Package: shapr (via r-universe)

September 13, 2024

Version 0.2.3.9200

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 the method described in Aas, Jullum and Løland (2019) <arXiv:1903.10464>, which accounts for any feature dependence, and thereby produces more accurate estimates of the true Shapley values. An accompanying Python wrapper (shaprpy) is available on GitHub.

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

LazyData true

ByteCompile true

Language en-US

RoxygenNote 7.3.1

Depends R (>= 3.5.0)

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

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

LinkingTo RcppArmadillo, Rcpp

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

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

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

RemoteRef HEAD

RemoteSha ddd32c7c92ba9f37c8505720129d7e10979ccc4c

Contents

`compute_vS` *Computes `v(S)` for all features subsets `S`.*

Description

Computes `v(S)` for all features subsets `S`.

Usage

```
compute_vS(internal, model, predict_model, method = "future")
```

Arguments

`internal`	List. Holds all parameters, data, functions and computed objects used within `explain()` The list contains one or more of the elements `parameters`, `data`, `objects`, `output`.
`model`	Objects. The model object that ought to be explained. See the documentation of `explain()` for details.
`predict_model`	Function. The prediction function used when `model` is not natively supported. See the documentation of `explain()` for details.
`method`	Character Indicates whether the lappy method (default) or loop method should be used.

`explain` *Explain the output of machine learning models with more accurately estimated Shapley values*

Description

Computes dependence-aware Shapley values for observations in `x_explain` from the specified `model` by using the method specified in `approach` to estimate the conditional expectation.

Usage

```
explain(
  model,
  x_explain,
  x_train,
  approach,
  prediction_zero,
  n_combinations = NULL,
  group = NULL,
  n_samples = 1000,
  n_batches = NULL,
  seed = 1,
  keep_samp_for_vS = FALSE,
  predict_model = NULL,
  get_model_specs = NULL,
  MSEv_uniform_comb_weights = TRUE,
  timing = TRUE,
  verbose = 0,
  ...
)
```

Arguments

`model`	The model whose predictions we 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`	A matrix or data.frame/data.table. Contains the the features, whose predictions ought to be explained.
`x_train`	Matrix or data.frame/data.table. Contains the data used to estimate the (conditional) distributions for the features 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 `"gaussian"`, `"copula"`, `"empirical"`, `"ctree"`, `"vaeac"`, `"categorical"`, `"timeseries"`, `"independence"`, `"regression_separate"`, or `"regression_surrogate"`. The two regression approaches can not be combined with any other approach. See details for more information.
`prediction_zero`	Numeric. The prediction value for unseen data, i.e. an estimate of the expected prediction without conditioning on any features. Typically we set this value equal to the mean of the response variable in our training data, but other choices such as the mean of the predictions in the training data are also reasonable.
`n_combinations`	Integer. If `group = NULL`, `n_combinations` represents the number of unique feature combinations to sample. If `group != NULL`, `n_combinations` represents the number of unique group combinations to sample. If `n_combinations = NULL`, the exact method is used and all combinations are considered. The maximum number of combinations equals `2^m`, where `m` is the number of 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. The list element contains character vectors with the features included in each of the different groups.
`n_samples`	Positive integer. Indicating the maximum number of samples to use in the Monte Carlo integration for every conditional expectation. See also details.
`n_batches`	Positive integer (or NULL). Specifies how many batches the total number of feature combinations should be split into when calculating the contribution function for each test observation. The default value is NULL which uses a reasonable trade-off between RAM allocation and computation speed, which depends on

approach and n_combinations. For models with many features, increasing the number of batches reduces the RAM allocation significantly. This typically comes with a small increase in computation time.

seed
: Positive integer. Specifies the seed before any randomness based code is being run. If NULL the seed will be inherited from the calling environment.

keep_samp_for_vS
: Logical. Indicates whether the samples used in the Monte Carlo estimation of v_S should be returned (in internal$output)

predict_model
: Function. The prediction function used 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, respectively, the model and a data.frame/data.table to compute predictions for. 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 argument and provides a list with 3 elements:

labels Character vector with the names of each feature.

classes Character vector with the classes of each features.

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 the checking is disabled for unsupported model classes. Can also be used to override the default function for natively supported model classes.

MSEv_uniform_comb_weights
: Logical. If TRUE (default), then the function weights the combinations uniformly when computing the MSEv criterion. If FALSE, then the function use the Shapley kernel weights to weight the combinations when computing the MSEv criterion. Note that the Shapley kernel weights are replaced by the sampling frequency when not all combinations are considered.

timing
: Logical. Whether the timing of the different parts of the explain() should saved in the model object.

verbose
: An integer specifying the level of verbosity. If 0, shapr will stay silent. If 1, it will print information about performance. If 2, some additional information will be printed out. Use 0 (default) for no verbosity, 1 for low verbose, and 2 for high verbose. TODO: Make this clearer when we end up fixing this and if they should force a progressr bar.

...
: Arguments passed on to setup_approach.empirical, setup_approach.independence, setup_approach.gaussian, setup_approach.copula, setup_approach.ctree, setup_approach.vaeac, setup_approach.categorical, setup_approach.regression_separate, setup_approach.regression_surrogate, setup_approach.timeseries

empirical.type Character. (default = "fixed_sigma") Should be equal to either "independence","fixed_sigma", "AICc_each_k" "AICc_full". TODO: Describe better what the methods do here.

`empirical.eta` Numeric. (default = 0.95) Needs to be `0 < eta <= 1`. Represents the minimum proportion of the total empirical weight that data samples should use. If e.g. `eta = .8` we will choose the `K` samples with the largest weight so that the sum of the weights accounts for 80\ `eta` is the η parameter in equation (15) of Aas et al (2021).

`empirical.fixed_sigma` Positive numeric scalar. (default = 0.1) Represents the kernel bandwidth in the distance computation used when conditioning on all different combinations. Only used when `empirical.type = "fixed_sigma"`

`empirical.n_samples_aicc` Positive integer. (default = 1000) Number of samples to consider in AICc optimization. Only used for `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

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

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

`empirical.cov_mat` Numeric matrix. (Optional, default = NULL) Containing the covariance matrix of the data generating distribution used to define the Mahalanobis distance. `NULL` means it is estimated from `x_train`.

`internal` Not used.

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

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

`ctree.mincriterion` Numeric scalar or vector. (default = 0.95) Either a scalar or vector of length equal to the number of features in the model. 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.

`ctree.minsplit` Numeric scalar. (default = 20) Determines minimum value that the sum of the left and right daughter nodes required for a split.

`ctree.minbucket` Numeric scalar. (default = 7) Determines the minimum sum of weights in a terminal node required for a split

`ctree.sample` Boolean. (default = TRUE) If TRUE, then the method always samples `n_samples` observations from the leaf nodes (with replacement). If FALSE and the number of observations in the leaf node is less than `n_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_samples`, the method will sample `n_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_samples`.

`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.

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

`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 tuneable 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 combination/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 combination. 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(dial 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 Integer (default is internal$parameters$used_n_combinations) specifying the number of unique combinations/coalitions to apply to each training observation. Maximum allowed value is "internal$parameters$used_n_combinations - 2". By default, we use all coalitions, but this can take a lot of memory in larger dimensions. Note that by "all", we mean all coalitions chosen by shapr to be used. This will be all $2^{n_{\text{features}}}$ coalitions (minus empty and grand coalition) if shapr is in the exact mode. If the user sets a lower value than internal$parameters$used_n_combinations, then 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_vec Numeric. (Default = 2) Represents the kernel bandwidth in the distance computation. TODO: What length should it have? 1?

timeseries.bounds Numeric vector of length two. (Default = c(NULL, NULL)) 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.

Details

The most important thing to notice is that shapr has implemented eight different Monte Carlo-based approaches for estimating the conditional distributions of the data, namely "empirical", "gaussian", "copula", "ctree", "vaeac", "categorical", "timeseries", and "independence". shapr has also implemented two regression-based approaches "regression_separate" and "regression_surrogate", and see the separate vignette on the regression-based approaches for more information. In addition, the user also has the option of combining the different Monte Carlo-based approaches. E.g., if you're in a situation where you have trained a model that consists of 10 features, and you'd like to use the "gaussian" approach when you condition on a single feature, the "empirical" approach if you condition on 2-5 features, and "copula" version if you condition on more than 5 features this can be done by simply passing approach = c("gaussian", rep("empirical", 4), rep("copula", 4)). If "approach[i]" = "gaussian" means that you'd like to use the "gaussian" approach when conditioning on i features. Conditioning on all features needs no approach as that is given by the complete prediction itself, and should thus not be part of the vector.

For `approach="ctree"`, `n_samples` corresponds to the number of samples from the leaf node (see an exception related to the `sample` argument). For `approach="empirical"`, `n_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.

Value

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

shapley_values data.table with the estimated Shapley values

internal List with the different parameters, data and functions 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.

`shapley_values` is a data.table where the number of rows equals the number of observations you'd like to explain, and the number of columns equals `m +1`, where `m` equals the total number of features in your model.

If `shapley_values[i, j + 1] > 0` it indicates that the j-th feature increased the prediction for the i-th observation. Likewise, if `shapley_values[i, j + 1] < 0` it indicates that the j-th feature decreased the prediction for the i-th observation. The magnitude of the value is also important to notice. E.g. if `shapley_values[i, k + 1]` and `shapley_values[i, j + 1]` are greater than `0`, where `j != k`, and `shapley_values[i, k + 1] > shapley_values[i, j + 1]` this indicates that feature `j` and `k` both increased the value of the prediction, but that the effect of the k-th feature was larger than the `j`-th feature.

The first column in `dt`, called `none`, is the prediction value not assigned to any of the features (ϕ_0). It's equal for all observations and set by the user through the argument `prediction_zero`. The difference between the prediction and `none` is distributed among the other features. In theory this value should be the expected prediction without conditioning on any features. Typically we set this value equal to the mean of the response variable in our training data, but other choices such as the mean of the predictions in the training data are also reasonable.

Author(s)

Martin Jullum, Lars Henry Berge Olsen

References

Aas, K., Jullum, M., & L<U+00F8>land, A. (2021). Explaining individual predictions when features are dependent: More accurate approximations to Shapley values. Artificial Intelligence, 298, 103502.

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])

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

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

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

# ctree approach
explain4 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "ctree",
  prediction_zero = p,
  n_samples = 1e2
)
```

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

# Print the Shapley values
print(explain1$shapley_values)

# 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",
  prediction_zero = p,
  n_samples = 1e2
)
print(explain_groups$shapley_values)

# Separate and surrogate regression approaches with linear regression models.
# More complex regression models can be used, and we can use CV to
# tune the hyperparameters of the regression models and preprocess
# the data before sending it to the model. See the regression vignette
# (Shapley value explanations using the regression paradigm) for more
# details about the `regression_separate` and `regression_surrogate` approaches.
explain_separate_lm <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  prediction_zero = p,
  approach = "regression_separate",
  regression.model = parsnip::linear_reg()
)

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

explain_forecast *Explain a forecast from a time series model using 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.

Usage

```
explain_forecast(
  model,
  y,
  xreg = NULL,
  train_idx = NULL,
  explain_idx,
  explain_y_lags,
  explain_xreg_lags = explain_y_lags,
  horizon,
  approach,
  prediction_zero,
  n_combinations = NULL,
  group_lags = TRUE,
  group = NULL,
  n_samples = 1000,
  n_batches = NULL,
  seed = 1,
  keep_samp_for_vS = FALSE,
  predict_model = NULL,
  get_model_specs = NULL,
  timing = TRUE,
  verbose = 0,
  ...
)
```

Arguments

`model`	The model whose predictions we 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 contemporaneusly 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 `"gaussian"`, `"copula"`, `"empirical"`, `"ctree"`, `"vaeac"`, `"categorical"`, `"timeseries"`, `"independence"`, `"regression_separate"`, or `"regression_surrogate"`. The two regression approaches can not be combined with any other approach. See details for more information.
`prediction_zero`	Numeric. The prediction value for unseen data, i.e. an estimate of the expected prediction without conditioning on any features. Typically we set this value equal to the mean of the response variable in our training data, but other choices such as the mean of the predictions in the training data are also reasonable.
`n_combinations`	Integer. If `group = NULL`, `n_combinations` represents the number of unique feature combinations to sample. If `group != NULL`, `n_combinations` represents the number of unique group combinations to sample. If `n_combinations = NULL`, the exact method is used and all combinations are considered. The maximum number of combinations equals `2^m`, where `m` is the number of features.
`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. The list element contains character vectors with the features included in each of the different groups.
`n_samples`	Positive integer. Indicating the maximum number of samples to use in the Monte Carlo integration for every conditional expectation. See also details.

`n_batches` Positive integer (or NULL). Specifies how many batches the total number of feature combinations should be split into when calculating the contribution function for each test observation. The default value is NULL which uses a reasonable trade-off between RAM allocation and computation speed, which depends on `approach` and `n_combinations`. For models with many features, increasing the number of batches reduces the RAM allocation significantly. This typically comes with a small increase in computation time.

`seed` Positive integer. Specifies the seed before any randomness based code is being run. If `NULL` the seed will be inherited from the calling environment.

`keep_samp_for_vS`
Logical. Indicates whether the samples used in the Monte Carlo estimation of v_S should be returned (in `internal$output`)

`predict_model` Function. The prediction function used 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, respectively, the model and a data.frame/data.table to compute predictions for. 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 argument and provides a list with 3 elements:

labels Character vector with the names of each feature.

classes Character vector with the classes of each features.

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 the checking is disabled for unsupported model classes. Can also be used to override the default function for natively supported model classes.

`timing` Logical. Whether the timing of the different parts of the `explain()` should saved in the model object.

`verbose` An integer specifying the level of verbosity. If `0`, `shapr` will stay silent. If `1`, it will print information about performance. If `2`, some additional information will be printed out. Use `0` (default) for no verbosity, `1` for low verbose, and `2` for high verbose. TODO: Make this clearer when we end up fixing this and if they should force a progressr bar.

`...` Arguments passed on to `setup_approach.empirical`, `setup_approach.independence`, `setup_approach.gaussian`, `setup_approach.copula`, `setup_approach.ctree`, `setup_approach.vaeac`, `setup_approach.categorical`, `setup_approach.timeseries`

`empirical.type` Character. (default = `"fixed_sigma"`) Should be equal to either `"independence"`,`"fixed_sigma"`, `"AICc_each_k"` `"AICc_full"`. TODO: Describe better what the methods do here.

`empirical.eta` Numeric. (default = 0.95) Needs to be `0 < eta <= 1`. Represents the minimum proportion of the total empirical weight that data samples should use. If e.g. `eta = .8` we will choose the `K` samples with the largest weight so that the sum of the weights accounts for 80\ `eta` is the η parameter in equation (15) of Aas et al (2021).

`empirical.fixed_sigma` Positive numeric scalar. (default = 0.1) Represents the kernel bandwidth in the distance computation used when conditioning on all different combinations. Only used when `empirical.type = "fixed_sigma"`

`empirical.n_samples_aicc` Positive integer. (default = 1000) Number of samples to consider in AICc optimization. Only used for `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

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

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

`empirical.cov_mat` Numeric matrix. (Optional, default = NULL) Containing the covariance matrix of the data generating distribution used to define the Mahalanobis distance. `NULL` means it is estimated from `x_train`.

`internal` Not used.

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

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

`ctree.mincriterion` Numeric scalar or vector. (default = 0.95) Either a scalar or vector of length equal to the number of features in the model. 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.

`ctree.minsplit` Numeric scalar. (default = 20) Determines minimum value that the sum of the left and right daughter nodes required for a split.

`ctree.minbucket` Numeric scalar. (default = 7) Determines the minimum sum of weights in a terminal node required for a split

`ctree.sample` Boolean. (default = TRUE) If TRUE, then the method always samples `n_samples` observations from the leaf nodes (with replacement). If FALSE and the number of observations in the leaf node is less than `n_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_samples`, the method will sample `n_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_samples`.

`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.

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

`timeseries.fixed_sigma_vec` Numeric. (Default = 2) Represents the kernel bandwidth in the distance computation. TODO: What length should it have? 1?

`timeseries.bounds` Numeric vector of length two. (Default = c(NULL, NULL)) 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.

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.

Value

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

shapley_values data.table with the estimated Shapley values

internal List with the different parameters, data and functions 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.

`shapley_values` is a data.table where the number of rows equals the number of observations you'd like to explain, and the number of columns equals `m +1`, where `m` equals the total number of features in your model.

If `shapley_values[i, j + 1] > 0` it indicates that the j-th feature increased the prediction for the i-th observation. Likewise, if `shapley_values[i, j + 1] < 0` it indicates that the j-th feature decreased the prediction for the i-th observation. The magnitude of the value is also important to notice. E.g. if `shapley_values[i, k + 1]` and `shapley_values[i, j + 1]` are greater than `0`, where `j != k`, and `shapley_values[i, k + 1] > shapley_values[i, j + 1]` this indicates that feature `j` and `k` both increased the value of the prediction, but that the effect of the k-th feature was larger than the `j`-th feature.

The first column in `dt`, called `none`, is the prediction value not assigned to any of the features (ϕ_0). It's equal for all observations and set by the user through the argument `prediction_zero`. The difference between the prediction and `none` is distributed among the other features. In theory this value should be the expected prediction without conditioning on any features. Typically we set this value equal to the mean of the response variable in our training data, but other choices such as the mean of the predictions in the training data are also reasonable.

Author(s)

Martin Jullum, Lars Henry Berge Olsen

References

Aas, K., Jullum, M., & L<U+00F8>land, A. (2021). Explaining individual predictions when features are dependent: More accurate approximations to Shapley values. Artificial Intelligence, 298, 103502.

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",
  prediction_zero = p0_ar,
  group_lags = FALSE
)
```

---

explain_tripledot_docs   *Documentation of the approach-specific parameters in* `explain()`

Description

This helper function displays the specific arguments applicable to the different approaches. Note that when calling `explain()` from Python, the parameters are renamed from the form `approach.parameter_name` to `approach_parameter_name`. That is, an underscore has replaced the dot as the dot is reserved in Python.

Usage

```
explain_tripledot_docs(...)
```

Arguments

`...`	Arguments passed on to `setup_approach.independence`, `setup_approach.empirical`, `setup_approach.categorical`, `setup_approach.copula`, `setup_approach.ctree`, `setup_approach.gaussian`, `setup_approach.regression_separate`, `setup_approach.regression` `setup_approach.timeseries`, `setup_approach.vaeac`

`empirical.type` Character. (default = `"fixed_sigma"`) Should be equal to either `"independence"`,`"fixed_sigma"`, `"AICc_each_k"` `"AICc_full"`. TODO: Describe better what the methods do here.

`empirical.eta` Numeric. (default = 0.95) Needs to be `0 < eta <= 1`. Represents the minimum proportion of the total empirical weight that data samples should use. If e.g. `eta = .8` we will choose the `K` samples with the largest weight so that the sum of the weights accounts for 80\ `eta` is the η parameter in equation (15) of Aas et al (2021).

`empirical.fixed_sigma` Positive numeric scalar. (default = 0.1) Represents the kernel bandwidth in the distance computation used when conditioning on all different combinations. Only used when `empirical.type = "fixed_sigma"`

`empirical.n_samples_aicc` Positive integer. (default = 1000) Number of samples to consider in AICc optimization. Only used for `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

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

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

`empirical.cov_mat` Numeric matrix. (Optional, default = NULL) Containing the covariance matrix of the data generating distribution used to define the Mahalanobis distance. `NULL` means it is estimated from `x_train`.

`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 `joint_probability_dt` is not supplied, probabilities/frequencies are estimated using `x_train`. If certain observations occur in `x_train` and NOT in `x_explain`, then epsilon is used as the proportion of times that these observations occurs 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. (default = 0.95) Either a scalar or vector of length equal to the number of features in the model. 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.

`ctree.minsplit` Numeric scalar. (default = 20) Determines minimum value that the sum of the left and right daughter nodes required for a split.

`ctree.minbucket` Numeric scalar. (default = 7) Determines the minimum sum of weights in a terminal node required for a split

`ctree.sample` Boolean. (default = TRUE) If TRUE, then the method always samples `n_samples` observations from the leaf nodes (with replacement). If FALSE and the number of observations in the leaf node is less than `n_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_samples`, the method will sample `n_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_samples`.

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

`gaussian.cov_mat` Numeric matrix. (Optional) 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 tuneable 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 combination/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 combination. 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(dial 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` Integer (default is `internal$parameters$used_n_combinations`) specifying the number of unique combinations/coalitions to apply to each training observation. Maximum allowed value is "`internal$parameters$used_n_combinations` - 2". By default, we use all coalitions, but this can take a lot of memory in larger dimensions. Note that by "all", we mean all coalitions chosen by `shapr` to be used. This will be all $2^{n_{\text{features}}}$ coalitions (minus empty and grand coalition) if `shapr` is in the exact mode. If the user sets a lower value than `internal$parameters$used_n_combinations`, then 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_vec` Numeric. (Default = 2) Represents the kernel bandwidth in the distance computation. TODO: What length should it have? 1?

`timeseries.bounds` Numeric vector of length two. (Default = c(NULL, NULL)) 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.

Author(s)

Lars Henry Berge Olsen and Martin Jullum

`feature_combinations` *Define feature combinations, and fetch additional information about each unique combination*

Description

Define feature combinations, and fetch additional information about each unique combination

Usage

```
feature_combinations(
  m,
  exact = TRUE,
  n_combinations = 200,
  weight_zero_m = 10^6,
  group_num = NULL
)
```

Arguments

`m`	Positive integer. Total number of features.
`exact`	Logical. If `TRUE` all `2^m` combinations are generated, otherwise a subsample of the combinations is used.

`n_combinations`	Positive integer. Note that if `exact = TRUE`, `n_combinations` is ignored. However, if `m > 12` you'll need to add a positive integer value for `n_combinations`.
`weight_zero_m`	Numeric. The value to use as a replacement for infinite combination weights when doing numerical operations.
`group_num`	List. Contains vector of integers indicating the feature numbers for the different groups.

Value

A data.table that contains the following columns:

id_combination Positive integer. Represents a unique key for each combination. Note that the table is sorted by `id_combination`, so that is always equal to `x[["id_combination"]] = 1:nrow(x)`.

features List. Each item of the list is an integer vector where `features[[i]]` represents the indices of the features included in combination `i`. Note that all the items are sorted such that `features[[i]] == sort(features[[i]])` is always true.

n_features Vector of positive integers. `n_features[i]` equals the number of features in combination `i`, i.e. `n_features[i] = length(features[[i]])`..

N Positive integer. The number of unique ways to sample `n_features[i]` features from `m` different features, without replacement.

Author(s)

Nikolai Sellereite, Martin Jullum

Examples

```
# All combinations
x <- feature_combinations(m = 3)
nrow(x) # Equals 2^3 = 8

# Subsample of combinations
x <- feature_combinations(exact = FALSE, m = 10, n_combinations = 1e2)
```

`finalize_explanation` *Computes the Shapley values given* `v(S)`

Description

Computes dependence-aware Shapley values for observations in `x_explain` from the specified `model` by using the method specified in `approach` to estimate the conditional expectation.

Usage

```
finalize_explanation(vS_list, internal)
```

Arguments

`vS_list`	List Output from `compute_vS()`
`internal`	List. Holds all parameters, data, functions and computed objects used within `explain()` The list contains one or more of the elements `parameters`, `data`, `objects`, `output`.

Details

The most important thing to notice is that `shapr` has implemented eight different Monte Carlo-based approaches for estimating the conditional distributions of the data, namely `"empirical"`, `"gaussian"`, `"copula"`, `"ctree"`, `"vaeac"`, `"categorical"`, `"timeseries"`, and `"independence"`. `shapr` has also implemented two regression-based approaches `"regression_separate"` and `"regression_surrogate"`, and see the separate vignette on the regression-based approaches for more information. In addition, the user also has the option of combining the different Monte Carlo-based approaches. E.g., if you're in a situation where you have trained a model that consists of 10 features, and you'd like to use the `"gaussian"` approach when you condition on a single feature, the `"empirical"` approach if you condition on 2-5 features, and `"copula"` version if you condition on more than 5 features this can be done by simply passing `approach = c("gaussian", rep("empirical", 4), rep("copula", 4))`. If `"approach[i]" = "gaussian"` means that you'd like to use the `"gaussian"` approach when conditioning on `i` features. Conditioning on all features needs no approach as that is given by the complete prediction itself, and should thus not be part of the vector.

For `approach="ctree"`, `n_samples` corresponds to the number of samples from the leaf node (see an exception related to the `sample` argument). For `approach="empirical"`, `n_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.

Value

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

shapley_values data.table with the estimated Shapley values

internal List with the different parameters, data and functions 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.

`shapley_values` is a data.table where the number of rows equals the number of observations you'd like to explain, and the number of columns equals `m +1`, where `m` equals the total number of features in your model.

If `shapley_values[i, j + 1] > 0` it indicates that the j-th feature increased the prediction for the i-th observation. Likewise, if `shapley_values[i, j + 1] < 0` it indicates that the j-th feature decreased the prediction for the i-th observation. The magnitude of the value is also important to notice. E.g. if `shapley_values[i, k + 1]` and `shapley_values[i, j + 1]` are greater than `0`, where `j != k`, and `shapley_values[i, k + 1] > shapley_values[i, j + 1]` this indicates that feature `j` and `k` both increased the value of the prediction, but that the effect of the k-th feature was larger than the j-th feature.

The first column in `dt`, called `none`, is the prediction value not assigned to any of the features (ϕ_0). It's equal for all observations and set by the user through the argument `prediction_zero`. The difference between the prediction and none is distributed among the other features. In theory this value should be the expected prediction without conditioning on any features. Typically we set this value equal to the mean of the response variable in our training data, but other choices such as the mean of the predictions in the training data are also reasonable.

Author(s)

Martin Jullum, Lars Henry Berge Olsen

References

Aas, K., Jullum, M., & L<U+00F8>land, A. (2021). Explaining individual predictions when features are dependent: More accurate approximations to Shapley values. Artificial Intelligence, 298, 103502.

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])

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

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

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

# ctree approach
explain4 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "ctree",
  prediction_zero = p,
  n_samples = 1e2
)

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

# Print the Shapley values
print(explain1$shapley_values)

# 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",
  prediction_zero = p,
  n_samples = 1e2
)
print(explain_groups$shapley_values)

# Separate and surrogate regression approaches with linear regression models.
# More complex regression models can be used, and we can use CV to
# tune the hyperparameters of the regression models and preprocess
# the data before sending it to the model. See the regression vignette
# (Shapley value explanations using the regression paradigm) for more
# details about the `regression_separate` and `regression_surrogate` approaches.
explain_separate_lm <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  prediction_zero = p,
  approach = "regression_separate",
  regression.model = parsnip::linear_reg()
)

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

get_cov_mat *get_cov_mat*

Description

get_cov_mat

Usage

```
get_cov_mat(x_train, min_eigen_value = 1e-06)
```

Arguments

`x_train`	Matrix or data.frame/data.table. Contains the data used to estimate the (conditional) distributions for the features needed to properly estimate the conditional expectations in the Shapley formula.
`min_eigen_value`	Numeric Specifies the smallest allowed eigen value before the covariance matrix of `x_train` is assumed to not be positive definite, and `Matrix::nearPD()` is used to find the nearest one.

`get_data_forecast` *Set up data for explain_forecast*

Description

Set up data for explain_forecast

Usage

```
get_data_forecast(
  y,
  xreg,
  train_idx,
  explain_idx,
  explain_y_lags,
  explain_xreg_lags,
  horizon
)
```

Arguments

`y`	A matrix or numeric vector containing the endogenous variables for the model. One variable per column, one observation per row.
`xreg`	A matrix containing exogenous regressors for the model. One variable per column, one observation per row. Should have nrow(data) + horizon rows.
`train_idx`	The observations indices in data to use as training examples.
`explain_idx`	The observations indices in data to explain.
`explain_y_lags`	Numeric vector Indicates the number of lags of y to include in the explanation.
`explain_xreg_lags`	Numeric vector Indicates the number of lags of xreg to include in the explanation.
`horizon`	The forecast horizon to explain.

Value

A list containing

- The data.frames x_train and x_explain which holds the lagged data examples.
- A numeric, n_endo denoting how many columns are endogenous in x_train and x_explain.
- A list, group with groupings of each variable to explain per variable and not per variable and lag.

`get_mu_vec` *get_mu_vec*

Description

get_mu_vec

Usage

```
get_mu_vec(x_train)
```

Arguments

`x_train`	Matrix or data.frame/data.table. Contains the data used to estimate the (conditional) distributions for the features needed to properly estimate the conditional expectations in the Shapley formula.

`get_supported_approaches` *Gets the implemented approaches*

Description

Gets 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()`.

lag_data *Lag a matrix of variables a specific number of lags for each variables.*

Description

Lag a matrix of variables a specific number of lags for each variables.

Usage

```
lag_data(x, lags)
```

Arguments

`x`	The matrix of variables (one variable per column).
`lags`	A numeric vector denoting how many lags each variable should have.

Value

A list with two items

- A matrix, lagged with the lagged data.
- A list, group, with groupings of the lagged data per variable.

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,
  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,
  ...
)
```

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 features 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"` summarises 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.
`digits`	Integer. Number of significant digits to use in the feature description. Applicable for `plot_type` `"bar"` and `"waterfall"`
`index_x_explain`	Integer vector. Which of the test observations to plot. E.g. if you have explained 10 observations using `explain()`, you can generate a plot for the first 5 observations by setting `index_x_explain = 1:5`.
`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 (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 i 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 what features to include in (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.
`...`	Currently not used.

Details

See the examples below, or `vignette("understanding_shapr", package = "shapr")` for an examples of how you should use the function.

Value

ggplot object with plots of the Shapley value explanations

Author(s)

Martin Jullum, Vilde Ung

Examples

```
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",
  prediction_zero = p,
  n_samples = 1e2
)

if (requireNamespace("ggplot2", 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 2 features with largest contribution
  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
}

# 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",
  prediction_zero = p,
  n_samples = 1e2
)
```

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

---

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 combination by only averaging over the combinations and observations/explicands, respectively.

Usage

```
plot_MSEv_eval_crit(
  explanation_list,
  index_x_explain = NULL,
  id_combination = 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. E.g. if you have explained 10 observations using explain(), you can generate a plot for the first 5 observations by setting index_x_explain = 1:5.
id_combination	Integer vector. Which of the combinations (coalitions) to plot. E.g. if you used n_combinations = 16 in explain(), you can generate a plot for the first 5 combinations and the 10th by setting id_combination = 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_combination. 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 $\pm$ 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* $\pm$ *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), "comb", 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 combinations/coalitions and observations/explicands. If `plot_type = "comb"`, then the plots (one line plot and one bar plot) associated with the MSEv evaluation criterion for each combination/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 combinations/coalitions. If `plot_type` is a vector of one or several of "overall", "comb", and "explicand", then the associated plots are created.

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

```
# Load necessary librarieslibrary(xgboost)
library(data.table)
library(shapr)
library(ggplot2)

# 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(
  data = as.matrix(x_train),
  label = y_train,
  nround = 20,
  verbose = FALSE
)

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

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

# Gaussian 1e1 approach
explanation_gaussian_1e1 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "gaussian",
  prediction_zero = prediction_zero,
  n_samples = 1e1
)

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

# ctree approach
explanation_ctree <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "ctree",
  prediction_zero = prediction_zero,
  n_samples = 1e2
)
```

```
# Combined approach
explanation_combined <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = c("gaussian", "independence", "ctree"),
  prediction_zero = prediction_zero,
  n_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
)

if (requireNamespace("ggplot2", quietly = TRUE)) {
  # Create the default MSEv plot where we average over both the combinations 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 combinations or observations.
  MSEv_figures <- plot_MSEv_eval_crit(explanation_list_named,
    CI_level = 0.95,
    plot_type = c("overall", "comb", "explicand")
  )
  MSEv_figures$MSEv_bar
  MSEv_figures$MSEv_combination_bar
  MSEv_figures$MSEv_explicand_bar

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

  # We can specify which observations or combinations to plot
  plot_MSEv_eval_crit(explanation_list_named,
    plot_type = "explicand",
    index_x_explain = c(1, 3:4, 6),
    CI_level = 0.95
  )$MSEv_explicand_bar
  plot_MSEv_eval_crit(explanation_list_named,
    plot_type = "comb",
    id_combination = c(3, 4, 9, 13:15),
    CI_level = 0.95
  )$MSEv_combination_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.

Usage

```
plot_SV_several_approaches(
  explanation_list,
  index_explicands = NULL,
  only_these_features = NULL,
  plot_phi0 = FALSE,
  digits = 4,
  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
)
```

Arguments

`explanation_list`
: A list of `explain()` objects applied to the same data and model. 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`.

`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)`.

`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.

`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

Value

A `ggplot2::ggplot()` object.

Author(s)

Lars Henry Berge Olsen

Examples

```
# Load necessary libraries
library(xgboost)
library(data.table)

# 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(
  data = as.matrix(x_train),
  label = y_train,
  nround = 20,
  verbose = FALSE
)

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

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

# Empirical approach
explanation_empirical <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "empirical",
  prediction_zero = prediction_zero,
  n_samples = 1e2
)

# Gaussian 1e1 approach
explanation_gaussian_1e1 <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "gaussian",
  prediction_zero = prediction_zero,
  n_samples = 1e1
)

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

# Combined approach
explanation_combined <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = c("gaussian", "ctree", "empirical"),
  prediction_zero = prediction_zero,
  n_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
)

if (requireNamespace("ggplot2", quietly = TRUE)) {
  # 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
  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
  ) +
    ggplot2::theme_minimal() +
   ggplot2::theme(legend.position = "bottom", plot.title = ggplot2::element_text(size = 0))


 # 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
  )
}
```

process_factor_data *Treat factors as numeric values*

Description

Factors are given a numeric value above the highest numeric value in the data. The value of the different levels are sorted by factor and then level.

Usage

```
process_factor_data(dt, factor_cols)
```

Arguments

`dt`	data.table to plot
`factor_cols`	Columns that are factors or character

Value

A list of a lookup table with each factor and level and its numeric value, a data.table very similar to the input data, but now with numeric values for factors, and the maximum feature value.

`reg_forecast_setup` *Set up exogenous regressors for explanation in a forecast model.*

Description

Set up exogenous regressors for explanation in a forecast model.

Usage

```
reg_forecast_setup(x, horizon, group)
```

Arguments

`x`	A matrix with the exogenous variables.
`horizon`	The forecast horizon.
`group`	The list of endogenous groups, to append exogenous groups to.

Value

A list containing

- fcast A matrix containing the exogenous observations needed for each observation.
- group The list group with the exogenous groups appended.

`release_questions` *Auxiliary function for the vaeac vignette*

Description

Function that question if the main and vaeac vignette has been built using the `rebuild-long-running-vignette.R` function. This is only useful when using devtools to release `shapr` to cran. See `devtools::release()` for more information.

Usage

```
release_questions()
```

setup *check_setup*

Description

check_setup

Usage

```
setup(
  x_train,
  x_explain,
  approach,
  prediction_zero,
  output_size = 1,
  n_combinations,
  group,
  n_samples,
  n_batches,
  seed,
  keep_samp_for_vS,
  feature_specs,
  MSEv_uniform_comb_weights = TRUE,
  type = "normal",
  horizon = NULL,
  y = NULL,
  xreg = NULL,
  train_idx = NULL,
  explain_idx = NULL,
  explain_y_lags = NULL,
  explain_xreg_lags = NULL,
  group_lags = NULL,
  timing,
  verbose,
  is_python = FALSE,
  ...
)
```

Arguments

`x_train`	Matrix or data.frame/data.table. Contains the data used to estimate the (conditional) distributions for the features needed to properly estimate the conditional expectations in the Shapley formula.
`x_explain`	A matrix or data.frame/data.table. Contains the the features, whose predictions ought to be explained.

`approach`	Character vector of length `1` or one less than the number of features. All elements should, either be `"gaussian"`, `"copula"`, `"empirical"`, `"ctree"`, `"vaeac"`, `"categorical"`, `"timeseries"`, `"independence"`, `"regression_separate"`, or `"regression_surrogate"`. The two regression approaches can not be combined with any other approach. See details for more information.
`prediction_zero`	Numeric. The prediction value for unseen data, i.e. an estimate of the expected prediction without conditioning on any features. Typically we set this value equal to the mean of the response variable in our training data, but other choices such as the mean of the predictions in the training data are also reasonable.
`output_size`	TODO: Document
`n_combinations`	Integer. If `group = NULL`, `n_combinations` represents the number of unique feature combinations to sample. If `group != NULL`, `n_combinations` represents the number of unique group combinations to sample. If `n_combinations = NULL`, the exact method is used and all combinations are considered. The maximum number of combinations equals `2^m`, where `m` is the number of 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. The list element contains character vectors with the features included in each of the different groups.
`n_samples`	Positive integer. Indicating the maximum number of samples to use in the Monte Carlo integration for every conditional expectation. See also details.
`n_batches`	Positive integer (or NULL). Specifies how many batches the total number of feature combinations should be split into when calculating the contribution function for each test observation. The default value is NULL which uses a reasonable trade-off between RAM allocation and computation speed, which depends on `approach` and `n_combinations`. For models with many features, increasing the number of batches reduces the RAM allocation significantly. This typically comes with a small increase in computation time.
`seed`	Positive integer. Specifies the seed before any randomness based code is being run. If `NULL` the seed will be inherited from the calling environment.
`keep_samp_for_vS`	Logical. Indicates whether the samples used in the Monte Carlo estimation of v_S should be returned (in `internal$output`)
`feature_specs`	List. The output from `get_model_specs()` or `get_data_specs()`. Contains the 3 elements: **labels** Character vector with the names of each feature. **classes** Character vector with the classes of each features. **factor_levels** Character vector with the levels for any categorical features.
`MSEv_uniform_comb_weights`	Logical. If `TRUE` (default), then the function weights the combinations uniformly when computing the MSEv criterion. If `FALSE`, then the function use the Shapley kernel weights to weight the combinations when computing the MSEv criterion. Note that the Shapley kernel weights are replaced by the sampling frequency when not all combinations are considered.

`type`	Character. Either "normal" or "forecast" corresponding to function `setup()` is called from, correspondingly the type of explanation that should be generated.
`horizon`	Numeric. The forecast horizon to explain. Passed to the `predict_model` function.
`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 contemporaneusly 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.
`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.
`timing`	Logical. Whether the timing of the different parts of the `explain()` should saved in the model object.
`verbose`	An integer specifying the level of verbosity. If `0`, `shapr` will stay silent. If `1`, it will print information about performance. If `2`, some additional information will be printed out. Use `0` (default) for no verbosity, `1` for low verbose, and `2` for high verbose. TODO: Make this clearer when we end up fixing this and if they should force a progressr bar.
`is_python`	Logical. Indicates whether the function is called from the Python wrapper. Default is FALSE which is never changed when calling the function via `explain()` in R. The parameter is later used to disallow running the AICc-versions of the empirical as that requires data based optimization.
`...`	Further arguments passed to specific approaches

`setup_approach` *Set up the framework chosen approach*

Description

The different choices of `approach` takes different (optional) parameters, which are forwarded from `explain()`.

Usage

```
setup_approach(internal, ...)

## S3 method for class 'categorical'
setup_approach(
  internal,
  categorical.joint_prob_dt = NULL,
  categorical.epsilon = 0.001,
  ...
)

## S3 method for class 'copula'
setup_approach(internal, ...)

## S3 method for class 'ctree'
setup_approach(
  internal,
  ctree.mincriterion = 0.95,
  ctree.minsplit = 20,
  ctree.minbucket = 7,
  ctree.sample = TRUE,
  ...
)

## S3 method for class 'empirical'
setup_approach(
  internal,
  empirical.type = "fixed_sigma",
  empirical.eta = 0.95,
  empirical.fixed_sigma = 0.1,
  empirical.n_samples_aicc = 1000,
  empirical.eval_max_aicc = 20,
  empirical.start_aicc = 0.1,
  empirical.cov_mat = NULL,
  model = NULL,
  predict_model = NULL,
  ...
)

## S3 method for class 'gaussian'
setup_approach(internal, gaussian.mu = NULL, gaussian.cov_mat = NULL, ...)

## S3 method for class 'independence'
setup_approach(internal, ...)

## S3 method for class 'regression_separate'
setup_approach(
  internal,
  regression.model = parsnip::linear_reg(),
  regression.tune_values = NULL,
  regression.vfold_cv_para = NULL,
  regression.recipe_func = NULL,
  ...
)

## S3 method for class 'regression_surrogate'
setup_approach(
  internal,
  regression.model = parsnip::linear_reg(),
  regression.tune_values = NULL,
  regression.vfold_cv_para = NULL,
  regression.recipe_func = NULL,
  regression.surrogate_n_comb = internal$parameters$used_n_combinations - 2,
  ...
)

## S3 method for class 'timeseries'
setup_approach(
  internal,
  timeseries.fixed_sigma_vec = 2,
  timeseries.bounds = c(NULL, NULL),
  ...
)

## S3 method for class 'vaeac'
setup_approach(
  internal,
  vaeac.depth = 3,
  vaeac.width = 32,
  vaeac.latent_dim = 8,
  vaeac.activation_function = torch::nn_relu,
  vaeac.lr = 0.001,
  vaeac.n_vaeacs_initialize = 4,
  vaeac.epochs = 100,
  vaeac.extra_parameters = list(),
  ...
)
```

Arguments

`internal`	Not used.
`...`	approach-specific arguments. See below.
`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 `joint_probability_dt` is not supplied, probabilities/frequencies are estimated using `x_train`. If certain observations occur in `x_train` and NOT in `x_explain`, then epsilon is used as the proportion of times that these observations occurs 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. (default = 0.95) Either a scalar or vector of length equal to the number of features in the model. 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.

`ctree.minsplit`
: Numeric scalar. (default = 20) Determines minimum value that the sum of the left and right daughter nodes required for a split.

`ctree.minbucket`
: Numeric scalar. (default = 7) Determines the minimum sum of weights in a terminal node required for a split

`ctree.sample`
: Boolean. (default = TRUE) If TRUE, then the method always samples `n_samples` observations from the leaf nodes (with replacement). If FALSE and the number of observations in the leaf node is less than `n_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_samples`, the method will sample `n_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_samples`.

`empirical.type`
: Character. (default = `"fixed_sigma"`) Should be equal to either `"independence"`,`"fixed_sigma"`, `"AICc_each_k"` `"AICc_full"`. TODO: Describe better what the methods do here.

`empirical.eta`
: Numeric. (default = 0.95) Needs to be `0 < eta <= 1`. Represents the minimum proportion of the total empirical weight that data samples should use. If e.g. `eta = .8` we will choose the `K` samples with the largest weight so that the sum of the weights accounts for 80\ `eta` is the η parameter in equation (15) of Aas et al (2021).

`empirical.fixed_sigma`
: Positive numeric scalar. (default = 0.1) Represents the kernel bandwidth in the distance computation used when conditioning on all different combinations. Only used when `empirical.type` = `"fixed_sigma"`

`empirical.n_samples_aicc`
: Positive integer. (default = 1000) Number of samples to consider in AICc optimization. Only used for `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

`empirical.eval_max_aicc`
: Positive integer. (default = 20) Maximum number of iterations when optimizing the AICc. Only used for `empirical.type` is either `"AICc_each_k"` or `"AICc_full"`.

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

`empirical.cov_mat`	Numeric matrix. (Optional, default = NULL) Containing the covariance matrix of the data generating distribution used to define the Mahalanobis distance. `NULL` means it is estimated from `x_train`.
`model`	Objects. The model object that ought to be explained. See the documentation of `explain()` for details.
`predict_model`	Function. The prediction function used when `model` is not natively supported. See the documentation of `explain()` for details.
`gaussian.mu`	Numeric vector. (Optional) Containing the mean of the data generating distribution. `NULL` means it is estimated from the `x_train`.
`gaussian.cov_mat`	Numeric matrix. (Optional) 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, engine = "ranger", mode = "re` 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 tuneable 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 combination/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 combination. 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::m 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`
: Integer (default is `internal$parameters$used_n_combinations`) specifying the number of unique combinations/coalitions to apply to each training observation. Maximum allowed value is "`internal$parameters$used_n_combinations - 2`". By default, we use all coalitions, but this can take a lot of memory in larger dimensions. Note that by "all", we mean all coalitions chosen by `shapr` to be used. This will be all $2^{n_{\text{features}}}$ coalitions (minus empty and grand coalition) if `shapr` is in the exact mode. If the user sets a lower value than `internal$parameters$used_n_combinations`, then 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_vec`
: Numeric. (Default = 2) Represents the kernel bandwidth in the distance computation. TODO: What length should it have? 1?

`timeseries.bounds`
: Numeric vector of length two. (Default = c(NULL, NULL)) 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.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.lr`
: Positive numeric (default is `0.001`). The learning rate used in the `torch::optim_adam()` optimizer.

`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.

Author(s)

Martin Jullum

Lars Henry Berge Olsen

`setup_computation` *Sets up everything for the Shapley values computation in* `explain()`

Description

Sets up everything for the Shapley values computation in `explain()`

Usage

```
setup_computation(internal, model, predict_model)
```

Arguments

`internal`	List. Holds all parameters, data, functions and computed objects used within `explain()` The list contains one or more of the elements `parameters`, `data`, `objects`, `output`.
`model`	Objects. The model object that ought to be explained. See the documentation of `explain()` for details.
`predict_model`	Function. The prediction function used when `model` is not natively supported. See the documentation of `explain()` for details.

Value

List `internal` It holds all parameters, data, and computed objects used within `explain()`. The list contains one or more of the elements `parameters`, `data`, `objects`, `output`.

`vaeac_get_data_objects` *Function to set up data loaders and save file names*

Description

Function to set up data loaders and save file names

Usage

```
vaeac_get_data_objects(
  x_train,
  log_exp_cont_feat,
  val_ratio,
  batch_size,
  paired_sampling,
  model_description,
  depth,
  width,
  latent_dim,
  lr,
  epochs,
  save_every_nth_epoch,
  folder_to_save_model,
  train_indices = NULL,
  val_indices = NULL
)
```

Arguments

`x_train`	A data.table containing the training data. Categorical data must have class names $1, 2, \dots, K$.
`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.
`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, `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.
`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()`.
`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.
`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.
`depth`	Positive integer (default is `3`). The number of hidden layers in the neural networks of the masked encoder, full encoder, and decoder.
`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.

`latent_dim`	Positive integer (default is `8`). The number of dimensions in the latent space.
`lr`	Positive numeric (default is `0.001`). The learning rate used in the `torch::optim_adam()` optimizer.
`epochs`	Positive integer (default is `100`). The number of epochs to train the final vaeac model. This includes `epochs_initiation_phase`, where the default is `2`.
`save_every_nth_epoch`	Positive integer (default is `NULL`). If provided, then the vaeac model after every `save_every_nth_epoch`th epoch will be saved.
`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`.
`train_indices`	Numeric array (optional) containing the indices of the training observations. There are conducted no checks to validdate the indices.
`val_indices`	Numeric array (optional) containing the indices of the validation observations. #' There are conducted no checks to validdate the indices.

Value

List of objects needed to train the `vaeac` model

`vaeac_get_evaluation_criteria`

Extract the Training VLB and Validation IWAE from a list of explanations objects using the vaeac approach

---

Description

Extract the Training VLB and Validation IWAE from a list of explanations objects using the vaeac approach

Usage

```
vaeac_get_evaluation_criteria(explanation_list)
```

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`.

Value

A data.table containing the training VLB, validation IWAE, and running validation IWAE at each epoch for each vaeac model.

Author(s)

Lars Henry Berge Olsen

---

vaeac_get_extra_para_default

Function to 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`, the we fall back to use CPU. If `FALSE`, we use 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_epoch`th 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/combinations and all explicands sent to `explain()` at the time. The number of coalitions are determined by `n_batches` in `explain()`. We recommend to tweak `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/combinations,

i.e., when n_combinations $< 2^{n_{\text{features}}}$, 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 but 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

`vaeac_plot_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

```
vaeac_plot_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

Examples

```
## Not run:
library(xgboost)
library(data.table)
library(shapr)

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(data = as.matrix(x_train), label = y_train, nround = 100, verbose = FALSE)

# 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 = approach,
  prediction_zero = p0,
  n_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 = approach,
  prediction_zero = p0,
  n_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
vaeac_plot_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.
vaeac_plot_eval_crit(
  explanation_list = explanation_list[2],
  plot_type = "method"
)

# Can alter the plot
vaeac_plot_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
vaeac_plot_eval_crit(
  explanation_list = explanation_list,
  criteria = "VLB",
  plot_type = "criterion"
)

# If we want only want the criterion version
tmp_fig_criterion <-
  vaeac_plot_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()

## End(Not run)
```

vaeac_plot_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

```
vaeac_plot_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

Examples

```
## Not run:
library(xgboost)
library(data.table)
library(shapr)

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(
  data = as.matrix(x_train),
  label = y_train,
  nround = 100,
  verbose = FALSE
)

explanation <- explain(
  model = model,
  x_explain = x_explain,
  x_train = x_train,
  approach = "vaeac",
  prediction_zero = mean(y_train),
  n_samples = 1,
  vaeac.epochs = 10,
  vaeac.n_vaeacs_initialize = 1
)

# Plot the results
figure <- vaeac_plot_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"))

## End(Not run)
```

vaeac_train_model *Train the Vaeac Model*

Description

Function that fits a vaeac model to the given dataset based on the provided parameters, as described in Olsen et al. (2022). Note that all default parameters specified below origin from `setup_approach.vaeac()` and `vaeac_get_extra_para_default()`.

Usage

```
vaeac_train_model(
  x_train,
  model_description,
  folder_to_save_model,
  cuda,
  n_vaeacs_initialize,
  epochs_initiation_phase,
  epochs,
  epochs_early_stopping,
  save_every_nth_epoch,
  val_ratio,
  val_iwae_n_samples,
  depth,
  width,
  latent_dim,
  lr,
  batch_size,
  running_avg_n_values,
  activation_function,
  skip_conn_layer,
  skip_conn_masked_enc_dec,
  batch_normalization,
  paired_sampling,
  masking_ratio,
  mask_gen_coalitions,
  mask_gen_coalitions_prob,
  sigma_mu,
  sigma_sigma,
  save_data,
  log_exp_cont_feat,
  which_vaeac_model,
  verbose,
  seed,
  ...
)
```

Arguments

`x_train` A data.table containing the training data. Categorical data must have class names $1, 2, \ldots, K$.

`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.

`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`.

`cuda` Logical (default is `FALSE`). If `TRUE`, then the `vaeac` model will be trained using cuda/GPU. If `torch::cuda_is_available()` is `FALSE`, the we fall back to use CPU. If `FALSE`, we use 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).

`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 `epochs_initiation_phase` epochs (default is `2`) and continue training that one.

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

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

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

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

`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, `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.
`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.
`depth`	Positive integer (default is `3`). The number of hidden layers in the neural networks of the masked encoder, full encoder, and decoder.
`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.
`latent_dim`	Positive integer (default is `8`). The number of dimensions in the latent space.
`lr`	Positive numeric (default is `0.001`). The learning rate used in the `torch::optim_adam()` optimizer.
`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()`.
`running_avg_n_values`	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.
`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()`.
`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 $X + activation(WX + b)$.
`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.
`batch_normalization`	Logical (default is `FALSE`). If `TRUE`, we apply batch normalization after the activation function. Note that if `skip_conn_layer = TRUE`, then the normalization is applied after the inclusion of the skip connection. That is, we batch normalize the whole quantity $X + activation(WX + b)$.
`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.

`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. `masking_ratio` will be overruled if `mask_gen_coalitions` is specified.
`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/combinations, i.e., when `n_combinations` $< 2^{n_{\text{features}}}$, and for group Shapley, i.e., when `group` is specified in `explain()`.
`mask_gen_coalitions_prob`	Numeric array (default is `NULL`). Array of length equal to the height of `mask_gen_coalitions` containing the probabilities of sampling the corresponding coalitions in `mask_gen_coalitions`.
`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).
`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).
`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()`.
`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.
`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.
`verbose`	Boolean. An integer specifying the level of verbosity. Use `0` (default) for no verbosity, `1` for low verbose, and `2` for high verbose.
`seed`	Positive integer (default is `1`). Seed for reproducibility. Specifies the seed before any randomness based code is being run.
`...`	List of extra parameters, currently not used.

Details

The vaeac model consists of three neural networks, i.e., a masked encoder, a full encoder, and a decoder. The networks have shared `depth`, `width`, and `activation_function`. The encoders maps the `x_train` to a latent representation of dimension `latent_dim`, while the decoder maps the latent representations back to the feature space. See Olsen et al. (2022) for more details. The function first initiates `n_vaeacs_initialize` vaeac models with different randomly initiated network parameter values to remedy poorly initiated values. After `epochs_initiation_phase` epochs, the `n_vaeacs_initialize` vaeac models are compared and the function continues to only train the best performing one for a total of `epochs` epochs. The networks are trained using the ADAM optimizer with the learning rate is `lr`.

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

`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 = 0,
  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 $1, 2, \dots, K$.
`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`	Boolean. An integer specifying the level of verbosity. Use `0` (default) for no verbosity, `1` for low verbose, and `2` for high verbose.
`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

Index