Package: shapr (via r-universe)

October 20, 2026

Version 1.1.0

Title Prediction Explanation with Dependence-Aware Shapley Values

Description Complex machine learning models are often hard to interpret. However, in many situations it is crucial to understand and explain why a model made a specific prediction. Shapley values is the only method for such prediction explanation framework with a solid theoretical foundation. Previously known methods for estimating the Shapley values do, however, assume feature independence. This package implements methods which accounts for any feature dependence, and thereby produces more accurate estimates of the true Shapley values. An accompanying 'Python' wrapper ('pyshapr') is available through PyPI.

URL https://norskregnesentral.github.io/shapr/, https://github.com/NorskRegnesentral/shapr/

BugReports https://github.com/NorskRegnesentral/shapr/issues

License MIT + file LICENSE

Encoding UTF-8

ByteCompile true

Language en-US

Depends R (>= 3.5.0)

Imports stats, data.table (>= 1.15.0), Rcpp (>= 0.12.15), Matrix, future, future.apply, methods, cli, rlang

Suggests arf, ranger, xgboost, mgcv, testthat (>= 3.0.0), knitr, rmarkdown, roxygen2, ggplot2, gbm, party, partykit, waldo, progressr, ggbeeswarm, vdiffr, forecast, torch, GGally, coro, parsnip, recipes, workflows, tune, dials, yardstick, hardhat, rsample

LinkingTo RcppArmadillo, Rcpp

VignetteBuilder knitr

Config/Needs/website DT

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Martin Jullum [cre, aut] (ORCID: <https://orcid.org/0000-0003-3908-5155>), Lars Henry Berge Olsen [aut] (ORCID: <https://orcid.org/0009-0006-9360-6993>), Annabelle Redelmeier [aut], Jon Lachmann [aut] (ORCID: <https://orcid.org/0000-0001-8396-5673>), Nikolai Sellereite [aut] (ORCID: <https://orcid.org/0000-0002-4671-0337>), Anders Løland [ctb], Jens Christian Wahl [ctb], Camilla Lingjærde [ctb], Norsk Regnesentral [cph, fnd]

Maintainer Martin Jullum <Martin.Jullum@nr.no>

Repository https://cran.r-universe.dev

Date/Publication 2026-09-11 21:00:02 UTC

RemoteUrl https://github.com/cran/shapr

RemoteRef HEAD

RemoteSha ac056d08458f7d13283eff7cdf6090f5d49d038f

Contents

explain . . . 3
explain_forecast . . . 15
get_extra_comp_args_default . . . 23
get_iterative_args_default . . . 26
get_output_args_default . . . 27
get_results . . . 28
get_supported_approaches . . . 30
get_supported_models . . . 30
plot.shapr . . . 31
plot_MSEv_eval_crit . . . 35
plot_SV_several_approaches . . . 39
plot_vaeac_eval_crit . . . 44
plot_vaeac_imputed_ggpairs . . . 48
print.shapr . . . 50
print.summary.shapr . . . 51
summary.shapr . . . 52
vaeac_get_extra_para_default . . . 53
vaeac_train_model_continue . . . 57

Index **60**

`explain` *Explain the Output of Machine Learning Models with Dependence-Aware (Conditional/Observational) Shapley Values*

Description

Compute dependence-aware Shapley values for observations in `x_explain` from the specified `model` using the method specified in `approach` to estimate the conditional expectation. See Aas et al. (2021) for a thorough introduction to dependence-aware prediction explanation with Shapley values. For an overview of the methodology and capabilities of the package, see the software paper Jullum et al. (2025), or the pkgdown site at norskregnesentral.github.io/shapr/.

Usage

```
explain(
  model,
  x_explain,
  x_train,
  approach,
  phi0,
  iterative = NULL,
  max_n_coalitions = NULL,
  group = NULL,
  n_MC_samples = 1000,
  seed = NULL,
  verbose = "basic",
  predict_model = NULL,
  get_model_specs = NULL,
  prev_shapr_object = NULL,
  asymmetric = FALSE,
  causal_ordering = NULL,
  confounding = NULL,
  extra_computation_args = list(),
  iterative_args = list(),
  output_args = list(),
  scope = "local",
  y_explain = NULL,
  ...
)
```

Arguments

`model`	Model object. The model whose predictions you want to explain. Run `get_supported_models()` for a table of which models `explain` supports natively. Unsupported models can still be explained by passing `predict_model` and (optionally) `get_model_specs`, see details for more information.

x_explain — Matrix or data.frame/data.table. Features for which predictions should be explained.

x_train — Matrix or data.frame/data.table. Data used to estimate the (conditional) feature distributions needed to properly estimate the conditional expectations in the Shapley formula.

approach — Character vector of length 1 or one less than the number of features. All elements should either be "arf", "categorical", "copula", "ctree", "empirical", "gaussian", "independence", "regression_separate", "regression_surrogate", "timeseries", or "vaeac". The two regression approaches cannot be combined with any other approach. See details for more information.

phi0 — Numeric. The prediction value for unseen data, i.e., an estimate of the expected prediction without conditioning on any features. Typically set this equal to the mean of the response in the training data, but alternatives such as the mean of the training predictions are also reasonable.

iterative — Logical or NULL. If NULL (default), set to TRUE if there are more than 5 features/groups, and FALSE otherwise. If TRUE, Shapley values are estimated iteratively for faster, sufficiently accurate results. First an initial number of coalitions is sampled, then bootstrapping estimates the variance of the Shapley values. A convergence criterion determines if the variances are sufficiently small. If not, additional samples are added. The process repeats until the variances are below the threshold. Specifics for the iterative process and convergence criterion are set via iterative_args.

max_n_coalitions — Integer. Upper limit on the number of unique feature/group coalitions to use in the iterative procedure (if iterative = TRUE). If iterative = FALSE, it represents the number of feature/group coalitions to use directly. The quantity refers to the number of unique feature coalitions if group = NULL, and group coalitions if group != NULL. max_n_coalitions = NULL corresponds to 2^n_features.

group — List. If NULL, regular feature-wise Shapley values are computed. If provided, group-wise Shapley values are computed. group then has length equal to the number of groups. Each list element contains the character vectors with the features included in the corresponding group. See Jullum et al. (2021) for more information on group-wise Shapley values.

n_MC_samples — Positive integer. For most approaches, it indicates the maximum number of samples to use in the Monte Carlo integration of every conditional expectation. For approach="ctree", n_MC_samples corresponds to the number of samples from the leaf node (see an exception related to the ctree.sample argument in setup_approach.ctree()). For approach="empirical", n_MC_samples is the K parameter in equations (14-15) of Aas et al. (2021), i.e. the maximum number of observations (with largest weights) that is used, see also the empirical.eta argument setup_approach.empirical().

seed — Positive integer. Specifies the seed before any code involving randomness is run. If NULL (default), no seed is set in the calling environment.

verbose — String vector or NULL. Controls verbosity (printout detail level) via one or more of "basic", "progress", "convergence", "shapley" and "vS_details". "basic" (default) displays basic information about the computation and messages about

parameters/checks. "progress" displays where in the calculation process the function currently is. "convergence" displays how close the Shapley value estimates are to convergence (only when iterative = TRUE). "shapley" displays intermediate Shapley value estimates and standard deviations (only when iterative = TRUE), and the final estimates. "vS_details" displays information about the v(S) estimates, most relevant for approach %in% c("regression_separate", "regression_surrogate", "vaeac"). NULL means no printout. Any combination can be used, e.g., verbose = c("basic", "vS_details").

predict_model Function. Prediction function to use when model is not natively supported. (Run get_supported_models() for a list of natively supported models.) The function must have two arguments, model and newdata, which specify the model and a data.frame/data.table to compute predictions for, respectively. The function must give the prediction as a numeric vector. NULL (the default) uses functions specified internally. Can also be used to override the default function for natively supported model classes.

get_model_specs

Function. An optional function for checking model/data consistency when model is not natively supported. (Run get_supported_models() for a list of natively supported models.) The function takes model as an argument and provides a list with 3 elements:

labels Character vector with the names of each feature.

classes Character vector with the class of each feature.

factor_levels Character vector with the levels for any categorical features.

If NULL (the default), internal functions are used for natively supported model classes, and checking is disabled for unsupported model classes. Can also be used to override the default function for natively supported model classes.

prev_shapr_object

shapr object or string. If an object of class shapr is provided, or a string with a path to where intermediate results are stored, then the function will use the previous object to continue the computation. This is useful if the computation is interrupted or you want higher accuracy than already obtained, and therefore want to continue the iterative estimation. See the general usage vignette for examples.

asymmetric Logical. Not applicable for (regular) non-causal explanations. If FALSE (default), explain computes regular symmetric Shapley values. If TRUE, explain computes asymmetric Shapley values based on the (partial) causal ordering given by causal_ordering. That is, explain only uses feature coalitions that respect the causal ordering. If asymmetric is TRUE and confounding is NULL (default), explain computes asymmetric conditional Shapley values as specified in Frye et al. (2020). If confounding is provided, i.e., not NULL, then explain computes asymmetric causal Shapley values as specified in Heskes et al. (2020).

causal_ordering

List. Not applicable for (regular) non-causal or asymmetric explanations. causal_ordering is an unnamed list of vectors specifying the components of the partial causal ordering that the coalitions must respect. Each vector represents a component and contains one or more features/groups identified by their names (strings)

or indices (integers). If `causal_ordering` is `NULL` (default), no causal ordering is assumed and all possible coalitions are allowed. No causal ordering is equivalent to a causal ordering with a single component that includes all features (`list(1:n_features)`) or groups (`list(1:n_groups)`) for feature-wise and group-wise Shapley values, respectively. For feature-wise Shapley values and `causal_ordering = list(c(1, 2), c(3, 4))`, the interpretation is that features 1 and 2 are the ancestors of features 3 and 4, while features 3 and 4 are on the same level. Note: All features/groups must be included in `causal_ordering` without duplicates.

`confounding` Logical vector. Not applicable for (regular) non-causal or asymmetric explanations. `confounding` is a logical vector specifying whether confounding is assumed for each component in the `causal_ordering`. If `NULL` (default), no assumption about the confounding structure is made and `explain` computes asymmetric/symmetric conditional Shapley values, depending on `asymmetric`. If `confounding` is a single logical (`FALSE` or `TRUE`), the assumption is set globally for all components in the causal ordering. Otherwise, `confounding` must have the same length as `causal_ordering`, indicating the confounding assumption for each component. When `confounding` is specified, `explain` computes asymmetric/symmetric causal Shapley values, depending on `asymmetric`. The `approach` cannot be `regression_separate` or `regression_surrogate`, as the regression-based approaches are not applicable to the causal Shapley methodology.

`extra_computation_args`
Named list. Specifies extra arguments related to the computation of the Shapley values. See the help file of `get_extra_comp_args_default()` for description of the arguments and their default values.

`iterative_args` Named list. Specifies the arguments for the iterative procedure. See the help file of `get_iterative_args_default()` for description of the arguments and their default values.

`output_args` Named list. Specifies certain arguments related to the output of the function. See the help file of `get_output_args_default()` for description of the arguments and their default values.

`scope` String. Either `"local"` (default) or `"global"`. If `"local"`, `explain` computes standard (local) Shapley values that explain individual predictions, i.e. SHAP (Shapley Additive exPlanations)-style explanations. If `"global"`, `explain` instead computes SAGE values (Shapley Additive Global importancE), which explain the global model loss over the observations in `x_explain` rather than individual predictions. See the details section and `vignette("general_usage", package = "shapr")` for more information.

`y_explain` Numeric vector. Only used (and required) when `scope = "global"`. The true response/outcome values corresponding to the observations in `x_explain`, used to evaluate the model loss when computing the SAGE values. Must be numeric and have the same number of elements as there are rows in `x_explain`.

`...` Arguments passed on to `setup_approach`

`arf.num_trees` Positive integer. The number of trees in the adversarial random forest.

`arf.min_node_size` Positive integer. The minimum number of observations in each terminal node.

`arf.delta` Non-negative numeric scalar. Tuning parameter passed to `arf::adversarial_rf()`.

`arf.max_iters` Positive integer. The maximum number of adversarial forest iterations.

`arf.alpha` Numeric scalar between 0 and 1. Tuning parameter passed to `arf::forde()`.

`arf.epsilon` Positive numeric scalar. Small regularization constant passed to `arf::forde()`.

`arf.parallel_train` Logical scalar. If `TRUE`, `arf::adversarial_rf()` and `arf::forde()` use parallel processing when training the feature distribution. The training step uses `ranger` threads, controlled by `options(ranger.num.threads = n)` or the `R_RANGER_NUM_THREADS` environment variable. The `arf::forde()` step uses `foreach` and only runs in parallel when a `foreach` backend is registered.

`arf.parallel_gen` Logical scalar. If `TRUE`, `arf::forge()` uses `foreach` parallel processing when generating samples for the Monte Carlo integration used to estimate `v(S)`. This can be faster for ARF-heavy explanations, but is best combined with sequential shapr batching, for example `extra_computation_args = list(vS_batching_method = "forloop", min_n_batches = 1, max_batch_size = Inf)`, to avoid nested parallelization with `future`.

`categorical.joint_prob_dt` Data.table. (Optional) Containing the joint probability distribution for each combination of feature values. `NULL` means it is estimated from the `x_train` and `x_explain`.

`categorical.epsilon` Numeric value. (Optional) If `categorical.joint_prob_dt` is not supplied, probabilities/frequencies are estimated using `x_train`. If certain observations occur in `x_explain` and NOT in `x_train`, then epsilon is used as the proportion of times that these observations occur in the training data. In theory, this proportion should be zero, but this causes an error later in the Shapley computation.

`ctree.mincriterion` Numeric scalar or vector. Either a scalar or vector of length equal to the number of features in the model. The value is equal to 1 - α where α is the nominal level of the conditional independence tests. If it is a vector, this indicates which value to use when conditioning on various numbers of features. The default value is 0.95.

`ctree.minsplit` Numeric scalar. Determines the minimum value that the sum of the left and right daughter nodes must reach for a split. The default value is 20.

`ctree.minbucket` Numeric scalar. Determines the minimum sum of weights in a terminal node required for a split. The default value is 7.

`ctree.sample` Boolean. If `TRUE` (default), then the method always samples `n_MC_samples` observations from the leaf nodes (with replacement). If `FALSE` and the number of observations in the leaf node is less than `n_MC_samples`, the method will take all observations in the leaf. If `FALSE` and the number of observations in the leaf node is more than `n_MC_samples`, the method will sample `n_MC_samples` observations (with replacement). This means that there will always be sampling in the leaf unless `sample = FALSE` *and* the number of obs in the node is less than `n_MC_samples`.

`empirical.type` Character. Must be one of `"fixed_sigma"` (default), `"AICc_each_k"`, `"AICc_full"` or `"independence"`. Note: `"empirical.type = independence"` is deprecated; use `approach = "independence"` instead. `"fixed_sigma"` uses a fixed bandwidth (set through `empirical.fixed_sigma`) in the kernel density estimation. `"AICc_each_k"` and `"AICc_full"` optimize the bandwidth using the AICc criterion, with respectively one bandwidth per coalition size and one bandwidth for all coalition sizes.

`empirical.eta` Numeric scalar. Needs to be `0 < empirical.eta <= 1`. The default value is 0.95. Represents the minimum proportion of the total empirical weight that data samples should use. For example, if `empirical.eta = .8`, we choose the `K` samples with the largest weights so that the sum of the weights accounts for 80% of the total weight. `empirical.eta` is the η parameter in equation (15) of Aas et al. (2021).

`empirical.fixed_sigma` Positive numeric scalar. The default value is 0.1. Represents the kernel bandwidth in the distance computation used when conditioning on all different coalitions. Only used when `empirical.type = "fixed_sigma"`

`empirical.n_samples_aicc` Positive integer. Number of samples to consider in AICc optimization. The default value is 1000. Only used when `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

`empirical.eval_max_aicc` Positive integer. Maximum number of iterations when optimizing the AICc. The default value is 20. Only used when `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

`empirical.start_aicc` Numeric. Start value of the `sigma` parameter when optimizing the AICc. The default value is 0.1. Only used when `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

`empirical.cov_mat` Numeric matrix. The covariance matrix of the data generating distribution used to define the Mahalanobis distance. `NULL` means it is estimated from `x_train`.

`gaussian.mu` Numeric vector. Containing the mean of the data generating distribution. `NULL` means it is estimated from the `x_train`.

`gaussian.cov_mat` Numeric matrix. Containing the covariance matrix of the data generating distribution. `NULL` means it is estimated from the `x_train`.

`regression.model` A `tidymodels` object of class `model_specs`. Default is a linear regression model, i.e., `parsnip::linear_reg()`. See tidymodels for all possible models, and see the vignette for how to add new/own models. Note, to make it easier to call `explain()` from Python, the `regression.model` parameter can also be a string specifying the model which will be parsed and evaluated. For example, `"parsnip::rand_forest(mtry = hardhat::tune(), trees = 100, `
is also a valid input. It is essential to include the package prefix if the package is not loaded.

`regression.tune_values` Either `NULL` (default), a data.frame/data.table/tibble, or a function. The data.frame must contain the possible hyperparameter value combinations to try. The column names must match the names of the tunable parameters specified in `regression.model`. If `regression.tune_values` is a function, then it should take one argument `x` which is the training data for the current coalition and returns a data.frame/data.table/tibble with the

properties described above. Using a function allows the hyperparameter values to change based on the size of the coalition See the regression vignette for several examples. Note, to make it easier to call `explain()` from Python, the `regression.tune_values` can also be a string containing an R function. For example, `"function(x) return(dials::grid_regular(dials::mtry(c(1, ncol(x))), levels = 3))"` is also a valid input. It is essential to include the package prefix if the package is not loaded.

`regression.vfold_cv_para` Either `NULL` (default) or a named list containing the parameters to be sent to `rsample::vfold_cv()`. See the regression vignette for several examples.

`regression.recipe_func` Either `NULL` (default) or a function that that takes in a `recipes::recipe()` object and returns a modified `recipes::recipe()` with potentially additional recipe steps. See the regression vignette for several examples. Note, to make it easier to call `explain()` from Python, the `regression.recipe_func` can also be a string containing an R function. For example, `"function(recipe) return(recipes::step_ns(recipe, recipes::all_numeric_predictors(), deg_free = 2))"` is also a valid input. It is essential to include the package prefix if the package is not loaded.

`regression.surrogate_n_comb` Positive integer. Specifies the number of unique coalitions to apply to each training observation. The default is the number of sampled coalitions in the present iteration. Any integer between 1 and the default is allowed. Larger values requires more memory, but may improve the surrogate model. If the user sets a value lower than the maximum, we sample this amount of unique coalitions separately for each training observations. That is, on average, all coalitions should be equally trained.

`timeseries.fixed_sigma` Positive numeric scalar. Represents the kernel bandwidth in the distance computation. The default value is 2.

`timeseries.bounds` Numeric vector of length two. Specifies the lower and upper bounds of the timeseries. The default is `c(NULL, NULL)`, i.e. no bounds. If one or both of these bounds are not `NULL`, we restrict the sampled time series to be between these bounds. This is useful if the underlying time series are scaled between 0 and 1, for example.

`vaeac.depth` Positive integer (default is `3`). The number of hidden layers in the neural networks of the masked encoder, full encoder, and decoder.

`vaeac.width` Positive integer (default is `32`). The number of neurons in each hidden layer in the neural networks of the masked encoder, full encoder, and decoder.

`vaeac.latent_dim` Positive integer (default is `8`). The number of dimensions in the latent space.

`vaeac.lr` Positive numeric (default is `0.001`). The learning rate used in the `torch::optim_adam()` optimizer.

`vaeac.activation_function` An `torch::nn_module()` representing an activation function such as, e.g., `torch::nn_relu()` (default), `torch::nn_leaky_relu()`, `torch::nn_selu()`, or `torch::nn_sigmoid()`.

`vaeac.n_vaeacs_initialize` Positive integer (default is `4`). The number of different vaeac models to initiate in the start. Pick the best performing

one after vaeac.extra_parameters$epochs_initiation_phase epochs (default is 2) and continue training that one.

vaeac.epochs Positive integer (default is 100). The number of epochs to train the final vaeac model. This includes vaeac.extra_parameters$epochs_initiation_phase, where the default is 2.

vaeac.extra_parameters Named list with extra parameters to the vaeac approach. See vaeac_get_extra_para_default() for description of possible additional parameters and their default values.

Details

The shapr package implements kernelSHAP estimation of dependence-aware Shapley values with nine different Monte Carlo-based approaches for estimating the conditional distributions of the data. These are all introduced in the general usage vignette. (From R: vignette("general_usage", package = "shapr")). For an overview of the methodology and capabilities of the package, please also see the software paper Jullum et al. (2025). Moreover, Aas et al. (2021) gives a general introduction to dependence-aware Shapley values and the approaches "empirical", "gaussian", "copula", and also discusses "independence". Redelmeier et al. (2020) introduces the approach "ctree". Olsen et al. (2022) introduces the "vaeac" approach. The "arf" approach uses adversarial random forests through the arf package. Approach "timeseries" is discussed in Jullum et al. (2021). shapr has also implemented two regression-based approaches "regression_separate" and "regression_surrogate", as described in Olsen et al. (2024). It is also possible to combine the different approaches, see the general usage vignette for more information.

The package also supports the computation of causal and asymmetric Shapley values as introduced by Heskes et al. (2020) and Frye et al. (2020). Asymmetric Shapley values were proposed by Frye et al. (2020) as a way to incorporate causal knowledge in the real world by restricting the possible feature combinations/coalitions when computing the Shapley values to those consistent with a (partial) causal ordering. Causal Shapley values were proposed by Heskes et al. (2020) as a way to explain the total effect of features on the prediction, taking into account their causal relationships, by adapting the sampling procedure in shapr.

When scope = "global", explain computes SAGE values (Shapley Additive Global importancE) as introduced by Covert et al. (2020). Rather than explaining individual predictions, SAGE values explain the global model loss by attributing the reduction in loss (relative to always predicting the baseline phi0) to each feature. The loss function is controlled via extra_computation_args$global_loss_func. shapr reuses the exact same machinery as for regular Shapley values, but replaces the value function v(S) with the negative expected loss -E[loss(y, E[f(x) | x_S])], averaged over the observations in x_explain. The conditional expectations are estimated with the chosen approach, so unlike the marginal sampling used by Covert et al. (2020), shapr can account for feature dependence. A single set of SAGE values is returned in shapley_values_est, with the corresponding standard deviations in shapley_values_sd. The regular per-observation Shapley value explanations of the predictions are always also computed and can be accessed with get_results(x, "shap_values_est"), while get_results(x, "sage_values_est") returns the SAGE values.

Because SAGE reuses the regular Shapley value machinery (only the value function is replaced), scope = "global" also works together with grouping (group), causal Shapley values (causal_ordering/confounding), and asymmetric Shapley values (asymmetric). See vignette("general_usage", package = "shapr") for details.

The package allows parallelized computation with progress updates through the tightly connected

future::future and progressr::progressr packages. The vaeac approach relies on torch external pointers that cannot be exported to separate R processes, so future multisession and cluster plans with multiple workers are unsupported. The only parallel option is future::plan(future::multicore) (unavailable on Windows and within RStudio). For sequential computation, use future::plan(future::sequential) or set extra_computation_args = list(vS_batching_method = "forloop"). See the examples below. For iterative estimation (iterative=TRUE), intermediate results may be printed to the console (according to the verbose argument). Moreover, the intermediate results are written to disk. This combined batch computation of the v(S) values enables fast and accurate estimation of the Shapley values in a memory-friendly manner.

Value

Object of class c("shapr", "list"). Contains the following items:

shapley_values_est data.table with the estimated Shapley values with explained observation in the rows and features along the columns. The column none is the prediction not devoted to any of the features (given by the argument phi0). If scope = "global", this instead contains a single row with the estimated SAGE values, and the column none gives the baseline model loss -loss(y_explain, phi0).

shapley_values_sd data.table with the standard deviation of the Shapley values reflecting the uncertainty in the coalition sampling part of the kernelSHAP procedure. These are, by definition, 0 when all coalitions are used. Only present when extra_computation_args$compute_sd=TRUE, which is the default when iterative = TRUE. If scope = "global", this contains a single row with the standard deviations of the SAGE values.

internal List with the different parameters, data, functions and other output used internally.

pred_explain Numeric vector with the predictions for the explained observations.

MSEv List with the values of the MSEv evaluation criterion for the approach. See the MSEv evaluation section in the general usage vignette for details.

timing List containing timing information for the different parts of the computation. summary contains the time stamps for the start and end time in addition to the total execution time. overall_timing_secs gives the time spent on different parts of the explanation computation. main_computation_timing_secs further decomposes the main computation time into different parts of the computation for each iteration of the iterative estimation routine, if used.

Author(s)

Martin Jullum, Lars Henry Berge Olsen

References

- Jullum, M., Olsen, L. H. B., Lachmann, J., & Redelmeier, A. (2025). shapr: Explaining Machine Learning Models with Conditional Shapley Values in R and Python. arXiv preprint arXiv:2504.01842.
- Aas, K., Jullum, M., & Løland, A. (2021). Explaining individual predictions when features are dependent: More accurate approximations to Shapley values. Artificial Intelligence, 298, 103502

- Frye, C., Rowat, C., & Feige, I. (2020). Asymmetric Shapley values: incorporating causal knowledge into model-agnostic explainability. Advances in neural information processing systems, 33, 1229-1239
- Heskes, T., Sijben, E., Bucur, I. G., & Claassen, T. (2020). Causal shapley values: Exploiting causal knowledge to explain individual predictions of complex models. Advances in neural information processing systems, 33, 4778-4789
- Jullum, M., Redelmeier, A. & Aas, K. (2021). Efficient and simple prediction explanations with groupShapley: A practical perspective. Italian Workshop on Explainable Artificial Intelligence 2021.
- Redelmeier, A., Jullum, M., & Aas, K. (2020). Explaining predictive models with mixed features using Shapley values and conditional inference trees. In Machine Learning and Knowledge Extraction: International Cross-Domain Conference, CD-MAKE 2020, Dublin, Ireland, August 25-28, 2020, Proceedings 4 (pp. 117-137). Springer International Publishing.
- Sellereite N., & Jullum, M. (2019). shapr: An R-package for explaining machine learning models with dependence-aware Shapley values. Journal of Open Source Software, 5(46), 2027
- Olsen, L. H., Glad, I. K., Jullum, M., & Aas, K. (2022). Using Shapley values and variational autoencoders to explain predictive models with dependent mixed features. Journal of machine learning research, 23(213), 1-51
- Olsen, L. H. B., Glad, I. K., Jullum, M., & Aas, K. (2024). A comparative study of methods for estimating model-agnostic Shapley value explanations. Data Mining and Knowledge Discovery, 1-48
- Olsen, L. H. B., & Jullum, M. (2025). Improving the Weighting Strategy in KernelSHAP. In World Conference on Explainable Artificial Intelligence (pp. 194-218). Springer.

Examples

```
# Load example data
data("airquality")
airquality <- airquality[complete.cases(airquality), ]
x_var <- c("Solar.R", "Wind", "Temp", "Month")
y_var <- "Ozone"

# Split data into test and training data
data_train <- head(airquality, -3)
data_explain <- tail(airquality, 3)

x_train <- data_train[, x_var]
x_explain <- data_explain[, x_var]

# Fit a linear model
lm_formula <- as.formula(paste0(y_var, " ~ ", paste0(x_var, collapse = " + ")))
model <- lm(lm_formula, data = data_train)

# Explain predictions
p <- mean(data_train[, y_var])
```

```
# (Optionally) enable parallelization via the future package
if (requireNamespace("future", quietly = TRUE)) {
 # For vaeac, use future::multicore where available (not Windows or RStudio), or run sequentially.
  future::plan("multisession", workers = 2)
}

# (Optionally) enable progress updates within every iteration via the progressr package
if (requireNamespace("progressr", quietly = TRUE)) {
  progressr::handlers(global = TRUE)
}

# Empirical approach
explain1 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "empirical",
  phi0 = p,
  n_MC_samples = 1e2
)

# Gaussian approach
explain2 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "gaussian",
  phi0 = p,
  n_MC_samples = 1e2
)

# Gaussian copula approach
explain3 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "copula",
  phi0 = p,
  n_MC_samples = 1e2
)

if (requireNamespace("party", quietly = TRUE)) {
  # ctree approach
  explain4 <- explain(
    model = model,
    x_explain = x_explain,
    x_train = x_train,
    approach = "ctree",
    phi0 = p,
    n_MC_samples = 1e2
  )
}
```

```
# Combined approach
approach <- c("gaussian", "gaussian", "empirical")
explain5 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = approach,
  phi0 = p,
  n_MC_samples = 1e2
)

## Printing
print(explain1) # The Shapley values
print(explain1) # The Shapley values

# The MSEv criterion (+sd). Smaller values indicate a better approach.
print(explain1, what = "MSEv")
print(explain2, what = "MSEv")
print(explain3, what = "MSEv")

## Summary
summary1 <- summary(explain1)
summary1 # Provides a nicely formatted summary of the explanation

# Various additional info stored in the summary object
# Examples
summary1$shapley_est # A data.table with the Shapley values
summary1$timing$total_time_secs # Total computation time in seconds
summary1$parameters$n_MC_samples # Number of Monte Carlo samples used for the numerical integration
summary1$parameters$empirical.type # Type of empirical approach used

# Plot the results
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot(explain1)
  plot(explain1, plot_type = "waterfall")
}

# Group-wise explanations
group_list <- list(A = c("Temp", "Month"), B = c("Wind", "Solar.R"))

explain_groups <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  group = group_list,
  approach = "empirical",
  phi0 = p,
  n_MC_samples = 1e2
)

print(explain_groups)

# Separate and surrogate regression approaches with linear regression models.
```

```
req_pkgs <- c("parsnip", "recipes", "workflows", "rsample", "tune", "yardstick")
if (requireNamespace(req_pkgs, quietly = TRUE)) {
  explain_separate_lm <- explain(
    model = model,
    x_explain = x_explain,
    x_train = x_train,
    phi0 = p,
    approach = "regression_separate",
    regression.model = parsnip::linear_reg()
  )

  explain_surrogate_lm <- explain(
    model = model,
    x_explain = x_explain,
    x_train = x_train,
    phi0 = p,
    approach = "regression_surrogate",
    regression.model = parsnip::linear_reg()
  )
}

# Iterative estimation
# For illustration only. By default not used for such small dimensions as here.
# Restricting the initial and maximum number of coalitions as well.

explain_iterative <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "gaussian",
  phi0 = p,
  iterative = TRUE,
  iterative_args = list(initial_n_coalitions = 8),
  max_n_coalitions = 12
)

# When not using all coalitions, we can also get the SD of the Shapley values,
# reflecting uncertainty in the coalition sampling part of the procedure.
print(explain_iterative, what = "shapley_sd")

## Summary
# For iterative estimation, convergence info is also provided
summary_iterative <- summary(explain_iterative)
```

`explain_forecast` *Explain a Forecast from Time Series Models with Dependence-Aware (Conditional/Observational) Shapley Values*

Description

Computes dependence-aware Shapley values for observations in `explain_idx` from the specified `model` by using the method specified in `approach` to estimate the conditional expectation. See Aas, et. al (2021) for a thorough introduction to dependence-aware prediction explanation with Shapley values. For an overview of the methodology and capabilities of the `shapr` package, see the software paper Jullum et al. (2025), or the pkgdown site at norskregnesentral.github.io/shapr/.

Usage

```
explain_forecast(
  model,
  y,
  xreg = NULL,
  train_idx = NULL,
  explain_idx,
  explain_y_lags,
  explain_xreg_lags = explain_y_lags,
  horizon,
  approach,
  phi0,
  max_n_coalitions = NULL,
  iterative = NULL,
  group_lags = TRUE,
  group = NULL,
  n_MC_samples = 1000,
  seed = NULL,
  predict_model = NULL,
  get_model_specs = NULL,
  verbose = "basic",
  extra_computation_args = list(),
  iterative_args = list(),
  output_args = list(),
  ...
)
```

Arguments

`model`	Model object. The model whose predictions you want to explain. Run `get_supported_models()` for a table of which models `explain` supports natively. Unsupported models can still be explained by passing `predict_model` and (optionally) `get_model_specs`, see details for more information.
`y`	Matrix, data.frame/data.table or a numeric vector. Contains the endogenous variables used to estimate the (conditional) distributions needed to properly estimate the conditional expectations in the Shapley formula including the observations to be explained.
`xreg`	Matrix, data.frame/data.table or a numeric vector. Contains the exogenous variables used to estimate the (conditional) distributions needed to properly estimate the conditional expectations in the Shapley formula including the observations

to be explained. As exogenous variables are used contemporaneously when producing a forecast, this item should contain nrow(y) + horizon rows.

`train_idx` Numeric vector. The row indices in data and reg denoting points in time to use when estimating the conditional expectations in the Shapley value formula. If `train_idx = NULL` (default) all indices not selected to be explained will be used.

`explain_idx` Numeric vector. The row indices in data and reg denoting points in time to explain.

`explain_y_lags` Numeric vector. Denotes the number of lags that should be used for each variable in `y` when making a forecast.

`explain_xreg_lags`
Numeric vector. If `xreg != NULL`, denotes the number of lags that should be used for each variable in `xreg` when making a forecast.

`horizon` Numeric. The forecast horizon to explain. Passed to the `predict_model` function.

`approach` Character vector of length `1` or one less than the number of features. All elements should either be `"arf"`, `"categorical"`, `"copula"`, `"ctree"`, `"empirical"`, `"gaussian"`, `"independence"`, `"regression_separate"`, `"regression_surrogate"`, `"timeseries"`, or `"vaeac"`. The two regression approaches cannot be combined with any other approach. See details for more information.

`phi0` Numeric. The prediction value for unseen data, i.e., an estimate of the expected prediction without conditioning on any features. Typically set this equal to the mean of the response in the training data, but alternatives such as the mean of the training predictions are also reasonable.

`max_n_coalitions`
Integer. Upper limit on the number of unique feature/group coalitions to use in the iterative procedure (if `iterative = TRUE`). If `iterative = FALSE`, it represents the number of feature/group coalitions to use directly. The quantity refers to the number of unique feature coalitions if `group = NULL`, and group coalitions if `group != NULL`. `max_n_coalitions = NULL` corresponds to `2^n_features`.

`iterative` Logical or NULL. If `NULL` (default), set to `TRUE` if there are more than 5 features/groups, and `FALSE` otherwise. If `TRUE`, Shapley values are estimated iteratively for faster, sufficiently accurate results. First an initial number of coalitions is sampled, then bootstrapping estimates the variance of the Shapley values. A convergence criterion determines if the variances are sufficiently small. If not, additional samples are added. The process repeats until the variances are below the threshold. Specifics for the iterative process and convergence criterion are set via `iterative_args`.

`group_lags` Logical. If `TRUE` all lags of each variable are grouped together and explained as a group. If `FALSE` all lags of each variable are explained individually.

`group` List. If `NULL`, regular feature-wise Shapley values are computed. If provided, group-wise Shapley values are computed. `group` then has length equal to the number of groups. Each list element contains the character vectors with the features included in the corresponding group. See Jullum et al. (2021) for more information on group-wise Shapley values.

`n_MC_samples` Positive integer. For most approaches, it indicates the maximum number of samples to use in the Monte Carlo integration of every conditional expectation. For `approach="ctree"`, `n_MC_samples` corresponds to the number of samples from the leaf node (see an exception related to the `ctree.sample` argument in `setup_approach.ctree()`). For `approach="empirical"`, `n_MC_samples` is the K parameter in equations (14-15) of Aas et al. (2021), i.e. the maximum number of observations (with largest weights) that is used, see also the `empirical.eta` argument `setup_approach.empirical()`.

`seed` Positive integer. Specifies the seed before any code involving randomness is run. If `NULL` (default), no seed is set in the calling environment.

`predict_model` Function. Prediction function to use when `model` is not natively supported. (Run `get_supported_models()` for a list of natively supported models.) The function must have two arguments, `model` and `newdata`, which specify the model and a data.frame/data.table to compute predictions for, respectively. The function must give the prediction as a numeric vector. `NULL` (the default) uses functions specified internally. Can also be used to override the default function for natively supported model classes.

`get_model_specs` Function. An optional function for checking model/data consistency when `model` is not natively supported. (Run `get_supported_models()` for a list of natively supported models.) The function takes `model` as an argument and provides a list with 3 elements:

labels Character vector with the names of each feature.

classes Character vector with the class of each feature.

factor_levels Character vector with the levels for any categorical features.

If `NULL` (the default), internal functions are used for natively supported model classes, and checking is disabled for unsupported model classes. Can also be used to override the default function for natively supported model classes.

`verbose` String vector or NULL. Controls verbosity (printout detail level) via one or more of `"basic"`, `"progress"`, `"convergence"`, `"shapley"` and `"vS_details"`. `"basic"` (default) displays basic information about the computation and messages about parameters/checks. `"progress"` displays where in the calculation process the function currently is. `"convergence"` displays how close the Shapley value estimates are to convergence (only when `iterative = TRUE`). `"shapley"` displays intermediate Shapley value estimates and standard deviations (only when `iterative = TRUE`), and the final estimates. `"vS_details"` displays information about the v(S) estimates, most relevant for `approach %in% c("regression_separate", "regression_surrogate", "vaeac")`. `NULL` means no printout. Any combination can be used, e.g., `verbose = c("basic", "vS_details")`.

`extra_computation_args` Named list. Specifies extra arguments related to the computation of the Shapley values. See the help file of `get_extra_comp_args_default()` for description of the arguments and their default values.

`iterative_args` Named list. Specifies the arguments for the iterative procedure. See the help file of `get_iterative_args_default()` for description of the arguments and their default values.

output_args Named list. Specifies certain arguments related to the output of the function. See the help file of get_output_args_default() for description of the arguments and their default values.

... Arguments passed on to setup_approach

arf.num_trees Positive integer. The number of trees in the adversarial random forest.

arf.min_node_size Positive integer. The minimum number of observations in each terminal node.

arf.delta Non-negative numeric scalar. Tuning parameter passed to arf::adversarial_rf().

arf.max_iters Positive integer. The maximum number of adversarial forest iterations.

arf.alpha Numeric scalar between 0 and 1. Tuning parameter passed to arf::forde().

arf.epsilon Positive numeric scalar. Small regularization constant passed to arf::forde().

arf.parallel_train Logical scalar. If TRUE, arf::adversarial_rf() and arf::forde() use parallel processing when training the feature distribution. The training step uses ranger threads, controlled by options(ranger.num.threads = n) or the R_RANGER_NUM_THREADS environment variable. The arf::forde() step uses foreach and only runs in parallel when a foreach backend is registered.

arf.parallel_gen Logical scalar. If TRUE, arf::forge() uses foreach parallel processing when generating samples for the Monte Carlo integration used to estimate v(S). This can be faster for ARF-heavy explanations, but is best combined with sequential shapr batching, for example extra_computation_args = list(vS_batching_method = "forloop", min_n_batches = 1, max_batch_size = Inf), to avoid nested parallelization with future.

categorical.joint_prob_dt Data.table. (Optional) Containing the joint probability distribution for each combination of feature values. NULL means it is estimated from the x_train and x_explain.

categorical.epsilon Numeric value. (Optional) If categorical.joint_prob_dt is not supplied, probabilities/frequencies are estimated using x_train. If certain observations occur in x_explain and NOT in x_train, then epsilon is used as the proportion of times that these observations occur in the training data. In theory, this proportion should be zero, but this causes an error later in the Shapley computation.

ctree.mincriterion Numeric scalar or vector. Either a scalar or vector of length equal to the number of features in the model. The value is equal to 1 - α where α is the nominal level of the conditional independence tests. If it is a vector, this indicates which value to use when conditioning on various numbers of features. The default value is 0.95.

ctree.minsplit Numeric scalar. Determines the minimum value that the sum of the left and right daughter nodes must reach for a split. The default value is 20.

ctree.minbucket Numeric scalar. Determines the minimum sum of weights in a terminal node required for a split. The default value is 7.

`ctree.sample` Boolean. If `TRUE` (default), then the method always samples `n_MC_samples` observations from the leaf nodes (with replacement). If `FALSE` and the number of observations in the leaf node is less than `n_MC_samples`, the method will take all observations in the leaf. If `FALSE` and the number of observations in the leaf node is more than `n_MC_samples`, the method will sample `n_MC_samples` observations (with replacement). This means that there will always be sampling in the leaf unless `sample = FALSE` *and* the number of obs in the node is less than `n_MC_samples`.

`empirical.type` Character. Must be one of `"fixed_sigma"` (default), `"AICc_each_k"`, `"AICc_full"` or `"independence"`. Note: `"empirical.type = independence"` is deprecated; use `approach = "independence"` instead. `"fixed_sigma"` uses a fixed bandwidth (set through `empirical.fixed_sigma`) in the kernel density estimation. `"AICc_each_k"` and `"AICc_full"` optimize the bandwidth using the AICc criterion, with respectively one bandwidth per coalition size and one bandwidth for all coalition sizes.

`empirical.eta` Numeric scalar. Needs to be `0 < empirical.eta <= 1`. The default value is 0.95. Represents the minimum proportion of the total empirical weight that data samples should use. For example, if `empirical.eta = .8`, we choose the `K` samples with the largest weights so that the sum of the weights accounts for 80% of the total weight. `empirical.eta` is the η parameter in equation (15) of Aas et al. (2021).

`empirical.fixed_sigma` Positive numeric scalar. The default value is 0.1. Represents the kernel bandwidth in the distance computation used when conditioning on all different coalitions. Only used when `empirical.type = "fixed_sigma"`

`empirical.n_samples_aicc` Positive integer. Number of samples to consider in AICc optimization. The default value is 1000. Only used when `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

`empirical.eval_max_aicc` Positive integer. Maximum number of iterations when optimizing the AICc. The default value is 20. Only used when `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

`empirical.start_aicc` Numeric. Start value of the `sigma` parameter when optimizing the AICc. The default value is 0.1. Only used when `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

`empirical.cov_mat` Numeric matrix. The covariance matrix of the data generating distribution used to define the Mahalanobis distance. `NULL` means it is estimated from `x_train`.

`gaussian.mu` Numeric vector. Containing the mean of the data generating distribution. `NULL` means it is estimated from the `x_train`.

`gaussian.cov_mat` Numeric matrix. Containing the covariance matrix of the data generating distribution. `NULL` means it is estimated from the `x_train`.

`timeseries.fixed_sigma` Positive numeric scalar. Represents the kernel bandwidth in the distance computation. The default value is 2.

`timeseries.bounds` Numeric vector of length two. Specifies the lower and upper bounds of the timeseries. The default is `c(NULL, NULL)`, i.e. no bounds. If one or both of these bounds are not `NULL`, we restrict the sampled

time series to be between these bounds. This is useful if the underlying time series are scaled between 0 and 1, for example.

`vaeac.depth` Positive integer (default is `3`). The number of hidden layers in the neural networks of the masked encoder, full encoder, and decoder.

`vaeac.width` Positive integer (default is `32`). The number of neurons in each hidden layer in the neural networks of the masked encoder, full encoder, and decoder.

`vaeac.latent_dim` Positive integer (default is `8`). The number of dimensions in the latent space.

`vaeac.lr` Positive numeric (default is `0.001`). The learning rate used in the `torch::optim_adam()` optimizer.

`vaeac.activation_function` An `torch::nn_module()` representing an activation function such as, e.g., `torch::nn_relu()` (default), `torch::nn_leaky_relu()`, `torch::nn_selu()`, or `torch::nn_sigmoid()`.

`vaeac.n_vaeacs_initialize` Positive integer (default is `4`). The number of different vaeac models to initiate in the start. Pick the best performing one after `vaeac.extra_parameters$epochs_initiation_phase` epochs (default is `2`) and continue training that one.

`vaeac.epochs` Positive integer (default is `100`). The number of epochs to train the final vaeac model. This includes `vaeac.extra_parameters$epochs_initiation_phase`, where the default is `2`.

`vaeac.extra_parameters` Named list with extra parameters to the `vaeac` approach. See `vaeac_get_extra_para_default()` for description of possible additional parameters and their default values.

Details

This function explains a forecast of length `horizon`. The argument `train_idx` is analogous to x_train in `explain()`, however, it just contains the time indices of where in the data the forecast should start for each training sample. In the same way `explain_idx` defines the time index (indices) which will precede a forecast to be explained.

As any autoregressive forecast model will require a set of lags to make a forecast at an arbitrary point in time, `explain_y_lags` and `explain_xreg_lags` define how many lags are required to "refit" the model at any given time index. This allows the different approaches to work in the same way they do for time-invariant models.

See the forecasting section of the general usage vignette for further details. See also the software paper Jullum et al. (2025, Sec. 6) for a more detailed introduction to the methodology, and additional examples.

Value

Object of class `c("shapr", "list")`. Contains the following items:

`shapley_values_est` data.table with the estimated Shapley values with explained observation in the rows and features along the columns. The column `none` is the prediction not devoted to any of the features (given by the argument `phi0`). If `scope = "global"`, this instead contains a single row with the estimated SAGE values, and the column `none` gives the baseline model loss `-loss(y_explain, phi0)`.

`shapley_values_sd` data.table with the standard deviation of the Shapley values reflecting the uncertainty in the coalition sampling part of the kernelSHAP procedure. These are, by definition, 0 when all coalitions are used. Only present when `extra_computation_args$compute_sd=TRUE`, which is the default when `iterative = TRUE`. If `scope = "global"`, this contains a single row with the standard deviations of the SAGE values.

`internal` List with the different parameters, data, functions and other output used internally.

`pred_explain` Numeric vector with the predictions for the explained observations.

`MSEv` List with the values of the MSEv evaluation criterion for the approach. See the MSEv evaluation section in the general usage vignette for details.

`timing` List containing timing information for the different parts of the computation. `summary` contains the time stamps for the start and end time in addition to the total execution time. `overall_timing_secs` gives the time spent on different parts of the explanation computation. `main_computation_timing_secs` further decomposes the main computation time into different parts of the computation for each iteration of the iterative estimation routine, if used.

Author(s)

Jon Lachmann, Martin Jullum

References

- Jullum, M., Olsen, L. H. B., Lachmann, J., & Redelmeier, A. (2025). shapr: Explaining Machine Learning Models with Conditional Shapley Values in R and Python. arXiv preprint arXiv:2504.01842.
- Aas, K., Jullum, M., & Løland, A. (2021). Explaining individual predictions when features are dependent: More accurate approximations to Shapley values. Artificial Intelligence, 298, 103502
- Frye, C., Rowat, C., & Feige, I. (2020). Asymmetric Shapley values: incorporating causal knowledge into model-agnostic explainability. Advances in neural information processing systems, 33, 1229-1239
- Heskes, T., Sijben, E., Bucur, I. G., & Claassen, T. (2020). Causal shapley values: Exploiting causal knowledge to explain individual predictions of complex models. Advances in neural information processing systems, 33, 4778-4789
- Jullum, M., Redelmeier, A. & Aas, K. (2021). Efficient and simple prediction explanations with groupShapley: A practical perspective. Italian Workshop on Explainable Artificial Intelligence 2021.
- Redelmeier, A., Jullum, M., & Aas, K. (2020). Explaining predictive models with mixed features using Shapley values and conditional inference trees. In Machine Learning and Knowledge Extraction: International Cross-Domain Conference, CD-MAKE 2020, Dublin, Ireland, August 25-28, 2020, Proceedings 4 (pp. 117-137). Springer International Publishing.
- Sellereite N., & Jullum, M. (2019). shapr: An R-package for explaining machine learning models with dependence-aware Shapley values. Journal of Open Source Software, 5(46), 2027
- Olsen, L. H., Glad, I. K., Jullum, M., & Aas, K. (2022). Using Shapley values and variational autoencoders to explain predictive models with dependent mixed features. Journal of machine learning research, 23(213), 1-51

- Olsen, L. H. B., Glad, I. K., Jullum, M., & Aas, K. (2024). A comparative study of methods for estimating model-agnostic Shapley value explanations. Data Mining and Knowledge Discovery, 1-48
- Olsen, L. H. B., & Jullum, M. (2025). Improving the Weighting Strategy in KernelSHAP. In World Conference on Explainable Artificial Intelligence (pp. 194-218). Springer.

Examples

```
# Load example data
data("airquality")
data <- data.table::as.data.table(airquality)

# Fit an AR(2) model.
model_ar_temp <- ar(data$Temp, order = 2)

# Calculate the zero prediction values for a three step forecast.
p0_ar <- rep(mean(data$Temp), 3)

# Empirical approach, explaining forecasts starting at T = 152 and T = 153.
explain_forecast(
  model = model_ar_temp,
  y = data[, "Temp"],
  train_idx = 2:151,
  explain_idx = 152:153,
  explain_y_lags = 2,
  horizon = 3,
  approach = "empirical",
  phi0 = p0_ar,
  group_lags = FALSE
)
```

---

get_extra_comp_args_default

Get the Default Values for the Extra Computation Arguments

Description

Get the Default Values for the Extra Computation Arguments

Usage

```
get_extra_comp_args_default(
  internal,
  paired_shap_sampling = isFALSE(internal$parameters$asymmetric),
  semi_deterministic_sampling = FALSE,
  kernelSHAP_reweighting = "on_all_cond",
  compute_sd = isFALSE(internal$parameters$exact),
```

```
  n_boot_samps = 100,
  vS_batching_method = "future",
  max_batch_size = 10,
  min_n_batches = 10,
  max_batch_cube_size = 1e+06,
  global_loss_func = NULL
)
```

Arguments

`internal`	List. Not used directly, but passed through from `explain()`.
`paired_shap_sampling`	Logical. If `TRUE` paired versions of all sampled coalitions are also included in the computation. That is, if there are 5 features and e.g. coalitions (1,3,5) are sampled, then also coalition (2,4) is used for computing the Shapley values. This is done to reduce the variance of the Shapley value estimates. `TRUE` is the default and is recommended for highest accuracy. For asymmetric, `FALSE` is the default and the only legal value.
`semi_deterministic_sampling`	Logical. If `FALSE` (default), then we sample from all coalitions. If `TRUE`, the sampling of coalitions is semi-deterministic, i.e. the sampling is done in a way that ensures that coalitions that are expected to be sampled based on the number of coalitions are deterministically included such that we sample among fewer coalitions. This is done to reduce the variance of the Shapley value estimates, and corresponds to the PySHAP* strategy in the paper Olsen & Jullum (2025).
`kernelSHAP_reweighting`	String. How to reweight the sampling frequency weights in the kernelSHAP solution after sampling. The aim of this is to reduce the randomness and thereby the variance of the Shapley value estimates. The options are one of `'none'`, `'on_N'`, `'on_all'`, `'on_all_cond'` (default). `'none'` means no reweighting, i.e. the sampling frequency weights are used as is. `'on_N'` means the sampling frequencies are averaged over all coalitions with the same original sampling probabilities. `'on_all'` means the original sampling probabilities are used for all coalitions. `'on_all_cond'` means the original sampling probabilities are used for all coalitions, while adjusting for the probability that they are sampled at least once. `'on_all_cond'` is preferred as it performs the best in simulation studies, see Olsen & Jullum (2025).
`compute_sd`	Logical. Whether to estimate the standard deviations of the Shapley value estimates. This is TRUE whenever sampling based kernelSHAP is applied (either iteratively or with a fixed number of coalitions).
`n_boot_samps`	Integer. The number of bootstrapped samples (i.e. samples with replacement) from the set of all coalitions used to estimate the standard deviations of the Shapley value estimates.
`vS_batching_method`	String. The method used to perform batch computing of vS. `"future"` (default), utilizes future.apply::future_apply (via the future::future package), enabling parallelized computation and progress updates via progressr::progressr.

Alternatively, "forloop" can be used for straightforward sequential computation, which is mainly useful for package development and debugging purposes.

max_batch_size Integer. The maximum number of coalitions to estimate simultaneously within each iteration. A larger number requires more memory, but may have a slight computational advantage.

min_n_batches Integer. The minimum number of batches to split the computation into within each iteration. Larger numbers give more frequent progress updates. If parallelization is applied, this should be set no smaller than the number of parallel workers.

max_batch_cube_size
Numeric. The largest number of elements allowed in the dense per-batch array built by the gaussian, copula and empirical approaches. For gaussian and copula this array has a total of n_MC_samples * n_explain * coalitions_per_batch * n_features elements, while for empirical it is the distance array with n_train * n_explain * coalitions_per_batch elements. When a batch would exceed this, the batch size is automatically reduced (i.e. more batches are used) and a message is given. The default 1e6 keeps peak memory modest and tends to reduce runtime in high-dimensional settings, while staying far below the 32-bit indexing limit of the underlying RcppArmadillo arrays (which fails with Cube::init(): requested size is too large). Raise it to allow larger batches, or lower it to use even smaller ones.

global_loss_func
Function or NULL. Only used when scope = "global" (i.e. when computing SAGE values). The loss function used to measure the model loss when computing the SAGE values. Must take two arguments, the true response and the model prediction (in that order), and return a single numeric loss value. If NULL (default), logistic (cross-entropy) loss is used for binary responses (values in 0/1) and mean squared error loss otherwise.

Value

A list with the default values for the extra computation arguments.

Author(s)

Martin Jullum

References

- Olsen, L. H. B., & Jullum, M. (2025). Improving the Weighting Strategy in KernelSHAP. In World Conference on Explainable Artificial Intelligence (pp. 194-218). Springer.

`get_iterative_args_default` *Function to specify arguments of the iterative estimation procedure*

Description

Function to specify arguments of the iterative estimation procedure

Usage

```
get_iterative_args_default(
  internal,
  initial_n_coalitions = ceiling(min(200, max(5, internal$parameters$n_features,
   (2^internal$parameters$n_features)/10), internal$parameters$max_n_coalitions)),
  fixed_n_coalitions_per_iter = NULL,
  max_iter = 20,
  convergence_tol = 0.02,
  n_coal_next_iter_factor_vec = c(seq(0.1, 1, by = 0.1), rep(1, max_iter - 10))
)
```

Arguments

`internal`	List. Not used directly, but passed through from `explain()`.
`initial_n_coalitions`	Integer. Number of coalitions to use in the first estimation iteration.
`fixed_n_coalitions_per_iter`	Integer. Number of `n_coalitions` to use in each iteration. `NULL` (default) means setting it based on estimates based on a set convergence threshold.
`max_iter`	Integer. Maximum number of estimation iterations
`convergence_tol`	Numeric. The t variable in the convergence threshold formula on page 6 in the paper Covert and Lee (2021), 'Improving KernelSHAP: Practical Shapley Value Estimation via Linear Regression' https://arxiv.org/pdf/2012.01536. Smaller values requires more coalitions before convergence is reached.
`n_coal_next_iter_factor_vec`	Numeric vector. The number of `n_coalitions` that must be used to reach convergence in the next iteration is estimated. The number of `n_coalitions` actually used in the next iteration is set to this estimate multiplied by `n_coal_next_iter_factor_vec[i]` for iteration `i`. It is wise to start with smaller numbers to avoid using too many `n_coalitions` due to uncertain estimates in the first iterations.

Details

The functions sets default values for the iterative estimation procedure, according to the function defaults. If the argument `iterative` of `explain()` is FALSE, it sets parameters corresponding to the use of a non-iterative estimation procedure

Value

A list with the default values for the iterative estimation procedure

Author(s)

Martin Jullum

`get_output_args_default` *Get the Default Values for the Output Arguments*

Description

Get the Default Values for the Output Arguments

Usage

```
get_output_args_default(
  keep_samp_for_vS = FALSE,
  MSEv_uniform_comb_weights = TRUE,
  saving_path = tempfile("shapr_obj_", fileext = ".rds")
)
```

Arguments

`keep_samp_for_vS`	Logical. Indicates whether the samples used in the Monte Carlo estimation of `v(S)` should be returned (in `internal$output`). Not used for `approach="regression_separate"` or `approach="regression_surrogate"`.
`MSEv_uniform_comb_weights`	Logical. If `TRUE` (default), then the function weights the coalitions uniformly when computing the MSEv criterion. If `FALSE`, then the function use the Shapley kernel weights to weight the coalitions when computing the MSEv criterion. Note that the Shapley kernel weights are replaced by the sampling frequency when not all coalitions are considered.
`saving_path`	String. The path to the directory where the results of the iterative estimation procedure should be saved. Defaults to a temporary directory.

Value

A list of default output arguments.

Author(s)

Martin Jullum

get_results *Extract Components from a Shapr Object*

Description

Extract Components from a Shapr Object

Usage

```
get_results(x, what = NULL, ...)
```

Arguments

x	A shapr object
what	Character vector specifying one or more components to extract. Options: "calling_function", "proglang", "approach", "scope", "shapley_est", "shapley_sd", "shap_values_est", "sage_values_est", "pred_explain", "MSEv", "MSEv_explicand", "MSEv_coalition", "iterative_info", "iterative_shapley_est", "iterative_shapley_sd", "saving_path", "timing_summary", "timing_details", "parameters", "x_train", "x_explain", "dt_vS", "dt_samp_for_vS", "dt_used_coalitions", "dt_valid_causal_coalitions", "dt_coal_samp_info". The default returns the standard set of components. sage_values_est is never part of the default, since it is NULL for local explanations and identical to shapley_est for global (SAGE) ones. shap_values_est is included only for global explanations, where it holds the distinct per-observation decomposition; for local explanations it would duplicate shapley_est and is omitted. All components remain available on explicit request regardless of scope. See details for what each component contains.
...	Not used

Details

The function extracts a full suite of information related to the computation of the Shapley values from a shapr object. The allowed characters in what provides information as follows:

calling_function Name of function called to create the shapr object, (explain() or explain_forecast()).

proglang Programming language used to initiate the computations (R or Python).

approach Approach used to estimate the conditional expectations.

scope Explanation scope: "local" for standard per-observation Shapley values or "global" for SAGE values.

shapley_est data.table with the estimated Shapley values.

shapley_sd data.table with the standard deviation of the Shapley values reflecting the uncertainty in the coalition sampling part of the kernelSHAP procedure.

shap_values_est data.table with the per-observation (local) Shapley value explanations of the predictions. Always available: when scope = "local" this equals shapley_est; when scope = "global" (SAGE) it holds the per-observation decomposition while shapley_est holds the SAGE values.

`sage_values_est` data.table with the SAGE values (global feature importance). Only available when `scope = "global"` (identical to `shapley_est` in that case); `NULL` otherwise.

`pred_explain` Numeric vector with the predictions for the explained observations.

`MSEv/MSEv_explicand/MSEv_coalition` Data.tables with MSEv evaluation criterion values overall/ per explicand/per coalition. Smaller values indicate better estimates of `v(S)`. See the MSEv evaluation section in the general usage vignette for details.

`iterative_info` Data.table with information about the iterative estimation procedure.

`iterative_shapley_est/iterative_shapley_sd` Data.tables with the estimated Shapley values/their standard deviation for each iteration (when using the iterative estimation procedure).

`saving_path` Character string with the path where the (temporary) results are saved.

`timing_summary` Data.table with one row and three columns: `init_time` and `end_time` give the time stamps for the start and end of the computation, respectively, while `total_time_secs` gives the total time in seconds for the full computation.

`timing_details` List containing timing information for the different parts of the computation. `summary` contains the information from `timing_summary`. `overall_timing_secs` gives the time spent on the different parts of the explanation computation. `main_computation_timing_secs` further decomposes the main computation time into the different parts of the computation for each iteration of the iterative estimation routine, if used.

`parameters` List with the parameters used in the computation.

`x_train/x_explain` Data.tables with the training data used in the computation/observations to explain.

`dt_vS` Data.table with the contribution function (`v(S)`) estimates for each coalition.

`dt_samp_for_vS` Data.table with the samples used in the Monte Carlo estimation of the contribution function (`v(S)`). This is only available if `output_args_default$keep_samp_for_vS = TRUE` (defaults to FALSE) in `explain()`.

`dt_used_coalitions` Data.table with an overview of the coalitions used in the computation.

`dt_valid_causal_coalitions` Data.table with the valid causal coalitions used in the computation.

`dt_coal_samp_info` Data.table with information related to the coalition sampling procedure being used.

Note that the `summary.shapr()` function provides a nicely formatted printout with the most important information, to then invisibly return the output of the present function. The `print.shapr()` allows direct printing of the main results.

Value

If a single component is requested, returns that object. If multiple are requested, returns a named list.

get_supported_approaches *Get the Implemented Approaches*

Description

Get the Implemented Approaches

Usage

```
get_supported_approaches()
```

Value

Character vector. The names of the implemented approaches that can be passed to argument `approach` in `explain()`.

get_supported_models *Provide a* `data.table` *with the Supported Models*

Description

Provide a `data.table` with the Supported Models

Usage

```
get_supported_models()
```

Value

A data.table with the supported models.

plot.shapr *Plot of the Shapley Value Explanations*

Description

Plots the individual prediction explanations.

Usage

```
## S3 method for class 'shapr'
plot(
  x,
  plot_type = "bar",
  digits = 3,
  print_ggplot = TRUE,
  index_x_explain = NULL,
  top_k_features = NULL,
  col = NULL,
  bar_plot_phi0 = TRUE,
  bar_plot_order = "largest_first",
  scatter_features = NULL,
  scatter_hist = TRUE,
  include_group_feature_means = FALSE,
  beeswarm_cex = 1/length(index_x_explain)^(1/4),
  ...
)
```

Arguments

x	An shapr object. The output from explain().
plot_type	Character. Specifies the type of plot to produce. "bar" (the default) gives a regular horizontal bar plot of the Shapley value magnitudes. "waterfall" gives a waterfall plot indicating the changes in the prediction score due to each feature's contribution (their Shapley values). "scatter" plots the feature values on the x-axis and Shapley values on the y-axis, as well as (optionally) a background scatter_hist showing the distribution of the feature data. "beeswarm" summarizes the distribution of the Shapley values along the x-axis for all the features. Each point gives the Shapley value of a given instance, where the points are colored by the feature value of that instance. For SAGE values (scope = "global" in explain()), only "bar" and "waterfall" are supported, as SAGE produces a single global loss-based explanation rather than per-observation prediction explanations.
digits	Integer. Number of significant digits to use in the feature description. Applicable for plot_type "bar" and "waterfall"
print_ggplot	Logical. Whether to print the created ggplot object once it is returned. The default is TRUE which ensures the plot is always displayed also in loops, functions, when sourcing a script, and when assigning the output to a variable like

p <- plot.shapr(...). See ggplot2::print.ggplot() for more details. If you wish to further modify the returned ggplot object outside of plot.shapr, we recommend setting print_ggplot = FALSE to avoid force printing. See the examples for a practical use case.

index_x_explain

Integer vector. Which of the test observations to plot. For example, if you have explained 10 observations using explain(), you can generate a plot for the first five observations by setting index_x_explain = 1:5. Defaults to the first 10 observations for plot_type = "bar" and "waterfall", and to all observations for plot_type = "scatter" and "beeswarm".

top_k_features Integer. How many features to include in the plot. E.g. if you have 15 features in your model you can plot the 5 most important features, for each explanation, by setting top_k_features = 1:5. Applicable for plot_type "bar" and "waterfall"

col Character vector (where length depends on plot type). The color codes (hex codes or other names understood by ggplot2::ggplot()) for positive and negative Shapley values, respectively. The default is col=NULL, plotting with the default colors respective to the plot type. For plot_type = "bar" and plot_type = "waterfall", the default is c("#00BA38","#F8766D"). For plot_type = "beeswarm", the default is c("#F8766D","yellow","#00BA38"). For plot_type = "scatter", the default is "#619CFF".

If you want to alter the colors in the plot, the length of the col vector depends on plot type. For plot_type = "bar" or plot_type = "waterfall", two colors should be provided, first for positive and then for negative Shapley values. For plot_type = "beeswarm", either two or three colors can be given. If two colors are given, then the first color determines the color that points with high feature values will have, and the second determines the color of points with low feature values. If three colors are given, then the first colors high feature values, the second colors mid-range feature values, and the third colors low feature values. For instance, col = c("red", "yellow", "blue") will make high values red, mid-range values yellow, and low values blue. For plot_type = "scatter", a single color is to be given, which determines the color of the points on the scatter plot.

bar_plot_phi0 Logical. Whether to include phi0 in the plot for plot_type = "bar".

bar_plot_order Character. Specifies what order to plot the features with respect to the magnitude of the Shapley values with plot_type = "bar": "largest_first" (the default) plots the features ordered from largest to smallest absolute Shapley value. "smallest_first" plots the features ordered from smallest to largest absolute Shapley value. "original" plots the features in the original order of the data table.

scatter_features

Integer or character vector. Only used for plot_type = "scatter". Specifies which features to include in the scatter plot. Can be a numerical vector indicating feature index, or a character vector, indicating the name(s) of the feature(s) to plot.

scatter_hist Logical. Only used for plot_type = "scatter". Whether to include a scatter_hist indicating the distribution of the data when making the scatter plot. Note

that the bins are scaled so that when all the bins are stacked they fit the span of the y-axis of the plot.

include_group_feature_means
: Logical. Whether to include the average feature value in a group on the y-axis or not. If FALSE (default), then no value is shown for the groups. If TRUE, then shapr includes the mean of the features in each group.

beeswarm_cex
: Numeric. The cex argument of ggbeeswarm::geom_beeswarm(), controlling the spacing in the beeswarm plots.

...
: Other arguments passed to underlying functions, like ggbeeswarm::geom_beeswarm() for plot_type = "beeswarm".

Details

See the examples below, or vignette("general_usage", package = "shapr") for examples of how to use the function.

Value

ggplot object with plots of the Shapley value explanations

Author(s)

Martin Jullum, Vilde Ung, Lars Henry Berge Olsen

Examples

```
if (requireNamespace("party", quietly = TRUE)) {
  data("airquality")
  airquality <- airquality[complete.cases(airquality), ]
  x_var <- c("Solar.R", "Wind", "Temp", "Month")
  y_var <- "Ozone"

  # Split data into test- and training data
  data_train <- head(airquality, -50)
  data_explain <- tail(airquality, 50)

  x_train <- data_train[, x_var]
  x_explain <- data_explain[, x_var]

  # Fit a linear model
  lm_formula <- as.formula(paste0(y_var, " ~ ", paste0(x_var, collapse = " + ")))
  model <- lm(lm_formula, data = data_train)

  # Explain predictions
  p <- mean(data_train[, y_var])

  # Empirical approach
  x <- explain(
    model = model,
    x_explain = x_explain,
    x_train = x_train,
```

```
  approach = "empirical",
  phi0 = p,
  n_MC_samples = 1e2
)

if (requireNamespace(c("ggplot2", "ggbeeswarm"), quietly = TRUE)) {
  # The default plotting option is a bar plot of the Shapley values
  # We draw bar plots for the first 4 observations
  plot(x, index_x_explain = 1:4)

  # We can also make waterfall plots
  plot(x, plot_type = "waterfall", index_x_explain = 1:4)
  # And only showing the two features with the largest contributions
  plot(x, plot_type = "waterfall", index_x_explain = 1:4, top_k_features = 2)

  # Or scatter plots showing the distribution of the Shapley values and feature values
  plot(x, plot_type = "scatter")
  # And only for a specific feature
  plot(x, plot_type = "scatter", scatter_features = "Temp")

 # Or a beeswarm plot summarising the Shapley values and feature values for all features
  plot(x, plot_type = "beeswarm")
  plot(x, plot_type = "beeswarm", col = c("red", "black")) # we can change colors

 # Additional arguments can be passed to ggbeeswarm::geom_beeswarm() using the '...' argument.
  # For instance, sometimes the beeswarm plots overlap too much.
  # This can be fixed with the 'corral="wrap" argument.
  # See ?ggbeeswarm::geom_beeswarm for more information.
  plot(x, plot_type = "beeswarm", corral = "wrap")
}

# Example of scatter and beeswarm plot with factor variables
airquality$Month_factor <- as.factor(month.abb[airquality$Month])
airquality <- airquality[complete.cases(airquality), ]
x_var <- c("Solar.R", "Wind", "Temp", "Month_factor")
y_var <- "Ozone"

# Split data into test- and training data
data_train <- airquality
data_explain <- tail(airquality, 50)

x_train <- data_train[, x_var]
x_explain <- data_explain[, x_var]

# Fit a linear model
lm_formula <- as.formula(paste0(y_var, " ~ ", paste0(x_var, collapse = " + ")))
model <- lm(lm_formula, data = data_train)

# Explain predictions
p <- mean(data_train[, y_var])

# Empirical approach
x <- explain(
```

```
    model = model,
    x_explain = x_explain,
    x_train = x_train,
    approach = "ctree",
    phi0 = p,
    n_MC_samples = 1e2
  )

  if (requireNamespace(c("ggplot2", "ggbeeswarm"), quietly = TRUE)) {
    plot(x, plot_type = "scatter")
    plot(x, plot_type = "beeswarm")
  }

  # Example of further modification of the output from plot.shapr
  plt <- plot(x, index_x_explain = 1:4, print_ggplot = FALSE) # Storing without printing

  # Displays the modified ggplot object
  plt +
    ggplot2::ggtitle("My custom title") +
    ggplot2::ylab("Variable influence") +
    ggplot2::xlab("Variable")
}
```

`plot_MSEv_eval_crit` *Plots of the MSEv Evaluation Criterion*

Description

Make plots to visualize and compare the MSEv evaluation criterion for a list of `explain()` objects applied to the same data and model. The function creates bar plots and line plots with points to illustrate the overall MSEv evaluation criterion, but also for each observation/explicand and coalition by only averaging over the coalitions and observations/explicands, respectively.

Usage

```
plot_MSEv_eval_crit(
  explanation_list,
  index_x_explain = 1:10,
  id_coalition = NULL,
  CI_level = if (length(explanation_list[[1]]$pred_explain) < 20) NULL else 0.95,
  geom_col_width = 0.9,
  plot_type = "overall"
)
```

Arguments

`explanation_list`
: A list of `explain()` objects applied to the same data and model. If the entries in the list are named, then the function use these names. Otherwise, they default to the approach names (with integer suffix for duplicates) for the explanation objects in `explanation_list`.

`index_x_explain`
: Integer vector. Which of the test observations to plot. For example, if you have explained 10 observations using `explain()`, you can generate a plot for the first five observations by setting `index_x_explain = 1:5`. Defaults to the first 10 observations for `plot_type = "bar"` and `"waterfall"`, and to all observations for `plot_type = "scatter"` and `"beeswarm"`.

`id_coalition`
: Integer vector. Which of the coalitions to plot. E.g. if you used `n_coalitions = 16` in `explain()`, you can generate a plot for the first 5 coalitions and the 10th by setting `id_coalition = c(1:5, 10)`.

`CI_level`
: Positive numeric between zero and one. Default is `0.95` if the number of observations to explain is larger than 20, otherwise `CI_level = NULL`, which removes the confidence intervals. The level of the approximate confidence intervals for the overall MSEv and the MSEv_coalition. The confidence intervals are based on that the MSEv scores are means over the observations/explicands, and that means are approximation normal. Since the standard deviations are estimated, we use the quantile t from the T distribution with N_explicands - 1 degrees of freedom corresponding to the provided level. Here, N_explicands is the number of observations/explicands. MSEv +/- t*SD(MSEv)/sqrt(N_explicands). Note that the* `explain()` *function already scales the standard deviation by sqrt(N_explicands), thus, the CI are MSEv* \/- *t*MSEv_sd, where the values MSEv and MSEv_sd are extracted from the MSEv data.tables in the objects in the `explanation_list`.

`geom_col_width`
: Numeric. Bar width. By default, set to 90% of the `ggplot2::resolution()` of the data.

`plot_type`
: Character vector. The possible options are "overall" (default), "coalition", and "explicand". If `plot_type = "overall"`, then the plot (one bar plot) associated with the overall MSEv evaluation criterion for each method is created, i.e., when averaging over both the coalitions and observations/explicands. If `plot_type = "coalition"`, then the plots (one line plot and one bar plot) associated with the MSEv evaluation criterion for each coalition are created, i.e., when we only average over the observations/explicands. If `plot_type = "explicand"`, then the plots (one line plot and one bar plot) associated with the MSEv evaluation criterion for each observations/explicands are created, i.e., when we only average over the coalitions. If `plot_type` is a vector of one or several of "overall", "coalition", and "explicand", then the associated plots are created.

Details

Note that in contrast to `plot.shapr()`, `plot_MSEv_eval_crit()` always just returns the ggplot objects, i.e. no force displaying through `ggplot2::print.ggplot()`.

Value

Either a single ggplot2::ggplot() object of the MSEv criterion when plot_type = "overall", or a list of ggplot2::ggplot() objects based on the plot_type parameter.

Author(s)

Lars Henry Berge Olsen

Examples

```
if (requireNamespace("xgboost", quietly = TRUE) && requireNamespace("ggplot2", quietly = TRUE)) {
  # Get the data
  data("airquality")
  data <- data.table::as.data.table(airquality)
  data <- data[complete.cases(data), ]

  #' Define the features and the response
  x_var <- c("Solar.R", "Wind", "Temp", "Month")
  y_var <- "Ozone"

  # Split data into test and training data set
  ind_x_explain <- 1:25
  x_train <- data[-ind_x_explain, ..x_var]
  y_train <- data[-ind_x_explain, get(y_var)]
  x_explain <- data[ind_x_explain, ..x_var]

  # Fitting a basic xgboost model to the training data
  model <- xgboost::xgboost(
    x = x_train,
    y = y_train,
    nround = 20,
    verbosity = 0
  )

  # Specifying the phi_0, i.e. the expected prediction without any features
  phi0 <- mean(y_train)

  # Independence approach
  explanation_independence <- explain(
    model = model,
    x_explain = x_explain,
    x_train = x_train,
    approach = "independence",
    phi0 = phi0,
    n_MC_samples = 1e2
  )

  # Gaussian 1e1 approach
  explanation_gaussian_1e1 <- explain(
    model = model,
    x_explain = x_explain,
    x_train = x_train,
```

```
  approach = "gaussian",
  phi0 = phi0,
  n_MC_samples = 1e1
)

# Gaussian 1e2 approach
explanation_gaussian_1e2 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "gaussian",
  phi0 = phi0,
  n_MC_samples = 1e2
)

# ctree approach
explanation_ctree <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "ctree",
  phi0 = phi0,
  n_MC_samples = 1e2
)

# Combined approach
explanation_combined <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = c("gaussian", "independence", "ctree"),
  phi0 = phi0,
  n_MC_samples = 1e2
)

# Create a list of explanations with names
explanation_list_named <- list(
  "Ind." = explanation_independence,
  "Gaus. 1e1" = explanation_gaussian_1e1,
  "Gaus. 1e2" = explanation_gaussian_1e2,
  "Ctree" = explanation_ctree,
  "Combined" = explanation_combined
)

# Create the default MSEv plot where we average over both the coalitions and observations
# with approximate 95% confidence intervals
plot_MSEv_eval_crit(explanation_list_named, CI_level = 0.95, plot_type = "overall")

# Can also create plots of the MSEv criterion averaged only over the coalitions or observations.
MSEv_figures <- plot_MSEv_eval_crit(explanation_list_named,
  CI_level = 0.95,
  plot_type = c("overall", "coalition", "explicand")
)
```

```
MSEv_figures$MSEv_bar
MSEv_figures$MSEv_coalition_bar
MSEv_figures$MSEv_explicand_bar

# When there are many coalitions or observations, then it can be easier to look at line plots
MSEv_figures$MSEv_coalition_line_point
MSEv_figures$MSEv_explicand_line_point

# We can specify which observations or coalitions to plot
plot_MSEv_eval_crit(explanation_list_named,
  plot_type = "explicand",
  index_x_explain = c(1, 3:4, 6),
  CI_level = 0.95
)$MSEv_explicand_bar
plot_MSEv_eval_crit(explanation_list_named,
  plot_type = "coalition",
  id_coalition = c(3, 4, 9, 13:15),
  CI_level = 0.95
)$MSEv_coalition_bar

# We can alter the figures if other palette schemes or design is wanted
bar_text_n_decimals <- 1
MSEv_figures$MSEv_bar +
  ggplot2::scale_x_discrete(limits = rev(levels(MSEv_figures$MSEv_bar$data$Method))) +
  ggplot2::coord_flip() +
  ggplot2::scale_fill_discrete() + #' Default ggplot2 palette
  ggplot2::theme_minimal() + #' This must be set before the other theme call
  ggplot2::theme(
    plot.title = ggplot2::element_text(size = 10),
    legend.position = "bottom"
  ) +
  ggplot2::guides(fill = ggplot2::guide_legend(nrow = 1, ncol = 6)) +
  ggplot2::geom_text(
    ggplot2::aes(label = sprintf(
      paste("%.", sprintf("%d", bar_text_n_decimals), "f", sep = ""),
      round(MSEv, bar_text_n_decimals)
    )),
    vjust = -1.1, # This value must be altered based on the plot dimension
    hjust = 1.1, # This value must be altered based on the plot dimension
    color = "black",
    position = ggplot2::position_dodge(0.9),
    size = 5
  )
}
```

---

plot_SV_several_approaches

Shapley Value Bar Plots for Several Explanation Objects

Description

Make plots to visualize and compare the estimated Shapley values for a list of `explain()` objects applied to the same data and model. For group-wise Shapley values, the features values plotted are the mean feature values for all features in each group.

Usage

```
plot_SV_several_approaches(
  explanation_list,
  index_explicands = NULL,
  index_explicands_sort = FALSE,
  only_these_features = NULL,
  plot_phi0 = FALSE,
  digits = 4,
  print_ggplot = TRUE,
  add_zero_line = FALSE,
  axis_labels_n_dodge = NULL,
  axis_labels_rotate_angle = NULL,
  horizontal_bars = TRUE,
  facet_scales = "free",
  facet_ncol = 2,
  geom_col_width = 0.85,
  brewer_palette = NULL,
  include_group_feature_means = FALSE
)
```

Arguments

`explanation_list`
: A list of `explain()` objects applied to the same data and model. If the entries in the list are named, then the function use these names. Otherwise, they default to the approach names (with integer suffix for duplicates) for the explanation objects in `explanation_list`.

`index_explicands`
: Integer vector. Which of the explicands (test observations) to plot. E.g. if you have explained 10 observations using `explain()`, you can generate a plot for the first 5 observations/explicands and the 10th by setting `index_x_explain = c(1:5, 10)`. The argument `index_explicands_sort` must be `FALSE` to plot the explicand in the order specified in `index_x_explain`.

`index_explicands_sort`
: Boolean. If `FALSE` (default), then `shapr` plots the explicands in the order specified in `index_explicands`. If `TRUE`, then `shapr` sort the indices in increasing order based on their id.

`only_these_features`
: String vector. Containing the names of the features which are to be included in the bar plots.

`plot_phi0`
: Boolean. If we are to include the ϕ_0 in the bar plots or not.

digits — Integer. Number of significant digits to use in the feature description. Applicable for plot_type "bar" and "waterfall"

print_ggplot — Logical. Whether to print the created ggplot object once it is returned. The default is TRUE which ensures the plot is always displayed also in loops, functions, when sourcing a script, and when assigning the output to a variable like p <- plot.shapr(...). See ggplot2::print.ggplot() for more details. If you wish to further modify the returned ggplot object outside of plot.shapr, we recommend setting print_ggplot = FALSE to avoid force printing. See the examples for a practical use case.

add_zero_line — Boolean. If we are to add a black line for a feature contribution of 0.

axis_labels_n_dodge — Integer. The number of rows that should be used to render the labels. This is useful for displaying labels that would otherwise overlap.

axis_labels_rotate_angle — Numeric. The angle of the axis label, where 0 means horizontal, 45 means tilted, and 90 means vertical. Compared to setting the angle in ggplot2::theme() / ggplot2::element_text(), this also uses some heuristics to automatically pick the hjust and vjust that you probably want.

horizontal_bars — Boolean. Flip Cartesian coordinates so that horizontal becomes vertical, and vertical, horizontal. This is primarily useful for converting geoms and statistics which display y conditional on x, to x conditional on y. See ggplot2::coord_flip().

facet_scales — Should scales be free ("free", the default), fixed ("fixed"), or free in one dimension ("free_x", "free_y")? The user has to change the latter manually depending on the value of horizontal_bars.

facet_ncol — Integer. The number of columns in the facet grid. Default is facet_ncol = 2.

geom_col_width — Numeric. Bar width. By default, set to 85% of the ggplot2::resolution() of the data.

brewer_palette — String. Name of one of the color palettes from RColorBrewer::RColorBrewer(). If NULL, then the function uses the default ggplot2::ggplot() color scheme. The following palettes are available for use with these scales:

Diverging BrBG, PiYG, PRGn, PuOr, RdBu, RdGy, RdYlBu, RdYlGn, Spectral

Qualitative Accent, Dark2, Paired, Pastel1, Pastel2, Set1, Set2, Set3

Sequential Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu, YlOrBr, YlOrRd

include_group_feature_means — Logical. Whether to include the average feature value in a group on the y-axis or not. If FALSE (default), then no value is shown for the groups. If TRUE, then shapr includes the mean of the features in each group.

Value

A ggplot2::ggplot() object.

Author(s)

Lars Henry Berge Olsen

Examples

```
## Not run:
if (requireNamespace("xgboost", quietly = TRUE) && requireNamespace("ggplot2", quietly = TRUE)) {
  # Get the data
  data("airquality")
  data <- data.table::as.data.table(airquality)
  data <- data[complete.cases(data), ]

  # Define the features and the response
  x_var <- c("Solar.R", "Wind", "Temp", "Month")
  y_var <- "Ozone"

  # Split data into test and training data set
  ind_x_explain <- 1:12
  x_train <- data[-ind_x_explain, ..x_var]
  y_train <- data[-ind_x_explain, get(y_var)]
  x_explain <- data[ind_x_explain, ..x_var]

  # Fitting a basic xgboost model to the training data
  model <- xgboost::xgboost(
    x = x_train,
    y = y_train,
    nround = 20,
    verbosity = 0
  )

  # Specifying the phi_0, i.e. the expected prediction without any features
  phi0 <- mean(y_train)

  # Independence approach
  explanation_independence <- explain(
    model = model,
    x_explain = x_explain,
    x_train = x_train,
    approach = "independence",
    phi0 = phi0,
    n_MC_samples = 1e2
  )

  # Empirical approach
  explanation_empirical <- explain(
    model = model,
    x_explain = x_explain,
    x_train = x_train,
    approach = "empirical",
    phi0 = phi0,
    n_MC_samples = 1e2
  )
```

```
# Gaussian 1e1 approach
explanation_gaussian_1e1 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "gaussian",
  phi0 = phi0,
  n_MC_samples = 1e1
)

# Gaussian 1e2 approach
explanation_gaussian_1e2 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "gaussian",
  phi0 = phi0,
  n_MC_samples = 1e2
)

# Combined approach
explanation_combined <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = c("gaussian", "ctree", "empirical"),
  phi0 = phi0,
  n_MC_samples = 1e2
)

# Create a list of explanations with names
explanation_list <- list(
  "Ind." = explanation_independence,
  "Emp." = explanation_empirical,
  "Gaus. 1e1" = explanation_gaussian_1e1,
  "Gaus. 1e2" = explanation_gaussian_1e2,
  "Combined" = explanation_combined
)

# The function uses the provided names.
plot_SV_several_approaches(explanation_list)

# We can change the number of columns in the grid of plots and add other visual alterations
# Set `print_ggplot = FALSE` to avoid force displaying the ggplot object before the modifications
 # outside plot_SV_several_approaches()

plot_SV_several_approaches(explanation_list,
  facet_ncol = 3,
  facet_scales = "free_y",
  add_zero_line = TRUE,
  digits = 2,
  brewer_palette = "Paired",
```

```
  geom_col_width = 0.6,
  print_ggplot = FALSE
) +
  ggplot2::theme_minimal() +
 ggplot2::theme(legend.position = "bottom", plot.title = ggplot2::element_text(size = 10))


# We can specify which explicands to plot to get less chaotic plots and make the bars vertical
plot_SV_several_approaches(explanation_list,
  index_explicands = c(1:2, 5, 10),
  horizontal_bars = FALSE,
  axis_labels_rotate_angle = 45
)

# We can change the order of the features by specifying the
# order using the `only_these_features` parameter.
plot_SV_several_approaches(explanation_list,
  index_explicands = c(1:2, 5, 10),
  only_these_features = c("Temp", "Solar.R", "Month", "Wind")
)

# We can also remove certain features if we are not interested in them
# or want to focus on, e.g., two features. The function will give a
# message to if the user specifies non-valid feature names.
plot_SV_several_approaches(explanation_list,
  index_explicands = c(1:2, 5, 10),
  only_these_features = c("Temp", "Solar.R"),
  plot_phi0 = TRUE
)
}

## End(Not run)
```

`plot_vaeac_eval_crit` *Plot the training VLB and validation IWAE for* `vaeac` *models*

Description

This function makes (`ggplot2::ggplot()`) figures of the training VLB and the validation IWAE for a list of `explain()` objects with `approach = "vaeac"`. See `setup_approach()` for more information about the `vaeac` approach. Two figures are returned by the function. In the figure, each object in `explanation_list` gets its own facet, while in the second figure, we plot the criteria in each facet for all objects.

Usage

```
plot_vaeac_eval_crit(
  explanation_list,
  plot_from_nth_epoch = 1,
```

```
  plot_every_nth_epoch = 1,
  criteria = c("VLB", "IWAE"),
  plot_type = c("method", "criterion"),
  facet_wrap_scales = "fixed",
  facet_wrap_ncol = NULL
)
```

Arguments

`explanation_list`
: A list of `explain()` objects applied to the same data, model, and `vaeac` must be the used approach. If the entries in the list is named, then the function use these names. Otherwise, it defaults to the approach names (with integer suffix for duplicates) for the explanation objects in `explanation_list`.

`plot_from_nth_epoch`
: Integer. If we are only plot the results form the nth epoch and so forth. The first epochs can be large in absolute value and make the rest of the plot difficult to interpret.

`plot_every_nth_epoch`
: Integer. If we are only to plot every nth epoch. Usefully to illustrate the overall trend, as there can be a lot of fluctuation and oscillation in the values between each epoch.

`criteria`
: Character vector. The possible options are "VLB", "IWAE", "IWAE_running". Default is the first two.

`plot_type`
: Character vector. The possible options are "method" and "criterion". Default is to plot both.

`facet_wrap_scales`
: String. Should the scales be fixed (`"fixed"`, the default), free (`"free"`), or free in one dimension (`"free_x"`, `"free_y"`).

`facet_wrap_ncol`
: Integer. Number of columns in the facet wrap.

Details

See Olsen et al. (2022) or the blog post for a summary of the VLB and IWAE.

Value

Either a single `ggplot2::ggplot()` object or a list of `ggplot2::ggplot()` objects based on the `plot_type` parameter.

Author(s)

Lars Henry Berge Olsen

References

- Olsen, L. H., Glad, I. K., Jullum, M., & Aas, K. (2022). Using Shapley values and variational autoencoders to explain predictive models with dependent mixed features. Journal of machine learning research, 23(213), 1-51

Examples

```
if (requireNamespace("xgboost", quietly = TRUE) &&
  requireNamespace("torch", quietly = TRUE) &&
  torch::torch_is_installed()) {
  data("airquality")
  data <- data.table::as.data.table(airquality)
  data <- data[complete.cases(data), ]

  x_var <- c("Solar.R", "Wind", "Temp", "Month")
  y_var <- "Ozone"

  ind_x_explain <- 1:6
  x_train <- data[-ind_x_explain, ..x_var]
  y_train <- data[-ind_x_explain, get(y_var)]
  x_explain <- data[ind_x_explain, ..x_var]

  # Fitting a basic xgboost model to the training data
  model <- xgboost::xgboost(
    x = x_train,
    y = y_train,
    nround = 100,
    verbosity = 0
  )

  # Specifying the phi_0, i.e. the expected prediction without any features
  p0 <- mean(y_train)

  # Train vaeac with and without paired sampling
  explanation_paired <- explain(
    model = model,
    x_explain = x_explain,
    x_train = x_train,
    approach = "vaeac",
    phi0 = p0,
    n_MC_samples = 1, # As we are only interested in the training of the vaeac
    vaeac.epochs = 10, # Should be higher in applications.
    vaeac.n_vaeacs_initialize = 1,
    vaeac.width = 16,
    vaeac.depth = 2,
    vaeac.extra_parameters = list(vaeac.paired_sampling = TRUE)
  )

  explanation_regular <- explain(
    model = model,
    x_explain = x_explain,
```

```
  x_train = x_train,
  approach = "vaeac",
  phi0 = p0,
  n_MC_samples = 1, # As we are only interested in the training of the vaeac
  vaeac.epochs = 10, # Should be higher in applications.
  vaeac.width = 16,
  vaeac.depth = 2,
  vaeac.n_vaeacs_initialize = 1,
  vaeac.extra_parameters = list(vaeac.paired_sampling = FALSE)
)

# Collect the explanation objects in an named list
explanation_list <- list(
  "Regular sampling" = explanation_regular,
  "Paired sampling" = explanation_paired
)

# Call the function with the named list, will use the provided names
plot_vaeac_eval_crit(explanation_list = explanation_list)

# The function also works if we have only one method,
# but then one should only look at the method plot.
plot_vaeac_eval_crit(
  explanation_list = explanation_list[2],
  plot_type = "method"
)

# Can alter the plot
plot_vaeac_eval_crit(
  explanation_list = explanation_list,
  plot_from_nth_epoch = 2,
  plot_every_nth_epoch = 2,
  facet_wrap_scales = "free"
)

# If we only want the VLB
plot_vaeac_eval_crit(
  explanation_list = explanation_list,
  criteria = "VLB",
  plot_type = "criterion"
)

# If we want only want the criterion version
tmp_fig_criterion <-
  plot_vaeac_eval_crit(explanation_list = explanation_list, plot_type = "criterion")

# Since tmp_fig_criterion is a ggplot2 object, we can alter it
# by, e.g,. adding points or smooths with se bands
tmp_fig_criterion + ggplot2::geom_point(shape = "circle", size = 1, ggplot2::aes(col = Method))
tmp_fig_criterion$layers[[1]] <- NULL
tmp_fig_criterion + ggplot2::geom_smooth(method = "loess", formula = y ~ x, se = TRUE) +
  ggplot2::scale_color_brewer(palette = "Set1") +
  ggplot2::theme_minimal()
```

```
}
```

`plot_vaeac_imputed_ggpairs` *Plot Pairwise Plots for Imputed and True Data*

Description

A function that creates a matrix of plots (`GGally::ggpairs()`) from generated imputations from the unconditioned distribution $p(\boldsymbol{x})$ estimated by a `vaeac` model, and then compares the imputed values with data from the true distribution (if provided). See ggpairs for an introduction to `GGally::ggpairs()`, and the corresponding vignette.

Usage

```
plot_vaeac_imputed_ggpairs(
  explanation,
  which_vaeac_model = "best",
  x_true = NULL,
  add_title = TRUE,
  alpha = 0.5,
  upper_cont = c("cor", "points", "smooth", "smooth_loess", "density", "blank"),
  upper_cat = c("count", "cross", "ratio", "facetbar", "blank"),
  upper_mix = c("box", "box_no_facet", "dot", "dot_no_facet", "facethist",
    "facetdensity", "denstrip", "blank"),
  lower_cont = c("points", "smooth", "smooth_loess", "density", "cor", "blank"),
  lower_cat = c("facetbar", "ratio", "count", "cross", "blank"),
  lower_mix = c("facetdensity", "box", "box_no_facet", "dot", "dot_no_facet",
    "facethist", "denstrip", "blank"),
  diag_cont = c("densityDiag", "barDiag", "blankDiag"),
  diag_cat = c("barDiag", "blankDiag"),
  cor_method = c("pearson", "kendall", "spearman")
)
```

Arguments

`explanation`	Shapr list. The output list from the `explain()` function.
`which_vaeac_model`	String. Indicating which `vaeac` model to use when generating the samples. Possible options are always `'best'`, `'best_running'`, and `'last'`. All possible options can be obtained by calling `names(explanation$internal$parameters$vaeac$models)`.
`x_true`	Data.table containing the data from the distribution that the `vaeac` model is fitted to.
`add_title`	Logical. If `TRUE`, then a title is added to the plot based on the internal description of the `vaeac` model specified in `which_vaeac_model`.

alpha	Numeric between 0 and 1 (default is 0.5). The degree of color transparency.
upper_cont	String. Type of plot to use in upper triangle for continuous features, see GGally::ggpairs(). Possible options are: 'cor' (default), 'points', 'smooth', 'smooth_loess', 'density', and 'blank'.
upper_cat	String. Type of plot to use in upper triangle for categorical features, see GGally::ggpairs(). Possible options are: 'count' (default), 'cross', 'ratio', 'facetbar', and 'blank'.
upper_mix	String. Type of plot to use in upper triangle for mixed features, see GGally::ggpairs(). Possible options are: 'box' (default), 'box_no_facet', 'dot', 'dot_no_facet', 'facethist', 'facetdensity', 'denstrip', and 'blank'
lower_cont	String. Type of plot to use in lower triangle for continuous features, see GGally::ggpairs(). Possible options are: 'points' (default), 'smooth', 'smooth_loess', 'density', 'cor', and 'blank'.
lower_cat	String. Type of plot to use in lower triangle for categorical features, see GGally::ggpairs(). Possible options are: 'facetbar' (default), 'ratio', 'count', 'cross', and 'blank'.
lower_mix	String. Type of plot to use in lower triangle for mixed features, see GGally::ggpairs(). Possible options are: 'facetdensity' (default), 'box', 'box_no_facet', 'dot', 'dot_no_facet', 'facethist', 'denstrip', and 'blank'.
diag_cont	String. Type of plot to use on the diagonal for continuous features, see GGally::ggpairs(). Possible options are: 'densityDiag' (default), 'barDiag', and 'blankDiag'.
diag_cat	String. Type of plot to use on the diagonal for categorical features, see GGally::ggpairs(). Possible options are: 'barDiag' (default) and 'blankDiag'.
cor_method	String. Type of correlation measure, see GGally::ggpairs(). Possible options are: 'pearson' (default), 'kendall', and 'spearman'.

Value

A GGally::ggpairs() figure.

Author(s)

Lars Henry Berge Olsen

References

- Olsen, L. H., Glad, I. K., Jullum, M., & Aas, K. (2022). Using Shapley values and variational autoencoders to explain predictive models with dependent mixed features. Journal of machine learning research, 23(213), 1-51

Examples

```
if (requireNamespace("xgboost", quietly = TRUE) &&
  requireNamespace("ggplot2", quietly = TRUE) &&
  requireNamespace("torch", quietly = TRUE) &&
  torch::torch_is_installed()) {
```

```
data("airquality")
data <- data.table::as.data.table(airquality)
data <- data[complete.cases(data), ]

x_var <- c("Solar.R", "Wind", "Temp", "Month")
y_var <- "Ozone"

ind_x_explain <- 1:6
x_train <- data[-ind_x_explain, ..x_var]
y_train <- data[-ind_x_explain, get(y_var)]
x_explain <- data[ind_x_explain, ..x_var]

# Fitting a basic xgboost model to the training data
model <- xgboost::xgboost(
  x = x_train,
  y = y_train,
  nround = 100,
  verbosity = 0
)

explanation <- shapr::explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "vaeac",
  phi0 = mean(y_train),
  n_MC_samples = 1,
  vaeac.epochs = 10,
  vaeac.n_vaeacs_initialize = 1
)

# Plot the results
figure <- shapr::plot_vaeac_imputed_ggpairs(
  explanation = explanation,
  which_vaeac_model = "best",
  x_true = x_train,
  add_title = TRUE
)
figure

# Note that this is an ggplot2 object which we can alter, e.g., we can change the colors.
figure +
  ggplot2::scale_color_manual(values = c("#E69F00", "#999999")) +
  ggplot2::scale_fill_manual(values = c("#E69F00", "#999999"))
}
```

`print.shapr` *Print Method for Shapr Objects*

Description

Print Method for Shapr Objects

Usage

```
## S3 method for class 'shapr'
print(
  x,
 what = c("shapley_est", "shapley_sd", "MSEv", "MSEv_explicand", "MSEv_coalition",
    "timing_summary"),
  digits = 3L,
  ...
)
```

Arguments

x	A shapr object
what	Character. Which component to print. Options are "shapley_est", "shapley_sd", "MSEv", "MSEv_explicand", "MSEv_coalition", and "timing_summary". Defaults to "shapley_est". Only one component can be printed at a time. See the details section of `get_results()` for details about each component.
digits	Integer. Number of significant digits to display. Defaults to 3.
...	Further arguments passed to `data.table::print.data.table()`.

Value

The object is returned invisibly after printing selected output.

---

print.summary.shapr *Print Method for summary.shapr Objects*

Description

Print Method for summary.shapr Objects

Usage

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

Arguments

x	A summary.shapr object.
...	Currently unused.

Value

Invisibly returns the summary object.

summary.shapr *Summary Method for Shapr Objects*

Description

Provides a formatted summary of a shapr object and returns an object of class summary.shapr containing the same information as returned by get_results().

Usage

```
## S3 method for class 'shapr'
summary(object, digits = 2L, ...)
```

Arguments

object	A shapr object.
digits	Integer. (Maximum) number of digits to be displayed after the decimal point. Defaults to 2.
...	Currently unused.

Value

An object of class summary.shapr, which is a named list with the same accessible components as returned by get_results(). See get_results() for details about each component.

Examples

```
# Load example data
data("airquality")
airquality <- airquality[complete.cases(airquality), ]
x_var <- c("Solar.R", "Wind", "Temp", "Month")
y_var <- "Ozone"

# Split data into test and training data
data_train <- head(airquality, -3)
data_explain <- tail(airquality, 3)

x_train <- data_train[, x_var]
x_explain <- data_explain[, x_var]

# Fit a linear model
lm_formula <- as.formula(paste0(y_var, " ~ ", paste0(x_var, collapse = " + ")))
model <- lm(lm_formula, data = data_train)

# Explain predictions
p <- mean(data_train[, y_var])

explanation <- explain(
```

```
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "gaussian",
  phi0 = p,
  n_MC_samples = 1e2
)

# Call summary without assignment - prints formatted output to console
summary(explanation)

# Assign to variable - returns shapr.summary with summary information for later use
expl_summary <- summary(explanation) # print(expl_summary) provides the formatted output

# Access components from the summary object
expl_summary$shapley_est # Estimated Shapley values
expl_summary$timing_summary$total_time_secs # Total computation time
expl_summary$approach # Approach used
```

`vaeac_get_extra_para_default` *Specify the Extra Parameters in the* `vaeac` *Model*

Description

In this function, we specify the default values for the extra parameters used in `explain()` for `approach = "vaeac"`.

Usage

```
vaeac_get_extra_para_default(
  vaeac.model_description = make.names(Sys.time()),
  vaeac.folder_to_save_model = tempdir(),
  vaeac.pretrained_vaeac_model = NULL,
  vaeac.cuda = FALSE,
  vaeac.epochs_initiation_phase = 2,
  vaeac.epochs_early_stopping = NULL,
  vaeac.save_every_nth_epoch = NULL,
  vaeac.val_ratio = 0.25,
  vaeac.val_iwae_n_samples = 25,
  vaeac.batch_size = 64,
  vaeac.batch_size_sampling = NULL,
  vaeac.running_avg_n_values = 5,
  vaeac.skip_conn_layer = TRUE,
  vaeac.skip_conn_masked_enc_dec = TRUE,
  vaeac.batch_normalization = FALSE,
```

```
  vaeac.paired_sampling = TRUE,
  vaeac.masking_ratio = 0.5,
  vaeac.mask_gen_coalitions = NULL,
  vaeac.mask_gen_coalitions_prob = NULL,
  vaeac.sigma_mu = 10000,
  vaeac.sigma_sigma = 1e-04,
  vaeac.sample_random = TRUE,
  vaeac.save_data = FALSE,
  vaeac.log_exp_cont_feat = FALSE,
  vaeac.which_vaeac_model = "best",
  vaeac.save_model = TRUE
)
```

Arguments

`vaeac.model_description`
: String (default is `make.names(Sys.time())`). String containing, e.g., the name of the data distribution or additional parameter information. Used in the save name of the fitted model. If not provided, then a name will be generated based on `base::Sys.time()` to ensure a unique name. We use `base::make.names()` to ensure a valid file name for all operating systems.

`vaeac.folder_to_save_model`
: String (default is `base::tempdir()`). String specifying a path to a folder where the function is to save the fitted vaeac model. Note that the path will be removed from the returned `explain()` object if `vaeac.save_model = FALSE`. Furthermore, the model cannot be moved from its original folder if we are to use the `vaeac_train_model_continue()` function to continue training the model.

`vaeac.pretrained_vaeac_model`
: List or String (default is `NULL`). 1) Either a list of class `vaeac`, i.e., the list stored in `explanation$internal$parameters$vaeac` where `explanation` is the returned list from an earlier call to the `explain()` function. 2) A string containing the path to where the `vaeac` model is stored on disk, for example, `explanation$internal$parameters$vaeac$models$best`.

`vaeac.cuda`
: Logical (default is `FALSE`). If `TRUE`, then the `vaeac` model will be trained using cuda/GPU. If `torch::cuda_is_available()` is `FALSE`, we fall back to using the CPU. Using a GPU for smaller tabular dataset often do not improve the efficiency. See `vignette("installation", package = "torch")` fo help to enable running on the GPU (only Linux and Windows).

`vaeac.epochs_initiation_phase`
: Positive integer (default is 2). The number of epochs to run each of the `vaeac.n_vaeacs_initialize` `vaeac` models before continuing to train only the best performing model.

`vaeac.epochs_early_stopping`
: Positive integer (default is `NULL`). The training stops if there has been no improvement in the validation IWAE for `vaeac.epochs_early_stopping` epochs. If the user wants the training process to be solely based on this training criterion, then `vaeac.epochs` in `explain()` should be set to a large number. If `NULL`, then `shapr` will internally set `vaeac.epochs_early_stopping = vaeac.epochs` such that early stopping does not occur.

vaeac.save_every_nth_epoch
: Positive integer (default is NULL). If provided, then the vaeac model after every vaeac.save_every_nth_epochth epoch will be saved.

vaeac.val_ratio
: Numeric (default is 0.25). Scalar between 0 and 1 indicating the ratio of instances from the input data which will be used as validation data. That is, vaeac.val_ratio = 0.25 means that 75% of the provided data is used as training data, while the remaining 25% is used as validation data.

vaeac.val_iwae_n_samples
: Positive integer (default is 25). The number of generated samples used to compute the IWAE criterion when validating the vaeac model on the validation data.

vaeac.batch_size
: Positive integer (default is 64). The number of samples to include in each batch during the training of the vaeac model. Used in torch::dataloader().

vaeac.batch_size_sampling
: Positive integer (default is NULL) The number of samples to include in each batch when generating the Monte Carlo samples. If NULL, then the function generates the Monte Carlo samples for the provided coalitions and all explicands sent to explain() at the time. The number of coalitions are determined by the n_batches used by explain(). We recommend to tweak extra_computation_args$max_batch_size and extra_computation_args$min_n_batches rather than vaeac.batch_size_sampling. Larger batch sizes are often much faster provided sufficient memory.

vaeac.running_avg_n_values
: Positive integer (default is 5). The number of previous IWAE values to include when we compute the running means of the IWAE criterion.

vaeac.skip_conn_layer
: Logical (default is TRUE). If TRUE, we apply identity skip connections in each layer, see skip_connection(). That is, we add the input X to the outcome of each hidden layer, so the output becomes $X + activation(WX + b)$.

vaeac.skip_conn_masked_enc_dec
: Logical (default is TRUE). If TRUE, we apply concatenate skip connections between the layers in the masked encoder and decoder. The first layer of the masked encoder will be linked to the last layer of the decoder. The second layer of the masked encoder will be linked to the second to last layer of the decoder, and so on.

vaeac.batch_normalization
: Logical (default is FALSE). If TRUE, we apply batch normalization after the activation function. Note that if vaeac.skip_conn_layer = TRUE, then the normalization is applied after the inclusion of the skip connection. That is, we batch normalize the whole quantity $X + activation(WX + b)$.

vaeac.paired_sampling
: Logical (default is TRUE). If TRUE, we apply paired sampling to the training batches. That is, the training observations in each batch will be duplicated, where the first instance will be masked by S while the second instance will be masked by $\bar{S}$. This ensures that the training of the vaeac model becomes more stable as the model has access to the full version of each training observation.

However, this will increase the training time due to more complex implementation and doubling the size of each batch. See `paired_sampler()` for more information.

`vaeac.masking_ratio`
: Numeric (default is `0.5`). Probability of masking a feature in the `mcar_mask_generator()` (MCAR = Missing Completely At Random). The MCAR masking scheme ensures that `vaeac` model can do arbitrary conditioning as all coalitions will be trained. `vaeac.masking_ratio` will be overruled if `vaeac.mask_gen_coalitions` is specified.

`vaeac.mask_gen_coalitions`
: Matrix (default is `NULL`). Matrix containing the coalitions that the `vaeac` model will be trained on, see `specified_masks_mask_generator()`. This parameter is used internally in `shapr` when we only consider a subset of coalitions, i.e., when `n_coalitions` $< 2^{n_{\text{features}}}$, and for group Shapley, i.e., when `group` is specified in `explain()`.

`vaeac.mask_gen_coalitions_prob`
: Numeric array (default is `NULL`). Array of length equal to the height of `vaeac.mask_gen_coalitions` containing the probabilities of sampling the corresponding coalitions in `vaeac.mask_gen_coalitions`.

`vaeac.sigma_mu`
: Numeric (default is `1e4`). One of two hyperparameter values in the normal-gamma prior used in the masked encoder, see Section 3.3.1 in Olsen et al. (2022).

`vaeac.sigma_sigma`
: Numeric (default is `1e-4`). One of two hyperparameter values in the normal-gamma prior used in the masked encoder, see Section 3.3.1 in Olsen et al. (2022).

`vaeac.sample_random`
: Logical (default is `TRUE`). If `TRUE`, the function generates random Monte Carlo samples from the inferred generative distributions. If `FALSE`, the function use the most likely values, i.e., the mean and class with highest probability for continuous and categorical, respectively.

`vaeac.save_data`
: Logical (default is `FALSE`). If `TRUE`, then the data is stored together with the model. Useful if one are to continue to train the model later using `vaeac_train_model_continue()`.

`vaeac.log_exp_cont_feat`
: Logical (default is `FALSE`). If we are to $\log$ transform all continuous features before sending the data to `vaeac()`. The `vaeac` model creates unbounded Monte Carlo sample values. Thus, if the continuous features are strictly positive (as for, e.g., the Burr distribution and Abalone data set), it can be advantageous to $\log$ transform the data to unbounded form before using `vaeac`. If `TRUE`, then `vaeac_postprocess_data()` will take the $\exp$ of the results to get back to strictly positive values when using the `vaeac` model to impute missing values/generate the Monte Carlo samples.

`vaeac.which_vaeac_model`
: String (default is `best`). The name of the `vaeac` model (snapshots from different epochs) to use when generating the Monte Carlo samples. The standard choices are: `"best"` (epoch with lowest IWAE), `"best_running"` (epoch with lowest running IWAE, see `vaeac.running_avg_n_values`), and `last` (the last epoch).

Note that additional choices are available if vaeac.save_every_nth_epoch is provided. For example, if vaeac.save_every_nth_epoch = 5, then vaeac.which_vaeac_model can also take the values "epoch_5", "epoch_10", "epoch_15", and so on.

vaeac.save_model
: Boolean. If TRUE (default), the vaeac model will be saved either in a base::tempdir() folder or in a user specified location in vaeac.folder_to_save_model. If FALSE, then the paths to model and the model will will be deleted from the returned object from explain().

Details

The vaeac model consists of three neural network (a full encoder, a masked encoder, and a decoder) based on the provided vaeac.depth and vaeac.width. The encoders map the full and masked input representations to latent representations, respectively, where the dimension is given by vaeac.latent_dim. The latent representations are sent to the decoder to go back to the real feature space and provide a samplable probabilistic representation, from which the Monte Carlo samples are generated. We use the vaeac method at the epoch with the lowest validation error (IWAE) by default, but other possibilities are available by setting the vaeac.which_vaeac_model parameter. See Olsen et al. (2022) for more details.

Value

Named list of the default values vaeac extra parameter arguments specified in this function call. Note that both vaeac.model_description and vaeac.folder_to_save_model will change with time and R session.

Author(s)

Lars Henry Berge Olsen

References

- Olsen, L. H., Glad, I. K., Jullum, M., & Aas, K. (2022). Using Shapley values and variational autoencoders to explain predictive models with dependent mixed features. Journal of machine learning research, 23(213), 1-51

---

vaeac_train_model_continue

Continue to Train the vaeac *Model*

Description

Function that loads a previously trained vaeac model and continue the training, either on new data or on the same dataset as it was trained on before. If we are given a new dataset, then we assume that new dataset has the same distribution and one_hot_max_sizes as the original dataset.

Usage

```
vaeac_train_model_continue(
  explanation,
  epochs_new,
  lr_new = NULL,
  x_train = NULL,
  save_data = FALSE,
  verbose = NULL,
  seed = 1
)
```

Arguments

explanation	A explain() object and vaeac must be the used approach.
epochs_new	Positive integer. The number of extra epochs to conduct.
lr_new	Positive numeric. If we are to overwrite the old learning rate in the adam optimizer.
x_train	A data.table containing the training data. Categorical data must have class names $1, 2, \ldots, K$.
save_data	Logical (default is FALSE). If TRUE, then the data is stored together with the model. Useful if one are to continue to train the model later using vaeac_train_model_continue().
verbose	String vector or NULL. Controls verbosity (printout detail level) via one or more of "basic", "progress", "convergence", "shapley" and "vS_details". "basic" (default) displays basic information about the computation and messages about parameters/checks. "progress" displays where in the calculation process the function currently is. "convergence" displays how close the Shapley value estimates are to convergence (only when iterative = TRUE). "shapley" displays intermediate Shapley value estimates and standard deviations (only when iterative = TRUE), and the final estimates. "vS_details" displays information about the v(S) estimates, most relevant for approach %in% c("regression_separate", "regression_surrogate", "vaeac"). NULL means no printout. Any combination can be used, e.g., verbose = c("basic", "vS_details").
seed	Positive integer (default is 1). Seed for reproducibility. Specifies the seed before any randomness based code is being run.

Value

A list containing the training/validation errors and paths to where the vaeac models are saved on the disk.

Author(s)

Lars Henry Berge Olsen

References

- Olsen, L. H., Glad, I. K., Jullum, M., & Aas, K. (2022). Using Shapley values and variational autoencoders to explain predictive models with dependent mixed features. Journal of machine learning research, 23(213), 1-51

Index

arf::adversarial_rf(), *7*, *19*
arf::forde(), *7*, *19*
arf::forge(), *7*, *19*

base::make.names(), *54*
base::Sys.time(), *54*
base::tempdir(), *54*, *57*

data.table::print.data.table(), *51*

explain, 3
explain(), *9*, *24*, *26*, *29–32*, *35, 36*, *40*, *44, 45*, *48*, *53–58*
explain_forecast, 15

future.apply::future_apply, *24*
future::future, *11*, *24*

get_extra_comp_args_default, 23
get_extra_comp_args_default(), *6*, *18*
get_iterative_args_default, 26
get_iterative_args_default(), *6*, *18*
get_output_args_default, 27
get_output_args_default(), *6*, *19*
get_results, 28
get_results(), *51, 52*
get_supported_approaches, 30
get_supported_models, 30
get_supported_models(), *3*, *5*, *16*, *18*
GGally::ggpairs(), *48, 49*
ggbeeswarm::geom_beeswarm(), *33*
ggplot2::coord_flip(), *41*
ggplot2::element_text(), *41*
ggplot2::ggplot(), *32*, *37*, *41*, *44, 45*
ggplot2::print.ggplot(), *32*, *36*, *41*
ggplot2::resolution(), *36*, *41*
ggplot2::theme(), *41*

mcar_mask_generator(), *56*

paired_sampler(), *56*
parsnip::linear_reg(), *8*
plot.shapr, 31
plot.shapr(), *36*
plot_MSEv_eval_crit, 35
plot_MSEv_eval_crit(), *36*
plot_SV_several_approaches, 39
plot_vaeac_eval_crit, 44
plot_vaeac_imputed_ggpairs, 48
print.shapr, 50
print.shapr(), *29*
print.summary.shapr, 51
progressr::progressr, *11*, *24*

RColorBrewer::RColorBrewer(), *41*
recipes::recipe(), *9*
rsample::vfold_cv(), *9*

setup_approach, *6*, *19*
setup_approach(), *44*
setup_approach.ctree(), *4*, *18*
setup_approach.empirical(), *4*, *18*
skip_connection(), *55*
specified_masks_mask_generator(), *56*
summary.shapr, 52
summary.shapr(), *29*

torch::cuda_is_available(), *54*
torch::dataloader(), *55*
torch::nn_leaky_relu(), *9*, *21*
torch::nn_module(), *9*, *21*
torch::nn_relu(), *9*, *21*
torch::nn_selu(), *9*, *21*
torch::nn_sigmoid(), *9*, *21*
torch::optim_adam(), *9*, *21*

vaeac(), *56*
vaeac_get_extra_para_default, 53
vaeac_get_extra_para_default(), *10*, *21*
vaeac_postprocess_data(), *56*
vaeac_train_model_continue, 57
vaeac_train_model_continue(), *54*, *56*, *58*