TRUE),
  continuous = rnorm(n)
)


dgp_D <- function(X) rbinom(nrow(X), 1, 0.5)

dgp_Y <- function(D, X) {
  theta <- ifelse(
    X[, binary] == 1 & X[, discrete] <= 2,  # Group 1: High benefit
    1,
    ifelse(X[, binary] == 0 & X[, discrete] >= 4,  # Group 3: Negative benefit
           -1,
           0.1)  # Group 2: Neutral effect
  )
  Y <- D * (theta + rnorm(length(D), mean = 0, sd = 1)) +
    (1 - D) * rnorm(length(D))  # Outcome for untreated
  return(Y)
}

# Parameters
nb_simulations <- 100
nb_simulations_truth <- 200
batch <- 5

# Perform CRAM simulation
result <- cram_simulation(
  X = X_data,
  dgp_D = dgp_D,
  dgp_Y = dgp_Y,
  batch = batch,
  nb_simulations = nb_simulations,
  nb_simulations_truth = nb_simulations_truth,
  sample_size = 500
)

result$raw_results
result$interactive_table
```

---

cram_variance_estimator

Cram Policy: Variance Estimate of the crammed Policy Value Difference (Delta)

Description

This function estimates the asymptotic variance of the cram estimator for the policy value difference (delta).

Usage

```
cram_variance_estimator(X, Y, D, pi, batch_indices, propensity = NULL)
```

Arguments

X	A matrix or data frame of covariates for each sample.
Y	A vector of outcomes for the n individuals.
D	A vector of binary treatments for the n individuals.
pi	A matrix of n rows and (nb_batch + 1) columns, where n is the sample size and nb_batch is the number of batches, containing the policy assignment for each individual for each policy. The first column represents the baseline policy.

`batch_indices`	A list where each element is a vector of indices corresponding to the individuals in each batch.
`propensity`	The propensity score function

Value

The estimated variance of the policy value difference (Delta)

`cram_variance_estimator_policy_value` *Cram Policy: Variance Estimate of the crammed Policy Value estimate (Psi)*

Description

This function estimates the asymptotic variance of the cram estimator for the policy value (psi).

Usage

```
cram_variance_estimator_policy_value(
  X,
  Y,
  D,
  pi,
  batch_indices,
  propensity = NULL
)
```

Arguments

`X`	A matrix or data frame of covariates for each sample.
`Y`	A vector of outcomes for the n individuals.
`D`	A vector of binary treatments for the n individuals.
`pi`	A matrix of n rows and (nb_batch + 1) columns, where n is the sample size and nb_batch is the number of batches, containing the policy assignment for each individual for each policy. The first column represents the baseline policy.
`batch_indices`	A list where each element is a vector of indices corresponding to the individuals in each batch.
`propensity`	Propensity score function

Value

The variance estimate of the crammed Policy Value estimate (Psi) `cram_var_expected_loss` *Cram ML: Variance Estimate of the crammed expected loss estimate*

Description

This function computes the variance estimator based on the given loss matrix and batch indices.

Usage

```
cram_var_expected_loss(loss, batch_indices)
```

Arguments

`loss`	A matrix of loss values with N rows (data points) and K+1 columns (batches). We assume that the first column of the loss matrix contains only zeros. The following nb_batch columns contain the losses of each trained model for each individual.
`batch_indices`	A list where each element is a vector of indices corresponding to a batch.

Value

The variance estimate of the crammed expected loss estimate

`fit_model` *Cram Policy: Fit Model*

Description

This function trains a given unfitted model with the provided data and parameters, according to model type and learner type.

Usage

```
fit_model(model, X, Y, D, model_type, learner_type, model_params, propensity)
```

Arguments

`model`	An unfitted model object, as returned by ‘set_model‘.
`X`	A matrix or data frame of covariates for the samples.
`Y`	A vector of outcome values.
`D`	A vector of binary treatment indicators (1 for treated, 0 for untreated).
`model_type`	The model type for policy learning. Options include `"causal_forest"`, `"s_learner"`, and `"m_learner"`. Default is `"causal_forest"`.

`learner_type`	The learner type for the chosen model. Options include `"ridge"` for Ridge Regression and `"fnn"` for Feedforward Neural Network. Default is `"ridge"`.
`model_params`	A list of additional parameters to pass to the model, which can be any parameter defined in the model reference package. Defaults to `NULL`.
`propensity`	The propensity score

Value

The fitted model object.

`fit_model_ml` *Cram ML: Fit Model ML*

Description

This function trains a given unfitted model with the provided data and parameters, according to model type and learner type.

Usage

```
fit_model_ml(data, formula, caret_params, classify)
```

Arguments

`data`	The dataset
`formula`	The formula
`caret_params`	The parameters for caret model
`classify`	Indicate if this is a classification problem. Defaults to FALSE

Value

The fitted model object.

`get_betas` *Generate Reward Parameters for Simulated Linear Bandits*

Description

Creates a list of matrices representing the arm-specific reward-generating parameters (betas) used in contextual linear bandit simulations. Each matrix corresponds to one simulation and contains normalized random coefficients.

Usage

```
get_betas(simulations, d, k)
```

Arguments

`simulations`	Integer. Number of simulations.
`d`	Integer. Number of features (context dimensions).
`k`	Integer. Number of arms.

Value

A list of length `simulations + 1` (first element being discarded in the underlying simulation package), each containing a `d x k` matrix of normalized reward parameters.

`LinUCBDisjointPolicyEpsilon` *LinUCB Disjoint Policy with Epsilon-Greedy Exploration*

Description

LinUCB Disjoint Policy with Epsilon-Greedy Exploration

LinUCB Disjoint Policy with Epsilon-Greedy Exploration

Details

Implements the disjoint LinUCB algorithm with upper confidence bounds and epsilon-greedy exploration.

Methods

- ‘initialize(alpha = 1.0, epsilon = 0.1)‘: Create a new LinUCBDisjointPolicyEpsilon object. - ‘set_parameters(context_params)‘: Initialize arm-level parameters. - ‘get_action(t, context)‘: Selects an arm using epsilon-greedy UCB. - ‘set_reward(t, context, action, reward)‘: Updates internal statistics based on observed reward.

Super class

`cramR::NA`

Public fields

`alpha` Numeric, exploration parameter controlling the width of the confidence bound.

`epsilon` Numeric, probability of selecting a random action (exploration).

`class_name` Internal class name.

Methods

Public methods:

- `LinUCBDisjointPolicyEpsilon$new()`
- `LinUCBDisjointPolicyEpsilon$set_parameters()`
- `LinUCBDisjointPolicyEpsilon$get_action()`
- `LinUCBDisjointPolicyEpsilon$set_reward()`
- `LinUCBDisjointPolicyEpsilon$clone()`

Method `new()`**:** Initializes the policy with UCB parameter `alpha` and exploration rate `epsilon`.

Usage:

```
LinUCBDisjointPolicyEpsilon$new(alpha = 1, epsilon = 0.1)
```

Arguments:

`alpha` Numeric. Controls width of the UCB bonus.

`epsilon` Numeric between 0 and 1. Probability of random action selection.

Method `set_parameters()`**:** Set arm-specific parameter structures.

Usage:

```
LinUCBDisjointPolicyEpsilon$set_parameters(context_params)
```

Arguments:

`context_params` A list with context information, typically including the number of unique features.

Method `get_action()`**:** Selects an arm using epsilon-greedy Upper Confidence Bound (UCB).

Usage:

```
LinUCBDisjointPolicyEpsilon$get_action(t, context)
```

Arguments:

`t` Integer time step.

`context` A list with contextual features and number of arms.

Returns: A list containing the selected action.

Method `set_reward()`**:** Updates internal statistics using the observed reward for the selected arm.

Usage:

```
LinUCBDisjointPolicyEpsilon$set_reward(t, context, action, reward)
```

Arguments:

`t` Integer time step.

`context` Contextual features for all arms at time `t`.

`action` A list containing the chosen arm.

`reward` A list containing the observed reward for the selected arm.

Returns: Updated internal parameters.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LinUCBDisjointPolicyEpsilon$clone(deep = FALSE)
```

Arguments:

`deep`	Whether to make a deep clone.

`ml_learning` *Cram ML: Generalized ML Learning*

Description

This function performs batch-wise learning for machine learning models.

Usage

```
ml_learning(
  data,
  formula = NULL,
  batch,
  parallelize_batch = FALSE,
  loss_name = NULL,
  caret_params = NULL,
  custom_fit = NULL,
  custom_predict = NULL,
  custom_loss = NULL,
  n_cores = detectCores() - 1,
  classify = FALSE
)
```

Arguments

`data`	A matrix or data frame of features. Must include the target variable.
`formula`	Formula specifying the relationship between the target and predictors for supervised learning.
`batch`	Either an integer specifying the number of batches (randomly sampled) or a vector of length equal to the sample size indicating batch assignment for each observation.
`parallelize_batch`	Logical. Whether to parallelize batch processing. Defaults to 'FALSE'.
`loss_name`	The name of the loss function to be used (e.g., '"se"', '"logloss"').
`caret_params`	A list of parameters to pass to the 'caret::train()' function. - Required: 'method' (e.g., '"glm"', '"rf"').
`custom_fit`	A custom function for training user-defined models. Defaults to 'NULL'.
`custom_predict`	A custom function for making predictions from user-defined models. Defaults to 'NULL'.

`custom_loss`	Optional custom function for computing the loss of a trained model on the data. Should return a vector containing per-instance losses.
`n_cores`	Number of CPU cores to use for parallel processing (‘parallelize_batch = TRUE‘). Defaults to ‘detectCores() - 1‘.
`classify`	Indicate if this is a classification problem. Defaults to FALSE

Value

A list containing:

`final_ml_model`	The final trained ML model.
`losses`	A matrix of losses where each column represents a batch’s trained model. The first column contains zeros (baseline model).
`batch_indices`	The indices of observations in each batch.

`model_predict` *Cram Policy: Predict with the Specified Model*

Description

This function performs inference using a trained model, providing flexibility for different types of models such as Causal Forest, Ridge Regression, and Feedforward Neural Networks (FNNs).

Usage

```
model_predict(model, X, D, model_type, learner_type, model_params)
```

Arguments

`model`	A trained model object returned by the ‘fit_model‘ function.
`X`	A matrix or data frame of covariates for which predictions are required.
`D`	A vector of binary treatment indicators (1 for treated, 0 for untreated). Optional, depending on the model type.
`model_type`	The model type for policy learning. Options include `"causal_forest"`, `"s_learner"`, and `"m_learner"`. Default is `"causal_forest"`.
`learner_type`	The learner type for the chosen model. Options include `"ridge"` for Ridge Regression and `"fnn"` for Feedforward Neural Network. Default is `"ridge"`.
`model_params`	A list of additional parameters to pass to the model, which can be any parameter defined in the model reference package. Defaults to `NULL`.

Value

A vector of binary policy assignments, depending on the `model_type` and `learner_type`.

model_predict_ml *Cram ML: Predict with the Specified Model*

Description

This function performs inference using a trained model

Usage

```
model_predict_ml(
  model,
  data,
  formula,
  caret_params,
  cram_policy_handle = FALSE
)
```

Arguments

`model`	A trained model object returned by the ‘fit_model_ml‘ function.
`data`	The dataset
`formula`	The formula
`caret_params`	The parameters of the caret model
`cram_policy_handle`	Internal use. Post-process predictions differently for cram policy use. Defaults to FALSE.

Value

Predictions of the model on the data

set_model *Cram Policy: Set Model*

Description

This function maps the model type and learner type to the corresponding model function.

Usage

```
set_model(model_type, learner_type, model_params)
```

Arguments

`model_type`	The model type for policy learning. Options include `"causal_forest"`, `"s_learner"`, and `"m_learner"`. Default is `"causal_forest"`. Note: you can also set model_type to NULL and specify custom_fit and custom_predict to use your custom model.
`learner_type`	The learner type for the chosen model. Options include `"ridge"` for Ridge Regression, `"fnn"` for Feedforward Neural Network and `"caret"` for Caret. Default is `"ridge"`. if model_type is 'causal_forest', choose NULL, if model_type is 's_learner' or 'm_learner', choose between 'ridge', 'fnn' and 'caret'.
`model_params`	A list of additional parameters to pass to the model, which can be any parameter defined in the model reference package. Defaults to `NULL`. For FNNs, the following elements are defined in the model params list: `input_layer` A list defining the input layer. Must include: `units` Number of units in the input layer. `activation` Activation function for the input layer. `input_shape` Input shape for the layer. `layers` A list of lists, where each sublist specifies a hidden layer with: `units` Number of units in the layer. `activation` Activation function for the layer. `output_layer` A list defining the output layer. Must include: `units` Number of units in the output layer. `activation` Activation function for the output layer (e.g., `"linear"` or `"sigmoid"`). `compile_args` A list of arguments for compiling the model. Must include: `optimizer` Optimizer for training (e.g., `"adam"` or `"sgd"`). `loss` Loss function (e.g., `"mse"` or `"binary_crossentropy"`). `metrics` Optional list of metrics for evaluation (e.g., `c("accuracy")`). For other learners (e.g., `"ridge"` or `"causal_forest"`), `model_params` can include relevant hyperparameters.

Value

The instantiated model object or the corresponding model function.

`test_baseline_policy` *Validate or Set the Baseline Policy*

Description

This function validates a provided baseline policy or sets a default baseline policy of zeros for all individuals.

Usage

```
test_baseline_policy(baseline_policy, n)
```

Arguments

`baseline_policy`	A list representing the baseline policy for each individual. If `NULL`, a default baseline policy of zeros is created.
`n`	An integer specifying the number of individuals in the population.

Value

A validated or default baseline policy as a list of numeric values.

`test_batch` *Validate or Generate Batch Assignments*

Description

This function validates a provided batch assignment or generates random batch assignments for individuals.

Usage

```
test_batch(batch, n)
```

Arguments

`batch`	Either an integer specifying the number of batches or a vector/list of batch assignments for all individuals.
`n`	An integer specifying the number of individuals in the population.

Value

A list containing:

`batches` A list where each element contains the indices of individuals assigned to a specific batch.

`nb_batch` The total number of batches.

`validate_params` *Cram Policy: Validate User-Provided Parameters for a Model*

Description

This function validates user-provided parameters against the formal arguments of a specified model function. It ensures that all user-specified parameters are recognized by the model and raises an error for invalid parameters.

Usage

```
validate_params(model_function, model_type, learner_type, user_params)
```

Arguments

`model_function`	The model function for which parameters are being validated (e.g., `grf::causal_forest`).
`model_type`	The model type for policy learning. Options include `"causal_forest"`, `"s_learner"`, and `"m_learner"`. Default is `"causal_forest"`. Note: you can also set model_type to NULL and specify custom_fit and custom_predict to use your custom model.
`learner_type`	The learner type for the chosen model. Options include `"ridge"` for Ridge Regression, `"fnn"` for Feedforward Neural Network and `"caret"` for Caret. Default is `"ridge"`. if model_type is 'causal_forest', choose NULL, if model_type is 's_learner' or 'm_learner', choose between 'ridge', 'fnn' and 'caret'.
`user_params`	A named list of parameters provided by the user.

Value

A named list of validated parameters that are safe to pass to the model function.

`validate_params_fnn` *Cram Policy: Validate Parameters for Feedforward Neural Networks (FNNs)*

Description

This function validates user-provided parameters for a Feedforward Neural Network (FNN) model. It ensures the correct structure for `input_layer`, `layers`, `output_layer`, `compile_args` and `fit_params`.

Usage

```
validate_params_fnn(model_type, learner_type, model_params, X)
```

Arguments

`model_type`	The model type for policy learning. Options include `"causal_forest"`, `"s_learner"`, and `"m_learner"`. Default is `"causal_forest"`. Note: you can also set model_type to NULL and specify custom_fit and custom_predict to use your custom model.
`learner_type`	The learner type for the chosen model. Options include `"ridge"` for Ridge Regression, `"fnn"` for Feedforward Neural Network and `"caret"` for Caret. Default is `"ridge"`. if model_type is 'causal_forest', choose NULL, if model_type is 's_learner' or 'm_learner', choose between 'ridge', 'fnn' and 'caret'.
`model_params`	A named list of parameters provided by the user for configuring the FNN model.
`X`	A matrix or data frame of covariates for which the parameters are validated.

Value

A named list of validated parameters merged with defaults for any missing values.

Index

BatchContextualEpsilonGreedyPolicy, 3
BatchContextualLinTSPolicy, 5
BatchLinUCBDisjointPolicyEpsilon, 7

causal_forest, *18*, *22*, *25*
ContextualLinearBandit, 9
cram_bandit, 11
cram_bandit_est, 12
cram_bandit_sim, 13
cram_bandit_var, 15
cram_estimator, 15
cram_expected_loss, 16
cram_learning, 17
cram_ml, 18
cram_policy, 20
cram_policy_value_estimator, 22
cram_simulation, 23
cram_var_expected_loss, 28
cram_variance_estimator, 26
cram_variance_estimator_policy_value, 27
cv.glmnet, *18*, *22*, *25*

fit_model, 28
fit_model_ml, 29

get_betas, 29

keras_model_sequential, *18*, *22*, *25*

LinUCBDisjointPolicyEpsilon, 30

ml_learning, 32
model_predict, 33
model_predict_ml, 34

set_model, 34

test_baseline_policy, 35
test_batch, 36
train, *19*

validate_params, 37
validate_params_fnn, 37