Package 'kernelshap'

September 20, 2025

Title Kernel SHAP

Version 0.9.1

Description Efficient implementation of Kernel SHAP (Lundberg and Lee, 2017, <doi:10.48550/arXiv.1705.07874>) permutation SHAP, and additive SHAP for model interpretability. For Kernel SHAP and permutation SHAP, if the number of features is too large for exact calculations, the algorithms iterate until the SHAP values are sufficiently precise in terms of their standard errors. The package integrates smoothly with meta-learning packages such as 'tidymodels', 'caret' or 'mlr3'. It supports multi-output models, case weights, and parallel computations. Visualizations can be done using the R package 'shapviz'.

License GPL (>= 2)

Depends R (>= 3.2.0)

Encoding UTF-8

RoxygenNote 7.3.2

Imports doFuture, foreach, stats, utils

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

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

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

NeedsCompilation no

Author Michael Mayer [aut, cre] (ORCID: <https://orcid.org/0009-0007-2540-9629>), David Watson [aut] (ORCID: <https://orcid.org/0000-0001-9632-2159>), Przemyslaw Biecek [ctb] (ORCID: <https://orcid.org/0000-0001-8423-1823>)

Maintainer Michael Mayer <mayermichael79@gmail.com>

Repository CRAN

Date/Publication 2025-09-20 14:20:02 UTC

Contents

`additive_shap` *Additive SHAP*

Description

Exact additive SHAP assuming feature independence. The implementation works for models fitted via

- `lm()`,
- `glm()`,
- `mgcv::gam()`,
- `mgcv::bam()`,
- `gam::gam()`,
- `survival::coxph()`, and
- `survival::survreg()`.

Usage

```
additive_shap(object, X, verbose = TRUE, ...)
```

Arguments

`object`	Fitted additive model.
`X`	Dataframe with rows to be explained. Passed to `predict(object, newdata = X, type = "terms")`.
`verbose`	Set to `FALSE` to suppress messages.
`...`	Currently unused.

Details

The SHAP values are extracted via `predict(object, newdata = X, type = "terms")`, a logic adopted from `fastshap:::explain.lm(..., exact = TRUE)`. Models with interactions (specified via `:` or `*`), or with terms of multiple features like `log(x1/x2)` are not supported.

Note that the SHAP values obtained by `additive_shap()` are expected to match those of `permshap()` and `kernelshap()` as long as their background data equals the full training data (which is typically not feasible).

Value

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

- `S`: $(n \times p)$ matrix with SHAP values.
- `X`: Same as input argument `X`.
- `baseline`: The baseline.
- `exact`: `TRUE`.
- `txt`: Summary text.
- `predictions`: Vector with predictions of `X` on the scale of "terms".
- `algorithm`: "additive_shap".

Examples

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

# MODEL TWO: More complicated (but not very clever) formula
fit <- lm(
  Sepal.Length ~ poly(Sepal.Width, 2) + log(Petal.Length) + log(Sepal.Width),
  data = iris
)
s_add <- additive_shap(fit, head(iris))
s_add

# Equals kernelshap()/permshap() when background data is full training data
s_kernel <- kernelshap(
 fit, head(iris[c("Sepal.Width", "Petal.Length")]), bg_X = iris
)
all.equal(s_add$S, s_kernel$S)
```

`is.kernelshap` *Check for kernelshap*

Description

Is object of class "kernelshap"?

Usage

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

Arguments

`object`	An R object.

Value

TRUE if object is of class "kernelshap", and FALSE otherwise.

See Also

kernelshap()

Examples

```
fit <- 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

Efficient implementation of Kernel SHAP, see Lundberg and Lee (2017), and Covert and Lee (2021), abbreviated by CL21. By default, for up to p=8 features, exact SHAP values are returned (with respect to the selected background data). Otherwise, a partly exact hybrid algorithm combining exact calculations and iterative paired sampling is used, see Details.

Usage

```
kernelshap(object, ...)

## Default S3 method:
kernelshap(
  object,
  X,
  bg_X = NULL,
  pred_fun = stats::predict,
  feature_names = colnames(X),
  bg_w = NULL,
  bg_n = 200L,
  exact = length(feature_names) <= 8L,
  hybrid_degree = 1L + length(feature_names) %in% 4:16,
  m = 2L * length(feature_names) * (1L + 3L * (hybrid_degree == 0L)),
  tol = 0.005,
  max_iter = 100L,
  parallel = FALSE,
  parallel_args = NULL,
  verbose = TRUE,
  seed = NULL,
  ...
```

```
)

## S3 method for class 'ranger'
kernelshap(
  object,
  X,
  bg_X = NULL,
  pred_fun = NULL,
  feature_names = colnames(X),
  bg_w = NULL,
  bg_n = 200L,
  exact = length(feature_names) <= 8L,
  hybrid_degree = 1L + length(feature_names) %in% 4:16,
  m = 2L * length(feature_names) * (1L + 3L * (hybrid_degree == 0L)),
  tol = 0.005,
  max_iter = 100L,
  parallel = FALSE,
  parallel_args = NULL,
  verbose = TRUE,
  seed = NULL,
  survival = c("chf", "prob"),
  ...
)
```

Arguments

object	Fitted model object.
...	Additional arguments passed to pred_fun(object, X, ...).
X	$(n \times p)$ matrix or data.frame with rows to be explained. The columns should only represent model features, not the response (but see feature_names on how to overrule this).
bg_X	Background data used to integrate out "switched off" features, often a subset of the training data (typically 50 to 500 rows). 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. If no bg_X is passed (the default) and if X is sufficiently large, a random sample of bg_n rows from X serves as background data.
pred_fun	Prediction function of the form function(object, X, ...), providing $K \geq 1$ 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).
bg_w	Optional vector of case weights for each row of bg_X. If bg_X = NULL, must be of same length as X. Set to NULL for no weights.
bg_n	If bg_X = NULL: Size of background data to be sampled from X.
exact	If TRUE, the algorithm will produce exact SHAP values with respect to the background data. The default is TRUE for up to eight features, and FALSE otherwise.

`hybrid_degree` Integer controlling the exactness of the hybrid strategy. For $4 \leq p \leq 16$, 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 $\sum z \in \{1, p-1\}$ for the exact part. The remaining mass is covered by random sampling.
- `2`: Uses all $p(p+1)$ on-off vectors z with $\sum z \in \{1, 2, p-2, p-1\}$. The remaining mass is covered by sampling. Usually converges fast.
- `k>2`: Uses all on-off vectors with $\sum z \in \{1, \dots, k, p-k, \dots, p-1\}$.

`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. As in CL21, the algorithm keeps iterating until $\max(\sigma_n)/(\max(\beta_n) - \min(\beta_n)) < \text{tol}$, where the β_n are the SHAP values of a given observation, and σ_n their standard errors. 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`. For `permshap()`, the default is 0.01, while for `kernelshap()` it is set to 0.005.

`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 `foreach::foreach()` and `%dofuture%` to loop over rows to be explained. Must register backend beforehand, e.g., `plan(multisession)`, see README for an example. Currently disables the progress bar.

`parallel_args` Named list of arguments passed to `foreach::foreach(.options.future = ...)`, ideally `NULL` (default). Only relevant if `parallel = TRUE`. Example on Windows: if `object` is a GAM fitted with package 'mgcv', then one might need to set `parallel_args = list(packages = "mgcv")`. Similarly, if the model has been fitted with `ranger()`, then it might be necessary to pass `parallel_args = list(packages = "ranger")`.

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

`seed` Optional integer random seed. Note that it changes the global seed.

`survival` Should cumulative hazards ("chf", default) or survival probabilities ("prob") per time be predicted? Only in `ranger()` survival models.

Details

The pure iterative Kernel SHAP sampling as in Covert and Lee (2021) works like this:

1. A binary "on-off" vector z is drawn from $\{0, 1\}^p$ according to a special weighting logic.
2. For each j with $z_j = 1$, the j-th column of the original background data is replaced by the corresponding feature value x_j of the observation to be explained.
3. The average prediction v_z on the data of Step 2 is calculated, and the average prediction v_0 on the background data is subtracted.

4. Steps 1 to 3 are repeated m times. This produces a binary $m \times p$ matrix Z (each row equals one of the z) and a vector v of shifted predictions.
5. v is regressed onto Z under the constraint that the sum of the coefficients equals $v_1 - v_0$, where v_1 is the prediction of the observation to be explained. The resulting coefficients are the Kernel SHAP values.

This is repeated multiple times until convergence, see CL21 for details.

To avoid the re-evaluation of identical coalition vectors, we have implemented 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 $\sum z \in \{1, p-1\}$. 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 $\sum z \in \{2, p-2\}$ 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 normalized 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 a significant part of the mass is covered by exact calculations. Afterwards, the algorithm iterates until convergence. The output of Step 1 is reused in every iteration.

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 higher than 10 for exact calculations. For similar reasons, degree 2 hybrids should not use p larger than 40.

Value

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

- `S`: $(n \times p)$ 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`: Vector of length K representing the average prediction on the background data.
- `bg_X`: The background data.
- `bg_w`: The background case weights.
- `m_exact`: Number of on-off vectors evaluated for exact calculations.
- `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.
- `predictions`: $(n \times K)$ matrix with predictions of `X`.
- `algorithm`: "kernelshap".

- `m`: Number of sampled on-off vectors evaluated per iteration (if not exact).
- `SE`: Standard errors corresponding to `S` (if not exact).
- `n_iter`: Integer vector of length n providing the number of iterations per row of `X` (if not exact).
- `converged`: Logical vector of length n indicating convergence per row of `X` (if not exact).

Methods (by class)

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

References

1. Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017.
2. 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 <- lm(Sepal.Length ~ ., data = iris)

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

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

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

# Note 1: Feature columns can also be selected 'feature_names'
# Note 2: Especially when X is small, pass a sufficiently large background data bg_X
s <- kernelshap(
  fit,
  iris[1:4, ],
  bg_X = iris,
  feature_names = c("Petal.Length", "Petal.Width", "Species")
)
s
```

permshap *Permutation SHAP*

Description

Permutation SHAP algorithm with respect to a background dataset, see Strumbelj and Kononenko (2014) for the basic idea.

By default, for up to p=8 features, exact SHAP values are returned (exact with respect to the selected background data). Otherwise, the sampling process iterates until the resulting values are sufficiently precise, and standard errors are provided.

Usage

```
permshap(object, ...)

## Default S3 method:
permshap(
  object,
  X,
  bg_X = NULL,
  pred_fun = stats::predict,
  feature_names = colnames(X),
  bg_w = NULL,
  bg_n = 200L,
  exact = length(feature_names) <= 8L,
  low_memory = length(feature_names) > 15L,
  tol = 0.01,
  max_iter = 10L * length(feature_names),
  parallel = FALSE,
  parallel_args = NULL,
  verbose = TRUE,
  seed = NULL,
  ...
)

## S3 method for class 'ranger'
permshap(
  object,
  X,
  bg_X = NULL,
  pred_fun = NULL,
  feature_names = colnames(X),
  bg_w = NULL,
  bg_n = 200L,
  exact = length(feature_names) <= 8L,
  low_memory = length(feature_names) > 15L,
```

```
  tol = 0.01,
  max_iter = 10L * length(feature_names),
  parallel = FALSE,
  parallel_args = NULL,
  verbose = TRUE,
  seed = NULL,
  survival = c("chf", "prob"),
  ...
)
```

Arguments

object	Fitted model object.
...	Additional arguments passed to pred_fun(object, X, ...).
X	$(n \times p)$ matrix or data.frame with rows to be explained. The columns should only represent model features, not the response (but see feature_names on how to overrule this).
bg_X	Background data used to integrate out "switched off" features, often a subset of the training data (typically 50 to 500 rows). 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. If no bg_X is passed (the default) and if X is sufficiently large, a random sample of bg_n rows from X serves as background data.
pred_fun	Prediction function of the form function(object, X, ...), providing $K \geq 1$ 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).
bg_w	Optional vector of case weights for each row of bg_X. If bg_X = NULL, must be of same length as X. Set to NULL for no weights.
bg_n	If bg_X = NULL: Size of background data to be sampled from X.
exact	If TRUE, the algorithm will produce exact SHAP values with respect to the background data. The default is TRUE for up to eight features, and FALSE otherwise.
low_memory	If FALSE (default up to p = 15), the algorithm does p iterations in one chunk, evaluating Shapley's formula 2p^2 times. For models with interactions up to order two, you can set this to TRUE to save time.
tol	Tolerance determining when to stop. As in CL21, the algorithm keeps iterating until $\max(\sigma_n)/(\max(\beta_n) - \min(\beta_n)) <$ tol, where the β_n are the SHAP values of a given observation, and σ_n their standard errors. 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. For permshap(), the default is 0.01, while for kernelshap() it is set to 0.005.
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 foreach::foreach() and %dofuture% to loop over rows to be explained. Must register backend beforehand, e.g., plan(multisession), see README for an example. Currently disables the progress bar.
parallel_args	Named list of arguments passed to foreach::foreach(.options.future = ...), ideally NULL (default). Only relevant if parallel = TRUE. Example on Windows: if object is a GAM fitted with package 'mgcv', then one might need to set parallel_args = list(packages = "mgcv"). Similarly, if the model has been fitted with ranger(), then it might be necessary to pass parallel_args = list(packages = "ranger").
verbose	Set to FALSE to suppress messages and the progress bar.
seed	Optional integer random seed. Note that it changes the global seed.
survival	Should cumulative hazards ("chf", default) or survival probabilities ("prob") per time be predicted? Only in ranger() survival models.

Details

During each iteration, the algorithm cycles twice through a random permutation: It starts with all feature components "turned on" (i.e., taking them from the observation to be explained), then gradually turning off components according to the permutation. When all components are turned off, the algorithm - one by one - turns the components back on, until all components are turned on again. This antithetic scheme allows to evaluate Shapley's formula twice per feature using a single permutation and a total of 2p disjoint evaluations of the contribution function.

For models with interactions up to order two, one can show that even a single iteration provides exact SHAP values for all features (with respect to the given background dataset).

The Python implementation "shap" uses a similar approach, but without providing standard errors, and without early stopping.

For faster convergence, we use balanced permutations in the sense that p subsequent permutations each start with a different feature. Furthermore, the 2p on-off vectors with sum <=1 or >=p-1 are evaluated only once, similar to the degree 1 hybrid in kernelshap().

Value

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

- S: $(n \times p)$ 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: Vector of length K representing the average prediction on the background data.
- bg_X: The background data.
- bg_w: The background case weights.
- m_exact: Number of on-off vectors evaluated once per row of X.
- exact: Logical flag indicating whether calculations are exact or not.
- txt: Summary text.
- predictions: $(n \times K)$ matrix with predictions of X.

- `algorithm`: "permshap".
- `m`: Number of sampled on-off vectors evaluated per iteration (if not exact).
- `SE`: Standard errors corresponding to `S` (if not exact).
- `n_iter`: Integer vector of length n providing the number of iterations per row of `X` (if not exact).
- `converged`: Logical vector of length n indicating convergence per row of `X` (if not exact).

Methods (by class)

- `permshap(default)`: Default permutation SHAP method.
- `permshap(ranger)`: Permutation SHAP method for "ranger" models, see Readme for an example.

References

1. Erik Strumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. Knowledge and Information Systems 41, 2014.

Examples

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

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

# Calculate SHAP values
s <- permshap(fit, X_explain)
s

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

# Note 1: Feature columns can also be selected 'feature_names'
# Note 2: Especially when X is small, pass a sufficiently large background data bg_X
s <- permshap(
  fit,
  iris[1:4, ],
  bg_X = iris,
  feature_names = c("Petal.Length", "Petal.Width", "Species")
)
s
```

print.kernelshap *Prints "kernelshap" Object*

Description

Prints "kernelshap" Object

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

summary.kernelshap *Summarizes "kernelshap" Object*

Description

Summarizes "kernelshap" Object

Usage

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

Arguments

`object`	An object of class "kernelshap".
`compact`	Set to `TRUE` for a more compact summary.
`n`	Maximum number of rows of SHAP values etc. to print.
`...`	Further arguments passed from other methods.

Value

Invisibly, the input is returned.

See Also

`kernelshap()`

Examples

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

Index