Package 'kernelshap'

January 11, 2023

Title Kernel SHAP

Version 0.3.3

Description Multidimensional refinement of the Kernel SHAP algorithm described in Ian Covert and Su-In Lee (2021) <http://proceedings.mlr.press/v130/covert21a>. The package allows to calculate Kernel SHAP values in an exact way, by iterative sampling (as in the reference above), or by a hybrid of the two. As soon as sampling is involved, the algorithm iterates until convergence, and standard errors are provided. The package works with any model that provides numeric predictions of dimension one or higher. Examples include linear regression, logistic regression (on logit or probability scale), other generalized linear models, generalized additive models, and neural networks. The package plays well together with meta-learning packages like 'tidymodels', 'caret' or 'mlr3'. Visualizations can be done using the R package 'shapviz'.

License GPL (>= 2)

Depends R (>= 3.2.0)

Encoding UTF-8

RoxygenNote 7.2.1

Imports foreach, stats, utils

Suggests doFuture, testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/mayer79/kernelshap

BugReports https://github.com/mayer79/kernelshap/issues

NeedsCompilation no

Author Michael Mayer [aut, cre],
David Watson [ctb]

Maintainer Michael Mayer <mayermichael79@gmail.com>

Repository CRAN

Date/Publication 2023-01-11 17:30:02 UTC

R topics documented:

`is.kernelshap` *Check for kernelshap*

Description

Is object of class "kernelshap"?

Usage

```
is.kernelshap(object)
```

Arguments

`object`	An R object.

Value

Returns `TRUE` if `object` has "`kernelshap`" among its classes, and `FALSE` otherwise.

Examples

```
fit <- stats::lm(Sepal.Length ~ ., data = iris)
s <- kernelshap(fit, iris[1:2, -1], bg_X = iris[-1])
is.kernelshap(s)
is.kernelshap("a")
```

kernelshap *Kernel SHAP*

Description

Multidimensional refinement of the Kernel SHAP Algorithm described in Covert and Lee (2021), in the following abbreviated by "CL21". The function allows to calculate Kernel SHAP values in an exact way, by iterative sampling as in CL21, or by a hybrid of these two options. As soon as sampling is involved, the algorithm iterates until convergence, and standard errors are provided. The default behaviour depends on the number of features p, see also Details below:

- 2 <= p <= 8: Exact Kernel SHAP values are returned (for the given background data).
- p > 8: Hybrid (partly exact) iterative version of Kernel SHAP (degree 2 up to p = 16, degree 1 for p > 16, see Details).
- p = 1: Exact Shapley values are returned (independent of the background data).

Usage

```
kernelshap(object, ...)

## Default S3 method:
kernelshap(
  object,
  X,
  bg_X,
  pred_fun = stats::predict,
  feature_names = colnames(X),
  bg_w = NULL,
  exact = length(feature_names) <= 8L,
  hybrid_degree = 1L + length(feature_names) %in% 4:16,
  paired_sampling = TRUE,
  m = 2L * length(feature_names) * (1L + 3L * (hybrid_degree == 0L)),
  tol = 0.005,
  max_iter = 100L,
  parallel = FALSE,
  parallel_args = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'ranger'
kernelshap(
  object,
  X,
  bg_X,
  pred_fun = function(m, X, ...) stats::predict(m, X, ...)$predictions,
```

```
  feature_names = colnames(X),
  bg_w = NULL,
  exact = length(feature_names) <= 8L,
  hybrid_degree = 1L + length(feature_names) %in% 4:16,
  paired_sampling = TRUE,
  m = 2L * length(feature_names) * (1L + 3L * (hybrid_degree == 0L)),
  tol = 0.005,
  max_iter = 100L,
  parallel = FALSE,
  parallel_args = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'Learner'
kernelshap(
  object,
  X,
  bg_X,
  pred_fun = function(m, X) m$predict_newdata(X)$response,
  feature_names = colnames(X),
  bg_w = NULL,
  exact = length(feature_names) <= 8L,
  hybrid_degree = 1L + length(feature_names) %in% 4:16,
  paired_sampling = TRUE,
  m = 2L * length(feature_names) * (1L + 3L * (hybrid_degree == 0L)),
  tol = 0.005,
  max_iter = 100L,
  parallel = FALSE,
  parallel_args = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	Fitted model object.
`...`	Additional arguments passed to `pred_fun(object, X, ...)`.
`X`	A (n x p) matrix, data.frame, tibble or data.table of rows to be explained. The columns should only represent model features, not the response.
`bg_X`	Background data used to integrate out "switched off" features, often a subset of the training data (typically 50 to 500 rows) It should contain the same columns as `X`. In cases with a natural "off" value (like MNIST digits), this can also be a single row with all values set to the off value.
`pred_fun`	Prediction function of the form `function(object, X, ...)`, providing K >= 1 numeric predictions per row. Its first argument represents the model `object`, its second argument a data structure like `X`. Additional (named) arguments are passed via `...`. The default, `stats::predict`, will work in most cases.

feature_names
: Optional vector of column names in X used to calculate SHAP values. By default, this equals colnames(X). Not supported if X is a matrix.

bg_w
: Optional vector of case weights for each row of bg_X.

exact
: If TRUE, the algorithm will produce exact Kernel SHAP values with respect to the background data. In this case, the arguments hybrid_degree, m, paired_sampling, tol, and max_iter are ignored. The default is TRUE up to eight features, and FALSE otherwise.

hybrid_degree
: Integer controlling the exactness of the hybrid strategy. For 4 <= p <= 16, the default is 2, otherwise it is 1. Ignored if exact = TRUE.

  - 0: Pure sampling strategy not involving any exact part. It is strictly worse than the hybrid strategy and should therefore only be used for studying properties of the Kernel SHAP algorithm.
  - 1: Uses all 2p on-off vectors z with sum(z) equal to 1 or p-1 for the exact part, which covers at least 75% of the mass of the Kernel weight distribution. The remaining mass is covered by sampling.
  - 2: Uses all p(p+1) on-off vectors z with sum(z) equal to 1, p-1, 2, or p-2. This covers at least 92% of the mass of the Kernel weight distribution. The remaining mass is covered by sampling. Convergence usually happens in the minimal possible number of iterations of two.
  - k>2: Uses all on-off vectors with sum(z) in 1,...,k and p-1,...,p-k.

paired_sampling
: Logical flag indicating whether to do the sampling in a paired manner. This means that with every on-off vector z, also 1-z is considered. CL21 shows its superiority compared to standard sampling, therefore the default (TRUE) should usually not be changed except for studying properties of Kernel SHAP algorithms. Ignored if exact = TRUE.

m
: Even number of on-off vectors sampled during one iteration. The default is 2p, except when hybrid_degree == 0. Then it is set to 8p. Ignored if exact = TRUE.

tol
: Tolerance determining when to stop. The algorithm keeps iterating until max(sigma_n)/diff(range(beta_n) < tol, where the beta_n are the SHAP values of a given observation and sigma_n their standard errors, see CL21. For multidimensional predictions, the criterion must be satisfied for each dimension separately. The stopping criterion uses the fact that standard errors and SHAP values are all on the same scale. Ignored if exact = TRUE.

max_iter
: If the stopping criterion (see tol) is not reached after max_iter iterations, the algorithm stops. Ignored if exact = TRUE.

parallel
: If TRUE, use parallel foreach::foreach() to loop over rows to be explained. Must register backend beforehand, e.g. via "doFuture" package, see Readme for an example. Parallelization automatically disables the progress bar.

parallel_args
: A named list of arguments passed to foreach::foreach(), see ?foreach::foreach. Ideally, this is NULL (default). Only relevant if parallel = TRUE. Example on Windows: if object is a generalized additive model fitted with package "mgcv", then one might need to set parallel_args = list(.packages = "mgcv").

verbose
: Set to FALSE to suppress messages and the progress bar.

Details

The iterative Kernel SHAP sampling algorithm (CL21) works by randomly sample m on-off vectors z so that their sum follows the SHAP Kernel weight distribution (renormalized to the range from 1 to p-1). Based on these vectors, many predictions are formed. Then, Kernel SHAP values are derived as the solution of a constrained linear regression. This is done multiple times until convergence, see CL21 for details.

A drawback of this strategy is that many (at least 75%) of the z vectors will have sum(z) equal to 1 or p-1, producing many duplicates. Similarly, at least 92% of the mass will be used for the p(p+1) possible vectors with sum(z) in 1, 2, p-1, p-2. This inefficiency can be fixed by a hybrid strategy, combining exact calculations with sampling.

The hybrid algorithm has two steps:

1. Step 1 (exact part): There are 2p different on-off vectors z with sum(z) equals to 1 or p-1, covering a large proportion of the Kernel SHAP distribution. The degree 1 hybrid will list those vectors and use them according to their weights in the upcoming calculations. Depending on p, we can also go a step further to a degree 2 hybrid by adding all p(p-1) vectors with sum(z) equals to 2 or p-2 to the process etc. The necessary predictions are obtained along with other calculations similar to those described in CL21.
2. Step 2 (sampling part): The remaining weight is filled by sampling vectors z according to Kernel SHAP weights renormalized to the values not yet covered by Step 1. Together with the results from Step 1 - correctly weighted - this now forms a complete iteration as in CL21. The difference is that most mass is covered by exact calculations. Afterwards, the algorithm iterates until convergence. The output of Step 1 is reused in every iteration, leading to an extremely efficient strategy.

If p is sufficiently small, all possible 2^p-2 on-off vectors z can be evaluated. In this case, no sampling is required and the algorithm returns exact Kernel SHAP values with respect to the given background data. Since `kernelshap()` calculates predictions on data with MN rows (N is the background data size and M the number of z vectors), p should not be much higher than 10 for exact calculations. For similar reasons, degree 2 hybrids are limited to p up to 30-40.

Value

An object of class "kernelshap" with the following components:

- `S`: (n x p) matrix with SHAP values or, if the model output has dimension K > 1, a list of K such matrices.
- `X`: Same as input argument `X`.
- `baseline`: A vector of length K representing the average prediction on the background data.
- `SE`: Standard errors corresponding to `S` (and organized like `S`).
- `n_iter`: Integer vector of length n providing the number of iterations per row of `X`.
- `converged`: Logical vector of length n indicating convergence per row of `X`.
- `m`: Integer providing the effective number of sampled on-off vectors used per iteration.
- `m_exact`: Integer providing the effective number of exact on-off vectors used per iteration.
- `prop_exact`: Proportion of the Kernel SHAP weight distribution covered by exact calculations.

- `exact`: Logical flag indicating whether calculations are exact or not.
- `txt`: Summary text.

Methods (by class)

- `kernelshap(default)`: Default Kernel SHAP method.
- `kernelshap(ranger)`: Kernel SHAP method for "ranger" models, see Readme for an example.
- `kernelshap(Learner)`: Kernel SHAP method for "mlr3" models, see Readme for an example.

References

1. Ian Covert and Su-In Lee. Improving KernelSHAP: Practical Shapley Value Estimation Using Linear Regression. Proceedings of The 24th International Conference on Artificial Intelligence and Statistics, PMLR 130:3457-3465, 2021.

Examples

```
# MODEL ONE: Linear regression
fit <- stats::lm(Sepal.Length ~ ., data = iris)

# Select rows to explain (only feature columns)
X_explain <- iris[1:2, -1]

# Select small background dataset (could use all rows here because iris is small)
set.seed(1)
bg_X <- iris[sample(nrow(iris), 100), ]

# Calculate SHAP values
s <- kernelshap(fit, X_explain, bg_X = bg_X)
s

# MODEL TWO: Multi-response linear regression
fit <- stats::lm(
  as.matrix(iris[1:2]) ~ Petal.Length + Petal.Width + Species, data = iris
)
s <- kernelshap(fit, iris[1:4, 3:5], bg_X = bg_X)
summary(s)

# Non-feature columns can be dropped via 'feature_names'
s <- kernelshap(
  fit,
  iris[1:4, ],
  bg_X = bg_X,
  feature_names = c("Petal.Length", "Petal.Width", "Species")
)
s
```

ks_extract *Extractor Function*

Description

Function to extract an element of a "kernelshap" object, e.g., the SHAP values "S".

Usage

```
ks_extract(object, ...)

## S3 method for class 'kernelshap'
ks_extract(
  object,
  what = c("S", "X", "baseline", "SE", "n_iter", "converged", "m", "m_exact",
    "prop_exact", "exact", "txt"),
  ...
)

## Default S3 method:
ks_extract(object, ...)
```

Arguments

object	Object to extract something.
...	Currently unused.
what	Element to extract. One of "S", "X", "baseline", "SE", "n_iter", "converged", "m", "m_exact", "prop_exact", "exact", or "txt".

Value

The corresponding object is returned.

Methods (by class)

- ks_extract(kernelshap): Method for "kernelshap" object.
- ks_extract(default): No default method available.

Examples

```
fit <- stats::lm(Sepal.Length ~ ., data = iris)
s <- kernelshap(fit, iris[1:2, -1], bg_X = iris[-1])
ks_extract(s, what = "S")
```

print.kernelshap *Print Method*

Description

Prints the first two rows of the matrix (or matrices) of SHAP values.

Usage

```
## S3 method for class 'kernelshap'
print(x, n = 2L, ...)
```

Arguments

x	An object of class "kernelshap".
n	Maximum number of rows of SHAP values to print.
...	Further arguments passed from other methods.

Value

Invisibly, the input is returned.

See Also

kernelshap.

Examples

```
fit <- stats::lm(Sepal.Length ~ ., data = iris)
s <- kernelshap(fit, iris[1:3, -1], bg_X = iris[-1])
s
```

summary.kernelshap *Summary Method*

Description

Summary Method

Usage

```
## S3 method for class 'kernelshap'
summary(object, compact = FALSE, n = 2L, ...)
```

Arguments

object	An object of class "kernelshap".
compact	Set to TRUE to hide printing the top n SHAP values, standard errors and feature values.
n	Maximum number of rows of SHAP values, standard errors and feature values to print.
...	Further arguments passed from other methods.

Value

Invisibly, the input is returned.

See Also

kernelshap.

Examples

```
fit <- stats::lm(Sepal.Length ~ ., data = iris)
s <- kernelshap(fit, iris[1:3, -1], bg_X = iris[-1])
summary(s)
```

Index