Package 'kernelshap'

August 17, 2024

Title Kernel SHAP

Version 0.7.0

Description Efficient implementation of Kernel SHAP, see Lundberg and Lee (2017), and Covert and Lee (2021) <http://proceedings.mlr.press/v130/covert21a>. Furthermore, for up to 14 features, exact permutation SHAP values can be calculated. 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 7.3.2

Imports foreach, MASS, stats, utils

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

Maintainer Michael Mayer <mayermichael79@gmail.com>

Repository CRAN

Date/Publication 2024-08-17 16:00:02 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 model object.
`X`	Dataframe with rows to be explained. Will be used like `predict(object, newdata = X, type = "terms")`.
`verbose`	Set to `FALSE` to suppress messages and the progress bar.
`...`	Currently unused.

Details

The SHAP values are extracted via `predict(object, newdata = X, type = "terms")`, a logic heavily inspired by `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. For up to $p = 8$ features, the resulting Kernel SHAP values are exact regarding the selected background data. For larger p, an almost exact hybrid algorithm involving iterative sampling is used, see Details. For up to eight features, however, we recommend to use permshap().

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,
  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 = 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,
  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,
  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)`. Not supported if `X` is a matrix.
`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 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 $4 \leq p \leq 16$, 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, which covers at least 75% of the mass of the Kernel weight distribution. The remaining mass is covered by random sampling.
- `2`: Uses all $p(p+1)$ on-off vectors z with $\sum z \in \{1, 2, p-2, p-1\}$. 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 $\sum z \in \{1, \dots, k, p-k, \dots, p-1\}$.

`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. Following 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`.

`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` Named list of arguments passed to `foreach::foreach()`. Ideally, this is `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")`.

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

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

Details

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

1. A binary "on-off" vector z is drawn from $\{0,1\}^p$ such that its sum follows the SHAP Kernel weight distribution (normalized to the range $\{1,\dots,p-1\}$).
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.

A drawback of this strategy is that many (at least 75%) of the z vectors will have $\sum z \in \{1, p-1\}$, producing many duplicates. Similarly, at least 92% of the mass will be used for the $p(p+1)$ possible vectors with $\sum z \in \{1, 2, p-2, p-1\}$. 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 $\sum z \in \{1, p-1\}$, 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 $\sum z \in \{2, p-2\}$ 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 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 should not use p much 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.
- 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.
- predictions: $(n \times K)$ matrix with predictions of X.
- algorithm: "kernelshap".

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

Exact permutation SHAP algorithm with respect to a background dataset, see Strumbelj and Kononenko. The function works for up to 14 features. For eight or more features, we recomment to switch to `kernelshap()`.

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,
  parallel = FALSE,
  parallel_args = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'ranger'
permshap(
  object,
  X,
  bg_X = NULL,
  pred_fun = NULL,
  feature_names = colnames(X),
  bg_w = NULL,
  bg_n = 200L,
```

```
  parallel = FALSE,
  parallel_args = NULL,
  verbose = TRUE,
  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). Not supported if X is a matrix.
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.
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	Named list of arguments passed to foreach::foreach(). Ideally, this is 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").
verbose	Set to FALSE to suppress messages and the progress bar.
survival	Should cumulative hazards ("chf", default) or survival probabilities ("prob") per time be predicted? Only in ranger() survival models.

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`: Integer providing the effective number of exact on-off vectors used.
- `exact`: Logical flag indicating whether calculations are exact or not (currently `TRUE`).
- `txt`: Summary text.
- `predictions`: $(n \times K)$ matrix with predictions of `X`.
- `algorithm`: "permshap".

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