Package: gKRLS (via r-universe)

December 8, 2024

Type Package

Title Generalized Kernel Regularized Least Squares

Version 1.0.4

Date 2024-11-07

Encoding UTF-8

License GPL (>= 2)

Description Kernel regularized least squares, also known as kernel ridge regression, is a flexible machine learning method. This package implements this method by providing a smooth term for use with 'mgcv' and uses random sketching to facilitate scalable estimation on large datasets. It provides additional functions for calculating marginal effects after estimation and for use with ensembles ('SuperLearning'), double/debiased machine learning ('DoubleML'), and robust/clustered standard errors ('sandwich'). Chang and Goplerud (2024) <doi:10.1017/pan.2023.27> provide further details.

LinkingTo Rcpp, RcppEigen

Imports Rcpp (>= 1.0.6), Matrix, mlr3, R6

Depends mgcv, sandwich (>= 2.4.0)

Suggests SuperLearner, mlr3misc, DoubleML, testthat

SystemRequirements GNU make

RoxygenNote 7.3.2

NeedsCompilation yes

URL https://github.com/mgoplerud/gKRLS

BugReports https://github.com/mgoplerud/gKRLS/issues

Author Qing Chang [aut], Max Goplerud [aut, cre]

Maintainer Max Goplerud <mgoplerud@austin.utexas.edu>

Repository CRAN

Date/Publication 2024-11-07 22:30:02 UTC

Config/pak/sysreqs make

Contents

`calculate_effects` *Marginal Effects*

Description

These functions calculate marginal effects or predicted values after estimating a model with `gam` or `bam`.

Usage

```
calculate_effects(
  model,
  data = NULL,
  variables = NULL,
  continuous_type = c("IQR", "minmax", "derivative", "onesd", "predict",
    "second_derivative"),
  conditional = NULL,
  individual = FALSE,
  vcov = NULL,
  raw = FALSE,
  use_original = FALSE,
  epsilon = 1e-07,
  verbose = FALSE
)

calculate_interactions(
  model,
  variables,
  QOI = c("AMIE", "ACE", "AME", "AIE"),
  ...
)

get_individual_effects(x)

## S3 method for class 'gKRLS_mfx'
print(x, ...)

## S3 method for class 'gKRLS_mfx'
summary(object, ...)
```

Arguments

model	A model estimated using functions from mgcv (e.g., gam or bam).
data	A data frame that is used to calculate the marginal effect or set to NULL which will employ the data used when estimating the model. The default is NULL. Using a custom dataset may have unexpected implications for continuous and character/factor variables. See "WARNINGS" for more discussion.
variables	A character vector that specifies the variables for which to calculate effects. The default, NULL, calculates effects for all variables.
continuous_type	A character value, with a default of "IQR", that indicates the type of marginal effects to estimate when the variable is continuous (i.e. not binary, logical, factor, or character). Options are "IQR" (compares the variable at its 25% and 75% percentile), "minmax" (compares the variable at its minimum and maximum), "derivative" (numerically approximates the derivative at each observed value), "second_derivative" (numerically approximates the second derivative at each observed value), "onesd" (compares one standard deviation below and one standard deviation above the mean of the variable). It also accepts a named list where each named element corresponds to a continuous variable and has a two-length vector as each element. The two values are then compared. If this is used, then all continuous variables must have two values specified. A special option ("predict") produces predictions (e.g., predict(model, type = "response")) at each observed value and then averages them together. This, in conjunction with conditional, provides a way of calculating quantities such as predicted probability curves using an "observed value" approach (e.g., Hanmer and Kalkan 2013). Examples are provided below.
conditional	A data.frame or NULL. This is an analogue of Stata's at() option and the at argument in the margins package. For a marginal effect on some variable "a", this specifies fixed values for certain other covariates, e.g. data.frame("b" = 0). If conditional is NULL, all other covariates are held at their observed value. If conditional is a data.frame, then each row represents a different combination of covariate values to be held fixed, and marginal effects are calculated separately for each row. Examples are provided below.
individual	A logical value. TRUE calculates individual effects (i.e. an effect for each observation in the data). The default is FALSE.
vcov	A matrix that specifies the covariance matrix of the parameters. The default, NULL, uses the standard covariance matrix from mgcv. This can be used to specify clustered or robust standard errors using output from (for example) sandwich.
raw	A logical value. TRUE returns the raw values used to calculate the effect in addition to the estimated effect. The default is FALSE. If TRUE, an additional column ...id is present in the estimated effects that reports whether the row corresponds to the effect (effect), the first value (raw_0) or the second value (raw_1) where effect=raw_1 - raw_0. For "derivative", this is further scaled by the step size. For "second_derivative", effect=raw_2 - 2 * raw_1 + raw_0, scaled by the step size; see the discussion for epsilon for how the step size is calculated.

use_original — A logical value that indicates whether to use the estimation data (TRUE) or data (FALSE) when calculating quantities such as the IQR for continuous variables or the levels to examine for factor variables. Default (FALSE) uses the provided data; if data = NULL, this is equivalent to using the estimation data. The "WARNINGS" section provides more discussion of this option.

epsilon — A numerical value that defines the step size when calculating numerical derivatives (default of 1e-7). For "derivative", the step size for the approximation is $h = \epsilon \cdot \max(1, \max(|x|))$, i.e. $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$. Please see Leeper (2016) for more details.

For "second_derivative", the step size is $h = [\epsilon \cdot \max(1, \max(|x|))]^{0.5}$, i.e. $f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$

verbose — A logical value that indicates whether to report progress when calculating the marginal effects. The default is FALSE.

QOI — A vector of quantities of interest calculate for calculate_interactions. Options include "AME" (average marginal effect), "ACE" (average combination effect), "AIE" (average interaction effect) and "AMIE" (average marginal interaction effect); see "Details" for more information. The default setting calculates all four quantities.

... — An argument used for calculate_interactions to pass arguments to calculate_effects. It is unused for summary.gKRLS_mfx.

x — An object estimated using calculate_effects.

object — A model estimated using functions from mgcv (e.g., gam or bam).

Details

Overview: calculate_effects returns a data.frame of class "gKRLS_mfx" that reports the estimated average marginal effects and standard errors. Other columns include "type" that reports the type of marginal effect calculated. For families with multiple predicted outcomes (e.g., multinomial), the column "response" numbers the different outcomes in the same order as predict.gam(object) for the specified family. Many (but not all) extended and generalized families from mgcv are included.

The conditional argument while setting continuous_type = "predict" can be used to estimate predicted values at different covariate strata (e.g., to create an "observed value" predicted probability curve for a logistic regression). The examples provide an illustration.

Interactions: calculate_interactions provides some simple functions for calculating interaction effects between variables. The default quantities it can produce are listed below. Egami and Imai (2019) provide a detailed exposition of these quantities. All marginalization is done using an "observed value" approach, i.e. over the estimation data or a custom dataset provided to data.

- "AME" or Average Marginal Effect: This is the standard quantity reported from calculate_effects.
- "ACE" or Average Combination Effect: This is the effect of changing two variables simultaneously on the outcome.
- "AMIE" or Average Marginal Interaction Effect: This is ACE minus each corresponding AME.

- "AIE" or Average Interaction Effect: This has a "conditional effect" interpretation and reports the difference in average effect of one variable ("A") between two different levels of a second variable ("B").

Other Functions: `get_individual_effects` extracts the individual-level effects that are estimated if `individual=TRUE`.

Value

Both `calculate_effects` and `calculate_interactions` return data.frames. `calculate_effects` contains attributes—including the ones noted below—that may be useful for other analyses.

- "gradient": This contains the gradients used to calculate the standard error (via the delta method) for the estimates from `calculate_effects`. There is one column for each quantity calculated in the main object. The format of this object depends on the family used for `gam` or `bam`. This could be used manually to calculate a standard error on the difference between two estimated marginal effects.
- "N_eff": The number of observations (in the estimation data) minus the effective degrees of freedom. This is used when calculating p-values as the degrees of freedom for the t-distribution.
- "N": The number of observations.

WARNINGS

Using a custom dataset for `data`, i.e. a dataset other than the estimation data, may have unexpected implications. For continuous and character/factor variables, the estimated marginal effects may depend on the distribution of the variable in `data`. For example, if `continuous_type="IQR"`, the variable `x1` is counterfactually set to `quantile(data$x1, 0.25)` and `quantile(data$x1, 0.75)` where `data` is provided by `calculate_effects` (versus the estimation data). To force this range to be set based on the *estimation* data, set `use_original=TRUE`.

This default behavior if `data` is provided may be undesirable and thus `calculate_effects` will issue a warning if this situation arises and a custom `data` is provided. These settings are subject to change in future releases.

References

Egami, Naoki, and Kosuke Imai. 2019. "Causal Interaction in Factorial Experiments: Application to Conjoint Analysis." *Journal of the American Statistical Association*. 114(526):529-540.

Hanmer, Michael J., and Kerem Ozan Kalkan. 2013. "Behind the Curve: Clarifying the Best Approach to Calculating Predicted Probabilities and Marginal Effects from Limited Dependent Variable Models." *American Journal of Political Science* 57(1): 263-277.

Leeper, Thomas J. 2016. "Interpreting Regression Results using Average Marginal Effects with R's `margins`." Working paper available at https://s3.us-east-2.amazonaws.com/tjl-sharing/assets/AverageMarginalEffects.pdf.

Examples

```
set.seed(654)
n <- 50
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
state <- sample(letters[1:5], n, replace = TRUE)
y <- 0.3 * x1 + 0.4 * x2 + 0.5 * x3 + rnorm(n)
data <- data.frame(y, x1, x2, x3, state)

# Make character variables into factors for mgcv
data$state <- factor(data$state)

# A gKRLS model
fit_gKRLS <- mgcv::gam(y ~ state + s(x1, x2, x3, bs = "gKRLS"), data = data)

# calculate marginal effect using derivative
calculate_effects(fit_gKRLS, variables = "x1", continuous_type = "derivative")

# calculate marginal effect by specifying conditional variables
calculate_effects(fit_gKRLS,
  variables = "x1",
  conditional = data.frame(x2 = c(0.6, 0.8), x3 = 0.3)
)

# calculate interaction effects between two variables
# use the default setting ("IQR") for the baseline and
# comparison categories for each variable
calculate_interactions(fit_gKRLS,
   variables = list(c("x1", "x2")),
   QOI = c('AIE', 'AMIE')
)

# calculate marginal effect by specifying a factor conditional variable
# estimate the individual marginal effects
out <- calculate_effects(fit_gKRLS,
  variables = "x1", individual = TRUE,
  conditional = data.frame(state = c("a", "b", "c")), continuous_type = "derivative"
)

# Extract the individual marginal effects:
# shorthand for attr(fit_main, 'individual')
get_individual_effects(out)

# calculated the average expected value across a grid of "x1"
# using an observed value approach for the other covariates
calculate_effects(fit_gKRLS, conditional = data.frame(x1 = c(0, 0.2, 0.4, 0.6)),
  continuous_type = 'predict'
)
```

gKRLS *Generalized Kernel Regularized Least Squares*

Description

This page documents how to use gKRLS as part of a model estimated with mgcv. Post-estimation functions to calculate marginal effects are documented elsewhere, e.g. calculate_effects.

Usage

```
gKRLS(
  sketch_method = "subsampling",
  standardize = "Mahalanobis",
  bandwidth = NULL,
  sketch_multiplier = 5,
  sketch_size_raw = NULL,
  sketch_prob = NULL,
  rescale_penalty = TRUE,
  truncate.eigen.tol = sqrt(.Machine$double.eps),
  demean_kernel = FALSE,
  remove_instability = TRUE
)

get_calibration_information(object)
```

Arguments

sketch_method	A string that specifies which kernel sketching method should be used (default of "subsampling"). Options include "subsampling", "gaussian", "bernoulli", or "none" (no sketching). Drineas et al. (2005) and Yang et al. (2017) provide more details on these options. To force "subsampling" to select a specific set of observations, you can provide a vector of row positions to sketch_method. This manually sets the size of the sketching multiplier, implicitly overriding other options in gKRLS. The examples provide an illustration.
standardize	A string that specifies how the data is standardized before calculating the distance between observations. The default is "Mahalanobis" (i.e., demeaned and transformed to have an identity covariance matrix). Other options are "scaled" (all columns are scaled to have mean zero and variance of one) or "none" (no standardization).
bandwidth	A bandwidth P for the kernel where each element of the kernel (i, j) is defined by $\exp(-\|\|x_i - x_j\|\|_2^2/P)$. The default (NULL) uses the number of covariates in the kernel or the rank of the corresponding design matrix. An additional option ("calibrate") choses P to maximize the variance of the kernel, e.g., $var(vec(K))$ for the unsketched case. This follows Hartman et al. (2024) with modifications when the kernel is sketched. Please see gKRLS_addendum.pdf for a formal exposition.

`sketch_multiplier`
: A number that sets the size of the sketching dimension: `sketch_multiplier * ceiling(N^(1/3))` where `N` is the number of observations. The default is 5; Chang and Goplerud (2024) find that increasing this to 15 may improve stability for certain complex kernels. `sketch_size_raw` can directly set the size of the sketching dimension.

`sketch_size_raw`
: A number to set the exact size of the sketching dimension. The default, `NULL`, means that this argument is not used and the size depends on the number of observations; see `sketch_multiplier`. Exactly one of `sketch_size_raw` or `sketch_multiplier` must be `NULL`.

`sketch_prob`
: A probability for an element of the sketching matrix to equal 1 when using Bernoulli sketching. Yang et al. (2017) provide more details.

`rescale_penalty`
: A logical value for whether the penalty should be rescaled for numerical stability. See documentation for `mgcv::smooth.spec` on the meaning of this term. The default is `TRUE`.

`truncate.eigen.tol`
: A threshold to remove columns of the penalty SKS^T whose eigenvalues are small (below `truncate.eigen.tol`). These columns are removed from the sketched kernel and avoids instability due to numerically very small eigenvalues. The default is `sqrt(.Machine$double.eps)`. This adjustment can be disabled by setting `remove_instability = FALSE`.

`demean_kernel`
: A logical value that indicates whether columns of the (sketched) kernel should be demeaned before estimation. The default is `FALSE`.

`remove_instability`
: A logical value that indicates whether numerical zeros (set via `truncate.eigen.tol`) should be removed when building the penalty matrix. The default is `TRUE`.

`object`
: Model estimated using `mgcv::gam` or `mgcv::bam`

Details

Overview: The `gKRLS` function should not be called directly. Its options, described above, control how `gKRLS` is estimated. It should be passed to `mgcv` as follows: `s(x1, x2, x3, bs = "gKRLS", xt = gKRLS(...))`. Multiple kernels can be specified and have different `gKRLS` arguments. It can also be used alongside the existing options for `s()` in `mgcv`.

If `bandwidth="calibrate"`, the function `get_calibration_information` reports the estimated bandwidth and time (in minutes) needed to do so.

Default Settings: By default, `bs = "gKRLS"` uses Mahalanobis distance between the observations, random sketching using subsampling sketching (i.e., where the kernel is constructed using a random sample of the observations; Yang et al. 2017) and a sketching dimension of `5 * ceiling(N^(1/3))` where `N` is the number of observations. Chang and Goplerud (2024) provide an exploration of alternative options.

Notes: Please note that variables must be separated with commas inside of `s(...)` and that character variables should usually be passed as factors to work smoothly with `mgcv`. When using this function with `bam`, the sketching dimension uses `chunk.size` in place of `N` and thus either `chunk.size` or `sketch_size_raw` must be used to cause the sketching dimension to increase with `N`.

Value

gKRLS returns a named list with the elements in "Arguments".

References

Chang, Qing, and Max Goplerud. 2024. "Generalized Kernel Regularized Least Squares." *Political Analysis* 32(2):157-171.

Hartman, Erin, Chad Hazlett, and Ciara Sterbenz. 2024. "kpop: A Kernel Balancing Approach for Reducing Specification Assumptions in Survey Weighting." *Journal of the Royal Statistical Society Series A: Statistics in Society* doi:10.1093/jrsssa/qnae082.

Drineas, Petros, Michael W. Mahoney, and Nello Cristianini. 2005. "On the Nyström Method for Approximating a Gram Matrix For Improved Kernel-Based Learning." *Journal of Machine Learning Research* 6(12):2153-2175.

Yang, Yun, Mert Pilanci, and Martin J. Wainwright. 2017. "Randomized Sketches for Kernels: Fast and Optimal Nonparametric Regression." *Annals of Statistics* 45(3):991-1023.

Examples

```
set.seed(123)
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
state <- sample(letters[1:5], n, replace = TRUE)
y <- 0.3 * x1 + 0.4 * x2 + 0.5 * x3 + rnorm(n)
data <- data.frame(y, x1, x2, x3, state)
data$state <- factor(data$state)
# A gKRLS model without fixed effects
fit_gKRLS <- mgcv::gam(y ~ s(x1, x2, x3, bs = "gKRLS"), data = data)
summary(fit_gKRLS)
# A gKRLS model with fixed effects outside of the kernel
fit_gKRLS_FE <- mgcv::gam(y ~ state + s(x1, x2, x3, bs = "gKRLS"), data = data)

# HC3 is not available for mgcv; this uses the effective degrees of freedom
# instead of the number of columns; see ?estfun.gam for details
robust <- sandwich::vcovHC(fit_gKRLS, type = 'HC1')
cluster <- sandwich::vcovCL(fit_gKRLS, cluster = data$state)

# Change default standardization to "scaled", sketch method to Gaussian,
# and alter sketching multiplier
fit_gKRLS_alt <- mgcv::gam(y ~ s(x1, x2, x3,
  bs = "gKRLS",
  xt = gKRLS(
    standardize = "scaled",
    sketch_method = "gaussian",
    sketch_multiplier = 2
  )
),
data = data
)
```

```
# A model with multiple kernels
fit_gKRLS_2 <- mgcv::gam(y ~ s(x1, x2, bs = 'gKRLS') + s(x1, x3, bs = 'gKRLS'), data = data)
# A model with a custom set of ids for sketching
id <- sample(1:n, 5)
fit_gKRLS_custom <- mgcv::gam(y ~ s(x1, bs = 'gKRLS', xt = gKRLS(sketch_method = id)), data = data)
# Note that the ids of the sampled observations can be extracted
# from the fitted mgcv object
stopifnot(identical(id, fit_gKRLS_custom$smooth[[1]]$subsampling_id))
# calculate marginal effect (see ?calculate_effects for more examples)
calculate_effects(fit_gKRLS, variables = "x1")
```

`ml_gKRLS` *Machine Learning with gKRLS*

Description

This provides a number of functions to use `gKRLS` (and `mgcv` more generally) as part of machine learning algorithms. Integration into `SuperLearner` and `DoubleML` (and `mlr3`) is described below.

Usage

```
SL.mgcv(Y, X, newX, formula, family, obsWeights, bam = FALSE, ...)

## S3 method for class 'SL.mgcv'
predict(object, newdata, allow_missing_levels = TRUE, ...)

add_bam_to_mlr3()
```

Arguments

`Y`	This is not usually directly specified in `SL.mgcv`, see the examples below and documentation in `SuperLearner` for more details.
`X`	This is not usually directly specified in `SL.mgcv`, see the examples below and documentation in `SuperLearner` for more details.
`newX`	This is not usually directly specified in `SL.mgcv`, see the examples below and documentation in `SuperLearner` for more details.
`formula`	A formula used for `gam` or `bam` from `mgcv`. This must be specified, see the examples.
`family`	This is not usually directly specified in `SL.mgcv`, see the examples below and documentation in `SuperLearner` for more details.
`obsWeights`	This is not usually directly specified in `SL.mgcv`, see the examples below and documentation in `SuperLearner` for more details.
`bam`	A logical value for whether `mgcv::bam` should be used instead of `mgcv::gam`. Default is `FALSE`. For large datasets, this can dramatically improve estimation time. Wood et al. (2015) and `mgcv` provide details on `bam`.
`...`	Additional arguments to `mgcv::gam` and `mgcv::bam`.

`object` This is not usually directly specified in `SL.mgcv`, see the examples below and documentation in `SuperLearner` for more details.

`newdata` This is not usually directly specified in `SL.mgcv`, see the examples below and documentation in `SuperLearner` for more details.

`allow_missing_levels`
A logical variable that indicates whether missing levels in factors are allowed for prediction. The default is `TRUE`.

Details

Ensembles: `SuperLearner` integration is provided by `SL.mgcv` and the corresponding predict method. `mgcv::bam` can be enabled by using `bam = TRUE`. A formula **without an outcome** must be explicitly provided.

Please note that it is often useful to load `SuperLearner` before `gKRLS` or `mgcv` to avoid functions including `gam` and `s` being masked from other packages.

Double Machine Learning: `DoubleML` integration is provided in two ways. First, one could load `mlr3extralearners` to access `regr.gam` and `classif.gam`.

Second, this package provides `mgcv::bam` integration directly with a slight adaption of the `mlr3extralearner` implementation (see `?LearnerClassifBam` for more details). These can be either manually added to the list of `mlr3` learners by calling `add_bam_to_mlr3()` or direct usage. Examples are provided below. For `classif.bam` and `regr.bam`, the formula argument is mandatory.

Value

All three of the returned functions are usually called for use in other functions, i.e. creating objects for use in `SuperLearner` or adding `bam` models to `mlr3`.

References

Wood, Simon N., Yannig Goude, and Simon Shaw. 2015. "Generalized Additive Models for Large Data Sets." *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 64(1):139-155.

Examples

```
set.seed(789)
N <- 100
x1 <- rnorm(N)
x2 <- rbinom(N, size = 1, prob = .2)
y <- x1^3 - 0.5 * x2 + rnorm(N, 0, 1)
y <- y * 10
X <- cbind(x1, x2, x1 + x2 * 3)
X <- cbind(X, "x3" = rexp(nrow(X)))

if (requireNamespace("SuperLearner", quietly = TRUE)) {
# Estimate Ensemble with SuperLearner
  require(SuperLearner)
  sl_m <- function(...) { SL.mgcv(formula = ~ x1 + x2 + x3, ...) }
  fit_SL <- SuperLearner::SuperLearner(
    Y = y, X = data.frame(X),
```

```
    SL.library = "sl_m"
  )
  pred <- predict(fit_SL, newdata = data.frame(X))
}
# Estimate Double/Debiased Machine Learning
if (requireNamespace("DoubleML", quietly = TRUE)) {
  require(DoubleML)
  # Load the models; for testing *ONLY* have multiplier of 2
  double_bam_1 <- LearnerRegrBam$new()
  double_bam_1$param_set$values$formula <- ~ s(x1, x3, bs = "gKRLS",
    xt = gKRLS(sketch_multiplier = NULL, sketch_size_raw = 2))
  double_bam_2 <- LearnerClassifBam$new()
  double_bam_2$param_set$values$formula <- ~ s(x1, x3, bs = "gKRLS",
    xt = gKRLS(sketch_multiplier = NULL, sketch_size_raw = 2))

  # Create data
  dml_data <- DoubleMLData$new(
    data = data.frame(X, y),
    x_cols = c("x1", "x3"), y_col = "y",
    d_cols = "x2"
  )
  # Estimate effects treatment (works for other DoubleML methods)
  dml_est <- DoubleMLIRM$new(
    data = dml_data,
    n_folds = 2,
    ml_g = double_bam_1,
    ml_m = double_bam_2
  )$fit()
}
```

Index