Package 'FastKRR'

July 21, 2026

Type Package

Title Kernel Ridge Regression using 'RcppArmadillo'

Version 0.2.1

Description Provides core computational operations in C++ via 'RcppArmadillo', enabling faster performance than pure R, improved numerical stability, and parallel execution with OpenMP where available. On systems without OpenMP support, the package automatically falls back to single-threaded execution with no user configuration required. For efficient model selection, it integrates with 'CVST' to provide sequential-testing cross-validation and additionally supports restricted maximum likelihood (REML) for continuous optimization of the regularization parameter. The package offers a unified interface for exact kernel ridge regression and three scalable approximations—Nyström, Pivoted Cholesky, and Random Fourier Features—allowing analyses with substantially larger sample sizes than are feasible with exact KRR. It also integrates with the 'tidymodels' ecosystem via the 'parsnip' model specification 'krr_reg', and the S3 method tunable.krr_reg(). To understand the theoretical background, one can refer to Wainwright (2019) <doi:10.1017/9781108627771>.

License GPL (>= 2)

URL https://github.com/kybak90/FastKRR, https://www.tidymodels.org

BugReports https://github.com/kybak90/FastKRR/issues

Imports CVST, generics, parsnip, Rcpp, rlang, tibble, ggplot2

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown, dials, tidymodels, modeldata, dplyr, testthat (>= 3.0.0)

SystemRequirements OpenMP (optional)

Encoding UTF-8

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation yes

Author Gyeongmin Kim [aut] (Sungshin Women's University), Seyoung Lee [aut] (Sungshin Women's University),

Miyoung Jang [aut] (Sungshin Women's University),
Kwan-Young Bak [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-4541-160X>, Sungshin Women's
University)

Maintainer Kwan-Young Bak <kybak@sungshin.ac.kr>

Repository CRAN

Date/Publication 2026-07-21 09:30:18 UTC

Contents

`FastKRR-package` *Kernel Ridge Regression using the* **RcppArmadillo** *Package*

Description

The **FastKRR** implements its core computational operations in C++ via **RcppArmadillo**, enabling faster performance than pure R, improved numerical stability, and parallel execution with OpenMP where available. On systems without OpenMP support, the package automatically falls back to single-threaded execution with no user configuration required. For efficient model selection, it integrates with **CVST** to provide full and sequential-testing cross-validation and additionally supports restricted maximum likelihood (REML) for continuous optimization of the regularization parameter. The package offers a unified interface for exact kernel ridge regression and three widely used scalable approximations—Nyström, Pivoted Cholesky, and Random Fourier Features—allowing analyses with substantially larger sample sizes than are feasible with exact KRR while retaining strong predictive performance. This combination of a compiled backend and scalable algorithms addresses limitations of packages that rely solely on exact computation, which is often impractical for large n. It also integrates with the **tidymodels** ecosystem via the **parsnip** model specification `krr_reg`, and the S3 method `tunable.krr_reg()` (exposes tunable parameters to `dials`/`tune`); see their help pages for usage.

Directory structure

- `R/`: High-level R functions and user-facing API
- `src/`: C++ sources (kernel computation, fitting, prediction)

This package links against **Rcpp** and **RcppArmadillo** (via LinkingTo). It uses **CVST**, **parsnip**, and the **tidymodels** ecosystem through their public R APIs.

Author(s)

Maintainer: Kwan-Young Bak <kybak@sungshin.ac.kr> (ORCID) (Sungshin Women's University) [copyright holder]

Authors:

- Gyeongmin Kim <rlarudals0824@gmail.com> (Sungshin Women's University)
- Seyoung Lee <sudang0404@gmail.com> (Sungshin Women's University)
- Miyoung Jang <miyoung9072@gmail.com> (Sungshin Women's University)

See Also

CVST, **Rcpp**, **RcppArmadillo**, **parsnip**, **tidymodels**

`approx_kernel` *Compute low-rank approximations (Nyström, Pivoted Cholesky, RFF)*

Description

Computes low-rank kernel approximation $\tilde{K} \in \mathbb{R}^{n \times n}$ using three methods: Nyström approximation, Pivoted Cholesky decomposition, and Random Fourier Features (RFF).

Usage

```
approx_kernel(
  X = NULL,
  opt = c("nystrom", "pivoted", "rff"),
  kernel = c("gaussian", "laplace"),
  m = NULL,
  rho,
  eps = NULL,
  W = NULL,
  b = NULL,
  n_threads = NULL
)
```

Arguments

X	A numeric design matrix $X \in \mathbb{R}^{n \times d}$.
opt	Method for constructing or approximating: "nystrom" Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using the Nyström approximation. "pivoted" Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using Pivoted Cholesky decomposition. "rff" Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using Random Fourier Features (RFF).
kernel	Kernel type either "gaussian" or "laplace".
m	Approximation rank (number of random features) for the low-rank kernel approximation. If not specified, the recommended choice is $\lceil n^{1/2} \cdot \log(d+5) \rceil$ where X is design matrix, $n = nrow(X)$ and $d = ncol(X)$.
rho	Scaling parameter of the kernel (ρ), specified by the user.
eps	Tolerance parameter used only in "pivoted" for stopping criterion of the Pivoted Cholesky decomposition. If NULL, it is dynamically adjusted based on the smoothness of the chosen kernel: defaults to 1e-6 for the infinitely smooth Gaussian kernel to ensure precision, and 1e-4 for the less smooth Laplace kernel.
W	Random frequency matrix $\omega \in \mathbb{R}^{m \times d}$
b	Random phase vector $b \in \mathbb{R}^m$, i.i.d. $\mathrm{Unif}[0, 2\pi]$.
n_threads	Number of parallel threads. If NULL, defaults to half of the available system processors. It automatically falls back to 1 thread if the system has 3 or fewer processors. Applied only for opt = "nystrom", opt = "rff", or kernel = "laplace".

Details

Requirements and what to supply:

Common

- rho must be provided (non-NULL).

nystrom / pivoted

- If m is NULL, use $\lceil n^{1/2} \cdot \log(d+5) \rceil$.
- For "pivoted", a tolerance eps is used; the decomposition stops early when the next pivot (residual diagonal) drops below eps.

rff

- The function automatically generates W (random frequency matrix $\omega \in \mathbb{R}^{m \times d}$) and b (random phase vector $b \in \mathbb{R}^m$).
- If the user provides them manually, both W and b must be specified and their dimensions must be compatible.

Value

- `call`: The matched function call used to create the object.
- `opt`: The kernel approximation method actually used (`"nystrom"`, `"pivoted"`, `"rff"`).
- `K_approx`: The fully reconstructed $n \times n$ approximated kernel matrix.
- `approx_factor`: Low-rank component matrix (R for Nyström, PR for Pivoted Cholesky, Z for RFF).
- `m`: Kernel approximation degree.
- `rho`: Scaling parameter of the kernel.

Additional components depend on the value of opt:

nystrom

- `n_threads`: Number of threads used in the computation.

pivoted

- `eps`: Numerical tolerance used for early stopping in the pivoted Cholesky decomposition.

rff

- `d`: Input design matrix's dimension.
- `W`: $m \times d$ Random frequency matrix.
- `b`: Random phase m-vector.
- `used_supplied_Wb`: Logical; `TRUE` if user-supplied `W`, `b` were used, `FALSE` otherwise.
- `n_threads`: Number of threads used in the computation.

Examples

```
# Data setting
set.seed(1)
d = 1
rho = 1
n = 100
m = 50
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)

# Example: Nystrom approximation
K_nystrom = approx_kernel(X = X, opt = "nystrom",
                          m = m, rho = rho, n_threads = 1)

# Example: Pivoted Cholesky approximation
K_pivoted = approx_kernel(X = X, opt = "pivoted",
                          m = m, rho = rho)
# Example: RFF approximation
K_rff = approx_kernel(X = X, opt = "rff", kernel = "gaussian",
                      m = m, rho = rho, n_threads = 1)
```

coef.krr *Extract Model Coefficients from a Fitted KRR Model*

Description

Extracts the estimated coefficients from a fitted Kernel Ridge Regression (KRR) model. The type of coefficient reported depends on the kernel approximation method: for `opt = "exact"`, `"nystrom"`, or `"pivoted"`, the coefficients represent the dual weights α. For `opt = "rff"`, they represent the coefficients β.

Usage

```
## S3 method for class 'krr'
coef(object, ...)
```

Arguments

`object`	An S3 object of class `krr`, typically returned by `fastkrr`.
`...`	Additional arguments (currently ignored).

Value

A numeric vector of the estimated model coefficients (α or β).

See Also

`fastkrr`

Examples

```
# Data setting
set.seed(1)
lambda = 1e-4
d = 1
n = 50
rho = 1
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = sin(2 * pi * rowMeans(X)^3) + rnorm(n, mean = 0, sd = 0.1)

data = data.frame(X, y = y)

# Example: exact
model = fastkrr(data = data, response = "y",
                kernel = "gaussian", opt = "exact",
                rho = rho, lambda = lambda)

coef(model)
```

`error.krr` *Compute Model Error for Kernel Ridge Regression Models*

Description

Computes the model error for kernel ridge regression (`"krr"` object). Returns the mean squared error (MSE) between the observed responses and the fitted values stored in the object.

Usage

```
error(x, ...)

## S3 method for class 'krr'
error(x, data_new = NULL, ...)
```

Arguments

`x`	An object of class `"krr"`, typically returned by `fastkrr`.
`...`	Additional arguments (ignored).
`data_new`	An optional data frame containing new predictor variables and the observed response variable. The response column must have the same name as the response variable used to fit the model. If `NULL`, the training MSE is computed. Otherwise, the prediction MSE (PMSE) is computed using predictions for `data_new`.

Details

This method computes the mean squared error defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}(y_i - \hat{y}_i)^2$$

Depending on the presence of `data_new`, the components are defined as follows:

- If `data_new = NULL`, y is the observed response vector and $\hat{y}$ is the fitted values vector, both stored within the `"krr"` object.
- If `data_new` is provided, y is the observed response extracted from `data_new` and $\hat{y}$ is the predicted values vector generated by applying `predict.krr` to the new predictor variables.

Value

A numeric value giving the mean squared error (MSE). If `data_new = NULL`, this returns the training MSE based on the fitted values. If `data_new` is provided, it returns the prediction mean squared error (PMSE) evaluated on the new dataset.

See Also

`summary.krr`, `plot.krr`, `predict.krr`

Examples

```
# Data setting
set.seed(1)
lambda = 1e-4
d = 1
n = 50
rho = 1
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = sin(2 * pi * rowMeans(X)^3) + rnorm(n, mean = 0, sd = 0.1)

data = data.frame(X, y = y)

model = fastkrr(data = data, response = "y", kernel = "gaussian",
                opt = "exact", lambda = lambda)

# MSE
error(model)

new_n = 50
new_x = matrix(runif(new_n*d, 0, 1), nrow = new_n, ncol = d)
new_y = as.vector(sin(2*pi*rowMeans(new_x)^3) + rnorm(new_n, 0, 0.1))
new_data = data.frame(new_x, y = new_y)

# PMSE
error(model, data_new = new_data)
```

`fastkrr` *Fit kernel ridge regression using exact or approximate methods*

Description

This function performs kernel ridge regression (KRR). The regularization parameter λ can be supplied by the user or selected automatically using cross-validation or restricted maximum likelihood (REML). For scalability, three different kernel approximation strategies are supported (Nyström approximation, Pivoted Cholesky decomposition, Random Fourier Features(RFF)), and kernel matrix can be computed using two methods (Gaussian kernel, Laplace kernel).

Usage

```
fastkrr(
  data,
  response,
  kernel = "gaussian",
  opt = "exact",
  m = NULL,
  eps = NULL,
  rho = 1,
```

```
  lambda = NULL,
  selection_method = "exactCV",
  n_threads = NULL,
  verbose = TRUE,
  na.rm = FALSE
)
```

Arguments

`data`	A data frame containing the data point variables and response variable.
`response`	A character string specifying the name of the response variable in `data`.
`kernel`	Kernel type either "gaussian"or "laplace".
`opt`	Method for constructing or approximating : `"exact"` Construct the full kernel matrix $K \in \mathbb{R}^{n\times n}$ using design matrix X. `"nystrom"` Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n\times n}$ using the Nyström approximation. `"pivoted"` Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n\times n}$ using Pivoted Cholesky decomposition. `"rff"` Use Random Fourier Features to construct a feature map $Z \in \mathbb{R}^{n\times m}$ (with m random features) so that $K \approx ZZ^\top$. Here, m is the number of features.
`m`	Approximation rank(number of random features) used for the low-rank kernel approximation. If not provided by the user, it defaults to $\lceil n^{1/2} \cdot \log(d+5) \rceil$, where $n = nrow(X)$ and $d = ncol(X)$. Also it must be a positive integer.
`eps`	Tolerance parameter used only in `"pivoted"` for the stopping criterion of the Pivoted Cholesky decomposition. The default value is dynamically adjusted based on the mathematical smoothness of the selected kernel: it defaults to `1e-6` for the infinitely differentiable and smooth Gaussian kernel, and relaxes to `1e-4` for the non-smooth, rougher Laplace kernel to properly balance numerical precision and computational overhead.
`rho`	Scaling parameter of the kernel(ρ), specified by the user. Defaults to 1. $$\text{Gaussian kernel : } \mathcal{K}(x,x') = \exp(-\rho\|x-x'\|_2^2)$$ $$\text{Laplace kernel : } \mathcal{K}(x,x') = \exp(-\rho\|x-x'\|_1)$$
`lambda`	Regularization parameter. If `NULL`, the penalty parameter is chosen automatically via **CVST** package or REML: for `selection_method = "REML"` it defaults to the search range $[10^{-11}, 10^{-1}]$ (a length-2 min/max vector), and for `selection_method %in% c("exactCV", "fastCV")` it defaults to a grid of 100 values over $[10^{-11}, 10^{-1}]$.
`selection_method`	Method used to select λ when a grid or `NULL` is passed. One of: `"exactCV"` Full cross-validation via **CVST** (default). `"fastCV"` Accelerated sequential-testing CV via **CVST**. `"REML"` Restricted Maximum Likelihood.

n_threads	Number of parallel threads. If NULL, it defaults to half of the available system processors (max_threads %/% 2). Note that parallelization (implemented in C++) is applied for opt = "nystrom", opt = "rff", kernel = "laplace", or selection_method = "REML". For these parallelizable cases, if the system has 3 or fewer processors, it automatically falls back to 1 thread; otherwise, it is capped at max_threads - 1. For all other settings, it is restricted to 1 thread.
verbose	If TRUE, detailed progress and cross-validation results are printed to the console. If FALSE, suppresses intermediate output and only returns the final result.
na.rm	Logical. If TRUE, rows containing missing values are removed before fitting. Defaults to FALSE.

Details

The function performs several input checks and automatic adjustments:

- lambda can be specified in four ways:
  1. A positive numeric scalar, in which case the model is fitted with this single value and no selection is performed (any selection_method).
  2. A numeric vector of length 2 giving c(min, max); only valid when selection_method = "REML", which optimizes λ within this range.
  3. A numeric vector (length >= 3) of positive values used as a tuning grid; only valid when selection_method %in% c("exactCV", "fastCV"), and selection is performed by **CVST** cross-validation (sequential testing if selection_method = "fastCV").
  4. NULL: use a default range/grid (internal setting) and tune lambda via **CVST** or REML, depending on selection_method.

After fitting the model with fastkrr(), users can readily generate predictions for new test data or out-of-sample observations using the standard generic predict function, which is internally dispatched to predict.krr.

Value

- coefficients: Estimated coefficient vector. Accessible via model$coefficients. For opt = "rff", this is an m-dimensional weight vector in the random feature space. For all other options ("exact", "pivoted", "nystrom"), this is an n-dimensional dual coefficient vector.
- fitted.values: Fitted values $\mathbb{R}^n$. Accessible via model$fitted.values. Computed as:
  - $\hat{y} = K\hat{\alpha}$ for opt = "exact" (full kernel matrix K).
  - $\hat{y} = \tilde{K}\tilde{\alpha}$ for opt = "pivoted" or "nystrom" (low-rank kernel matrix $\tilde{K}$).
  - $\hat{y} = Z\hat{\beta}$ for opt = "rff" (random feature matrix Z).

  Here, $\hat{\alpha}$, $\tilde{\alpha}$, and $\hat{\beta}$ represent the estimated regression coefficient vectors (coefficients) obtained under each respective option opt.
- opt: Kernel approximation option. One of "exact", "pivoted", "nystrom", "rff".
- kernel: Kernel used ("gaussian" or "laplace").
- x: Input design matrix.
- y: Response vector.

- `lambda`: Regularization parameter. If `NULL`, tuned by cross-validation via **CVST** or REML.
- `rho`: Additional user-specified hyperparameter.
- `selection_method`: Tunning method for select hyperparmeter lambda
- `call`: The matched function call used to create the object.
- `n_threads`: Number of threads used for parallelization.
- `removed_row_idx`: Indices of rows removed due to missing values.

Additional components depend on the value of `opt`:

opt = "exact":

- `K`: The full kernel matrix $K \in \mathbb{R}^{n\times n}$.
- `chol_factor`: Lower triangular Cholesky factor $L \in \mathbb{R}^{n\times n}$ of $K+n\lambda I$; satisfies $K+n\lambda I = LL^\top$.

opt = "nystrom":

- `m`: Kernel approximation degree.
- `approx_factor`: The method provides a low-rank approximation to the kernel matrix $R \in \mathbb{R}^{n\times m}$ obtained via Nyström approximation; satisfies $K \approx RR^\top$.

opt = "pivoted":

- `m`: Kernel pproximation degree.
- `approx_factor`: The method provides a low-rank approximation to the kernel matrix $PR \in \mathbb{R}^{n\times m}$ obtained via Pivoted Cholesky decomposition; satisfies $K \approx PR\,(PR)^\top$.
- `eps`: Numerical tolerance used for early stopping in the pivoted Cholesky decomposition.

opt = "rff":

- `m`: Number of random features.
- `approx_factor`: Random Fourier Feature matrix $Z \in \mathbb{R}^{n\times m}$ with $Z_{ij} = z_j(x_i) = \sqrt{2/m}\cos(\omega_j^\top x_i + b_j), \quad j = 1, \cdots, m$, so that $K \approx ZZ^\top$.
- `W`: Random frequency matrix $\omega \in \mathbb{R}^{m\times d}$ (row j is $\omega_j^\top \in \mathbb{R}^d$), drawn i.i.d. from the spectral density of the chosen kernel:
  - Gaussian: $\omega_{jk} \sim \mathcal{N}(0, 2\gamma)$ (e.g., $\gamma = 1/\ell^2$).
  - Laplace: $\omega_{jk} \sim \mathrm{Cauchy}(0, 1/\sigma)$ i.i.d.
- `b` Random phase vector $b \in \mathbb{R}^m$, i.i.d. $\mathrm{Unif}[0,\, 2\pi]$.

See Also

`predict.krr`, `param.krr`, `make_kernel`

Examples

```
# Data setting
set.seed(1)
lambda = 1e-4
d = 3
rho = 1
n = 50
```

```
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = sin(2 * pi * rowMeans(X)^3) + rnorm(n, mean = 0, sd = 0.1)

data = data.frame(X, y = y)

# Example: pivoted cholesky
model = fastkrr(data = data, response = "y",  kernel = "gaussian",
                opt = "pivoted", rho = rho, lambda = lambda)

# Example: nystrom
model = fastkrr(data = data, response = "y", kernel = "gaussian",
                opt = "nystrom", rho = rho, lambda = lambda)

# Example: random fourier features
model = fastkrr(data = data, response = "y", kernel = "gaussian",
                opt = "rff", rho = rho, lambda = lambda)

# Example: Laplace kernel
model = fastkrr(data = data, response = "y",  kernel = "laplace",
                opt = "nystrom", n_threads = 1, rho = rho)

# Generate predictions for new data
new_X = matrix(runif(10 * d, 0, 1), nrow = 10, ncol = d)
pred = predict(model, new_X)
print(pred)
```

`krr_reg` *Kernel Ridge Regression*

Description

Defines a Kernel Ridge Regression model specification for use with the tidymodels ecosystem via **parsnip**. This spec can be paired with the `"fastkrr"` engine implemented in this package to fit exact or kernel approximation (Nyström, Pivoted Cholesky, Random Fourier Features) within **recipes/workflows** pipelines.

Usage

```
krr_reg(
  mode = "regression",
  kernel = NULL,
  opt = NULL,
  eps = NULL,
  n_threads = NULL,
  m = NULL,
  rho = NULL,
  penalty = NULL,
```

```
  na.rm = NULL
)
```

Arguments

mode	A single string; only ‘"regression"‘ is supported.
kernel	Kernel matrix has two kinds of Kernel ("gaussian", "laplace").
opt	Method for constructing or approximating : "exact" Construct the full kernel matrix $K \in \mathbb{R}^{n\times n}$ using design matrix X. "nystrom" Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n\times n}$ using the Nyström approximation. "pivoted" Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n\times n}$ using Pivoted Cholesky decomposition. "rff" Use Random Fourier Features to construct a feature map $Z \in \mathbb{R}^{n\times m}$ (with m random features) so that $K \approx ZZ^\top$. Here, m is the number of features.
eps	Tolerance parameter used only in "pivoted" for stopping criterion of the Pivoted Cholesky decomposition.
n_threads	Number of parallel threads. It is applied only for opt = "nystrom" or opt = "rff", and for the Laplace kernel (kernel = "laplace").
m	Approximation rank(number of random features) used for the low-rank kernel approximation.
rho	Scaling parameter of the kernel(ρ).
penalty	Regularization parameter. It must be a positive numeric value, a numeric vector containing at least three positive values, or tune(). When using the **tidymodels** interface for hyperparameter selection, setting penalty = tune() is recommended. Candidate values are then supplied through the grid argument of tune_grid() and evaluated using the resampling scheme specified by the user.
na.rm	Logical. If TRUE, rows containing missing values are removed before fitting. Defaults to FALSE.

Value

A parsnip model specification of class "krr_reg".

Examples

```
if (all(vapply(
  c("parsnip","stats","modeldata"),
  requireNamespace, quietly = TRUE, FUN.VALUE = logical(1)
))) {
library(tidymodels)
library(parsnip)
library(stats)
library(modeldata)

# Data analysis
```

```
data(ames)
ames = ames %>% mutate(Sale_Price = log10(Sale_Price))

set.seed(502)
ames_split = initial_split(ames, prop = 0.80, strata = Sale_Price)
ames_train = training(ames_split) # dim (2342, 74)
ames_test  = testing(ames_split) # dim (588, 74)

# Model spec
krr_spec = krr_reg(kernel = "gaussian", opt = "nystrom",
                   m = 50, eps = 1e-6, n_threads = 1,
                   rho = 1, penalty = tune()) %>%
 set_engine("fastkrr") %>%
 set_mode("regression")

# Define rec
rec = recipe(Sale_Price ~ Longitude + Latitude, data = ames_train)

# workflow
wf = workflow() %>%
  add_recipe(rec) %>%
  add_model(krr_spec)

# Define hyper-parameter grid
param_grid = grid_regular(
  dials::penalty(range = c(-10, -3)),
  levels = 5
)

# CV setting
set.seed(123)
cv_folds = vfold_cv(ames_train, v = 5, strata = Sale_Price)

# Tuning
tune_results = tune_grid(
  wf,
  resamples = cv_folds,
  grid = param_grid,
  metrics = metric_set(rmse),
  control = control_grid(verbose = TRUE, save_pred = TRUE)
)

# Result check
collect_metrics(tune_results)

# Select best parameter
best_params = select_best(tune_results, metric = "rmse")

# Finalized model spec using best parameter
final_spec = finalize_model(krr_spec, best_params)
final_wf = workflow() %>%
  add_recipe(rec) %>%
  add_model(final_spec)
```

```
# Finalized fitting using best parameter
final_fit = final_wf %>% fit(data = ames_train)

# Prediction
predict(final_fit, new_data = ames_test)
print(best_params)

}
```

`make_kernel` *Kernel matrix construction for given datasets*

Description

Constructs a kernel matrix $K \in \mathbb{R}^{n' \times n}$ given two datasets $X \in \mathbb{R}^{n \times d}$ and $X' \in \mathbb{R}^{n' \times d}$, where $x_i \in \mathbb{R}^d$ and $x'_j \in \mathbb{R}^d$ denote the i-th and j-th rows of X and X', respectively, and $K_{ji} = \mathcal{K}(x_i, x'_j)$ for a user-specified kernel. Implemented in C++ via RcppArmadillo.

Usage

```
make_kernel(X,
            X_new = NULL,
            kernel = "gaussian",
            rho = 1,
            n_threads = NULL)
```

Arguments

`X`	Design matrix $X \in \mathbb{R}^{n \times d}$ (rows $x_i \in \mathbb{R}^d$).
`X_new`	Second matrix $X' \in \mathbb{R}^{n' \times d}$ (rows $x'_j \in \mathbb{R}^d$). If omitted, $X' = X$ and $n' = n$.
`kernel`	Kernel type; one of `"gaussian"` or `"laplace"`.
`rho`	Kernel width parameter ($\rho > 0$). Default is 1.
`n_threads`	Number of parallel threads. If `NULL`, it defaults to half of the available system processors (`max_threads %/% 2`). For these parallelizable cases, if the system has 3 or fewer processors, it automatically falls back to 1 thread; otherwise, it is capped at `max_threads - 1`. For all other settings, it is restricted to 1 thread. Parallelization (implemented in C++) is one of the main advantages of this package and is applied only for `"laplace"` kernels.

Details

Gaussian kernel:

$$K_{ji} = \mathcal{K}(x_i, x_j) = \exp\big(-\rho \|x_i - x_j\|_2^2\big)$$

Laplace kernel:

$$K_{ji} = \mathcal{K}(x_i, x_j) = \exp\big(-\rho \|x_i - x_j\|_1\big)$$

Value

The computed kernel matrix. If `X_new` is `NULL`, the result is a symmetric matrix $K_{ij} = \mathcal{K}(x_i, x_j)$, with $K \in \mathbb{R}^{n \times n}$. Otherwise, the result is a rectangular matrix $K'_{ji} = \mathcal{K}(x_i, x'_j)$, with $K' \in \mathbb{R}^{n' \times n}$.

Examples

```
# Data setting
set.seed(1)
d = 1
rho = 1
n = 100
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)

# New design matrix
new_n = 150
new_X = matrix(runif(new_n*d, 0, 1), nrow = new_n, ncol = d)

# Make kernel : Gaussian kernel
K = make_kernel(X, kernel = "gaussian", rho = rho) ## symmetric matrix
new_K = make_kernel(X, new_X, kernel = "gaussian", rho = rho) ## rectangular matrix

# Make kernel : Laplace kernel
K = make_kernel(X, kernel = "laplace", rho = rho, n_threads = 1) ## symmetric matrix
new_K = make_kernel(X, new_X, kernel = "laplace", rho = rho, n_threads = 1) ## rectangular matrix
```

---

param.krr *Displays hyperparameters of fitted Kernel Ridge Regression models*

Description

Displays (and invisibly returns) the hyperparameters actually used by a fitted object. For `krr` objects returned by `fastkrr`, this prints a concise hyperparameter panel (e.g., `rho`, `lambda`, `m`, `eps`, `n_threads`, `d`).

Usage

```
## S3 method for class 'krr'
param(x, ...)
```

Arguments

`x`	An object of class `"krr"`, typically returned by `fastkrr`.
`...`	Additional arguments.

Details

Pivoted approximation note: When `opt = "pivoted"`, the effective number of pivots `m` used during the approximation may be smaller than the user-specified `m` because the algorithm can stop early based on `eps`. If you want to confirm the initial `m` that you set, please see the printed *Call* (the original function call shows your input arguments).

Value

Prints a human-readable panel to the console and invisibly returns a named list of class `"krr_params"` containing the extracted hyperparameters:

- `kernel`: Kernel type used ("gaussian" or "laplace").
- `opt`: Kernel approximation method.
- `selection_method`: Lambda tuning method.
- `rho`: Kernel scaling parameter.
- `lambda`: Regularization parameter.
- `m`: Rank or number of random features used for approximation.
- `eps`: Tolerance parameter (for pivoted Cholesky).
- `n_threads`: Number of parallel threads used.

See Also

`fastkrr`

Examples

```
# Data setting
set.seed(1)
lambda = 1e-4
d = 1
n = 50
rho = 1
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = sin(2 * pi * rowMeans(X)^3) + rnorm(n, mean = 0, sd = 0.1)

data = data.frame(X, y = y)

model = fastkrr(data = data, response = "y",
                kernel="gaussian", opt="pivoted",
                rho=1, lambda=lambda, n_threads = 1)

# Inspect hyperparameters
param(model)
```

plot.krr *Plot method for fitted Kernel Ridge Regression models*

Description

Visualizes the fitted regression curve from a Kernel Ridge Regression (KRR) model. Automatically generates predictions on a regular grid (1.2 times the training sample size) and overlays them with the training data.

Usage

```
## S3 method for class 'krr'
plot(x, show_points = TRUE, ...)
```

Arguments

x	A fitted KRR model (class "krr") returned by fastkrr.
show_points	Logical; if TRUE, displays the original training data points as a background layer. Default is TRUE.
...	Additional arguments (currently ignored).

Details

Currently, plot.krr supports only uni-variate inputs ($d = 1$). For multivariate settings ($d \geq 2$), the plot method will return an error, and users are encouraged to manually slice their data to visualize conditional main effects.

Value

A ggplot object showing the fitted regression line and training data.

See Also

fastkrr, predict.krr

Examples

```
set.seed(1)
n = 1000
rho = 1
d = 1
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d); colnames(X) = paste0("X", seq_len(d))
y = sin(2 * pi * rowMeans(X)^3) + rnorm(n, mean = 0, sd = 0.1)

data = data.frame(X, y = y)

model_exact = fastkrr(data = data, response = "y",
                      kernel = "gaussian", rho = rho, opt = "exact", verbose = FALSE)
```

```
plot(model_exact)
```

`predict.krr` *Predict responses for new data using fitted KRR model*

Description

Generates predictions from a fitted Kernel Ridge Regression (KRR) model for new data.

Usage

```
## S3 method for class 'krr'
predict(object, newdata, ...)
```

Arguments

`object`	A S3 object of class `krr` created by `fastkrr`.
`newdata`	A numeric design matrix containing new observations for which predictions are to be made. If `newdata` is missing, the function returns fitted values.
`...`	Additional arguments (currently ignored).

Value

A numeric vector of predicted values corresponding to `newdata` or fitted values.

See Also

`fastkrr`, `make_kernel`

Examples

```
# Data setting
n = 30
d = 1

X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = sin(2 * pi * rowMeans(X)^3) + rnorm(n, mean = 0, sd = 0.1)

data = data.frame(X, y = y)

lambda = 1e-4
rho = 1

# Fitting model: pivoted
model = fastkrr(data = data, response = "y",
                kernel = "gaussian", rho = rho, lambda = lambda, opt = "pivoted")
```

```
# Predict
new_n = 50
new_x = matrix(runif(new_n*d, 0, 1), nrow = new_n, ncol = d)
new_y = as.vector(sin(2*pi*rowMeans(new_x)^3) + rnorm(new_n, 0, 0.1))

pred = predict(model, new_x)
crossprod(pred - new_y) / new_n
```

`print.approx_kernel` *Print method for approximated kernel matrices*

Description

Displays the key algorithmic choices and hyperparameter options used to construct an approximated kernel matrix object.

Usage

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

Arguments

`x`	An S3 object created by `approx_kernel`.
`...`	Additional arguments (currently ignored).

Details

The function summarizes critical metadata attributes stored within the `approx_kernel` object, including the approximation strategy (`opt`), kernel type, approximation degree (`m`), numerical tolerance (`eps`), and the number of parallel computational threads used.

Value

Invisibly returns the input `approx_kernel` object after printing the options.

See Also

`approx_kernel`, `print.krr`

Examples

```
# Data setting
set.seed(1)
d = 1
n = 100
rho = 1
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
```

```
# Example: nystrom
K_nystrom = approx_kernel(X = X, opt = "nystrom", rho = rho, n_threads = 1)

print(K_nystrom)
```

`print.krr` *Print method for fitted Kernel Ridge Regression models*

Description

Displays a concise summary of key information from a fitted Kernel Ridge Regression (KRR) model, including the original function call and the main hyperparameter options used during fitting.

Usage

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

Arguments

`x`	An S3 object of class `krr`, typically returned by `fastkrr`.
`...`	Additional arguments (currently ignored).

Value

Invisibly returns the input `krr` object after printing the summary to the console.

See Also

`fastkrr`, `print.approx_kernel`

Examples

```
# Data setting
set.seed(1)
lambda = 1e-4
d = 1
n = 50
rho = 1
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = sin(2 * pi * rowMeans(X)^3) + rnorm(n, mean = 0, sd = 0.1)

data = data.frame(X, y = y)

# Example: exact
model = fastkrr(data = data, response = "y",
                kernel = "gaussian", opt = "exact",
                rho = rho, lambda = lambda)

print(model)
```

summary.krr *Summary method for fitted Kernel Ridge Regression models*

Description

Computes and displays a comprehensive summary of a fitted Kernel Ridge Regression (KRR) model, including the original function call, fitted hyperparameters (via param.krr), and the final training mean squared error (via error.krr).

Usage

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

Arguments

object	An S3 object of class krr, typically returned by fastkrr.
...	Additional arguments (currently ignored).

Value

Invisibly returns the hyperparameter list or model statistics after printing the summary panel to the console.

See Also

fastkrr, param.krr, error.krr

Examples

```
# Data setting
set.seed(1)
lambda = 1e-4
d = 1
n = 50
rho = 1
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = sin(2 * pi * rowMeans(X)^3) + rnorm(n, mean = 0, sd = 0.1)

data = data.frame(X, y = y)

# Example: exact
model = fastkrr(data = data, response = "y",
                kernel = "gaussian", opt = "exact",
                rho = rho, lambda = lambda)

summary(model)
```

tunable.krr_reg *Expose tunable parameters for* "krr_reg"

Description

Supplies a tibble of tunable arguments for "krr_reg()".

Usage

```
## S3 method for class 'krr_reg'
tunable(x, ...)
```

Arguments

x	A "krr_reg" model specification.
...	Not used; included for S3 method compatibility.

Value

A tibble (one row per tunable parameter) with columns "name", "call_info", "source", "component", and "component_id".

Index