Package ‘gplite’

October 13, 2022

Title General Purpose Gaussian Process Modelling

Version 0.13.0

Description Implements the most common Gaussian process (GP) models using Laplace and expectation propagation (EP) approximations, maximum marginal likelihood (or posterior) inference for the hyperparameters, and sparse approximations for larger datasets.

Depends R (>= 3.4.0),

Imports Matrix, methods, Rcpp

LinkingTo Rcpp, RcppArmadillo

License GPL-3

Encoding UTF-8

Biarch TRUE

RoxygenNote 7.2.0

Suggests testthat, knitr, rmarkdown, ggplot2

VignetteBuilder knitr

NeedsCompilation yes

Author Juho Piironen [cre, aut]

Maintainer Juho Piironen <juho.t.piironen@gmail.com>

Repository CRAN

Date/Publication 2022-08-24 07:40:02 UTC

R topics documented:

`gplite-package` *The 'gplite' package.*

Description

gplite implements some of the most common Gaussian process (GP) models. The package offers tools for integrating out the latent values analytically using Laplace or expectation propagation (EP) approximation and for estimating the hyperparameters based on maximizing the (approximate) marginal likelihood or posterior. The package also implements some common sparse approximations for larger datasets.

Functions

Here's a list of the most important functions:

gp_init Set up the GP model.

cf, **lik**, **method**, **approx** Choose the covariance functions, likelihood (observation model), type of the GP (full or some sparse approximation) and the latent function approximation method (Laplace, EP).

gp_optim, **gp_fit** Optimize the model hyperparameters, or just fit the model with the current hyperparameter values.

gp_pred, **gp_draw** Make predictions with the fitted model. Can also be used before fitting to obtain prior predictive distribution or draws.

gp_loo, **gp_compare** Model assessment and comparison using leave-one-out (LOO) cross-validation.

`approx` *Approximations to the posterior of the latent values*

Description

Functions for initializing the approximation for the latent values, which can then be passed to `gp_init`. The supported methods are:

`approx_laplace` Laplace's method, that is, based on local second order approximation to the log likelihood. For Gaussian likelihood, this means exact inference (no approximation).

`approx_ep` Expectation propagation, EP. Approximates the likelihood by introducing Gaussian pseudo-data so that the posterior marginals match to the so called tilted distributions (leave-one-out posterior times the true likelihood factor) as closely as possible. Typically more accurate than Laplace, but slower.

Usage

```
approx_laplace(maxiter = 30, tol = 1e-04)

approx_ep(damping = 0.9, quad_order = 11, maxiter = 100)
```

Arguments

`maxiter`	Maximum number of iterations in the Laplace/EP iteration.
`tol`	Convergence tolerance.
`damping`	Damping factor for EP. Should be between 0 and 1. Smaller values typically lead to more stable iterations, but also increase the number of iterations, and thus make the algorithm slower.
`quad_order`	Order of the Gauss-Hermite quadrature used to evaluate the required tilted moments in EP.

Value

The approximation object.

References

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian processes for machine learning. MIT Press.

Examples

```

# Basic usage
gp <- gp_init(
  cfs = cf_sexp(),
  lik = lik_bernoulli(),
  method = method_fitc(num_inducing = 100),
  approx = approx_ep()
)
```

cf — *Initialize covariance function*

Description

Functions for initializing the covariance functions which can then be passed to gp_init. See section Details for explanation of what covariance function is what.

Usage

```
cf_const(magn = 1, prior_magn = prior_logunif())

cf_lin(vars = NULL, magn = 1, prior_magn = prior_logunif(), normalize = FALSE)

cf_sexp(
  vars = NULL,
  lscale = 0.3,
  magn = 1,
  prior_lscale = prior_logunif(),
  prior_magn = prior_logunif(),
  normalize = FALSE
)

cf_matern32(
  vars = NULL,
  lscale = 0.3,
  magn = 1,
  prior_lscale = prior_logunif(),
  prior_magn = prior_logunif(),
  normalize = FALSE
)

cf_matern52(
  vars = NULL,
  lscale = 0.3,
  magn = 1,
  prior_lscale = prior_logunif(),
  prior_magn = prior_logunif(),
  normalize = FALSE
)

cf_nn(
  vars = NULL,
  sigma0 = 1,
  sigma = 3,
  magn = 1,
  prior_sigma0 = prior_half_t(),
  prior_sigma = prior_half_t(),
  prior_magn = prior_logunif(),
  normalize = TRUE
)

cf_periodic(
  vars = NULL,
  period = 1,
  cf_base = cf_sexp(),
  prior_period = prior_logunif()
)

cf_prod(...)

## S3 method for class 'cf'
cf1 * cf2
```

Arguments

`magn`	Initial value for the magnitude hyperparameter (depicts the magnitude of the variation captured by the given covariance function).
`prior_magn`	Prior for hypeparameter `magn`. See `priors`.
`vars`	Indices of the inputs which are taken into account when calculating this covariance. If the input matrix has named columns, can also be a vector of column names. Default is all the inputs.
`normalize`	Whether to automatically scale and center the inputs for the given covariance function. Can be useful for inputs with mean and variance far from 0 and 1, respectively.
`lscale`	Initial value for the length-scale hyperparameter.
`prior_lscale`	Prior for hyperparameter `lscale`. See `priors`.
`sigma0`	Prior std for the bias in the neural network covariance function.
`sigma`	Prior std for the weights in the hidden layers of the neural network covariance function.
`prior_sigma0`	Prior for hyperparameter `sigma0`. See `priors`.
`prior_sigma`	Prior for hyperparameter `sigma`. See `priors`.
`period`	Period length for the periodic covariance function.
`cf_base`	Base covariance function that is used to model the variability within each period in periodic covariance function.
`prior_period`	Prior for hyperparameter `period`. See `priors`.
`...`	Meaning depends on context. For `cf_prod` pass in the covariance functions in the product.
`cf1`	Instance of a covariance function.
`cf2`	Instance of a covariance function.

Details

The supported covariance functions are (see Rasmussen and Williams, 2006):

`cf_const` Constant covariance function. Can be used to model the intercept.

`cf_lin` Linear covariance function. Produces linear functions.

`cf_sexp` Squared exponential (or exponentiated quadratic, or Gaussian) covariance function.

`cf_matern32` Matern nu=3/2 covariance function.

`cf_matern52` Matern nu=5/2 covariance function.

`cf_nn` Neural network covariance function.

`cf_periodic` Periodic covariance function. The periodicity is achieved by mapping the original inputs through sine and cosine functions, and then applying the base kernel in this new space.

`cf_prod` Product of two or more covariance functions.

Value

The covariance function object.

References

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian processes for machine learning. MIT Press.

Examples

```
# Generate some toy data
set.seed(1242)
n <- 50
x <- matrix(rnorm(n * 3), nrow = n)
f <- sin(x[, 1]) + 0.5 * x[, 2]^2 + x[, 3]
y <- f + 0.5 * rnorm(n)
x <- data.frame(x1 = x[, 1], x2 = x[, 2], x3 = x[, 3])

# Basic usage (single covariance function)
cf <- cf_sexp()
lik <- lik_gaussian()
gp <- gp_init(cf, lik)
gp <- gp_optim(gp, x, y)
plot(gp_pred(gp, x)$mean, y)

# More than one covariance function; one for x1 and x2, and another one for x3
cf1 <- cf_sexp(c("x1", "x2"))
cf2 <- cf_lin("x3")
cfs <- list(cf1, cf2)
lik <- lik_gaussian()
gp <- gp_init(cfs, lik)
gp <- gp_optim(gp, x, y, maxiter = 500)
plot(gp_pred(gp, x)$mean, y)
plot(x[, 3], gp_pred(gp, x, cfind = 2)$mean) # plot effect w.r.t x3 only
```

gp_draw *Make predictions with a GP model*

Description

Function gp_pred can be used to make analytic predictions for the latent function values at test points, whereas gp_draw can be used to draw from the predictive distribution (or from the prior if the GP has not been fitted yet.)

Usage

```
gp_draw(
  gp,
  xnew,
  draws = NULL,
  transform = TRUE,
  target = FALSE,
  marginal = FALSE,
  cfind = NULL,
  jitter = NULL,
  seed = NULL,
  ...
)

gp_pred(
  gp,
  xnew,
  var = FALSE,
  quantiles = NULL,
  transform = FALSE,
  cfind = NULL,
  jitter = NULL,
  quad_order = 15,
  ...
)
```

Arguments

gp	A GP model object.
xnew	N-by-d matrix of input values (N is the number of test points and d the input dimension). Can also be a vector of length N if the model has only a single input.
draws	Number of draws to generate from the predictive distribution for the latent values.

`transform`	Whether to transform the draws of latent values to the same scale as the target y, that is, through the response (or inverse-link) function.
`target`	If TRUE, draws values for the target variable `y` instead of the latent function values.
`marginal`	If TRUE, then draws for each test point are only marginally correct, but the covariance structure between test points is not retained. However, this will make the sampling considerably faster in some cases, and can be useful if one is interested only in looking at the marginal predictive distributions for a large number of test locations (for example, in posterior predictive checking).
`cfind`	Indices of covariance functions to be used in the prediction. By default uses all covariance functions.
`jitter`	Magnitude of diagonal jitter for covariance matrices for numerical stability. Default is 1e-6.
`seed`	Random seed for draws.
`...`	Additional parameters that might be needed. For example `offset` or keyword `trials` for binomial and beta-binomial likelihoods.
`var`	Whether to compute the predictive variances along with predictive mean.
`quantiles`	Vector of probabilities between 0 and 1 indicating which quantiles are to be predicted.
`quad_order`	Quadrature order in order to compute the mean and variance on the transformed scale.

Value

`gp_pred` returns a list with fields giving the predictive mean, variance and quantiles (the last two are computed only if requested). `gp_draw` returns an N-by-draws matrix of random draws from the predictive distribution, where N is the number of test points.

References

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian processes for machine learning. MIT Press.

Examples

```
# Generate some toy data
set.seed(1242)
n <- 50
x <- matrix(rnorm(n * 3), nrow = n)
f <- sin(x[, 1]) + 0.5 * x[, 2]^2 + x[, 3]
y <- f + 0.5 * rnorm(n)
x <- data.frame(x1 = x[, 1], x2 = x[, 2], x3 = x[, 3])

# More than one covariance function; one for x1 and x2, and another one for x3
cf1 <- cf_nn(c("x1", "x2"), prior_sigma0 = prior_half_t(df = 4, scale = 2))
cf2 <- cf_sexp("x3")
```

```
cfs <- list(cf1, cf2)
lik <- lik_gaussian()
gp <- gp_init(cfs, lik)
gp <- gp_optim(gp, x, y, maxiter = 500)

# plot the predictions with respect to x1, when x2 = x3 = 0
xt <- cbind(x1 = seq(-3, 3, len = 50), x2 = 0, x3 = 0)
pred <- gp_pred(gp, xt)
plot(xt[, "x1"], pred$mean, type = "l")

# draw from the predictive distribution
xt <- cbind(x1 = seq(-3, 3, len = 50), x2 = 0, x3 = 0)
draws <- gp_draw(gp, xt, draws = 100)
plot(xt[, "x1"], draws[, 1], type = "l")
for (i in 2:50) {
  lines(xt[, "x1"], draws[, i])
}

# plot effect with respect to x3 only
xt <- cbind("x3" = seq(-3, 3, len = 50))
pred <- gp_pred(gp, xt, cfind = 2)
plot(xt, pred$mean, type = "l")
```

---

gp_energy *Energy of a GP model*

Description

Returns the energy (negative log marginal likelihood) of a fitted GP model with the current hyper-parameters. The result is exact for the Gaussian likelihood and dependent on the `approx` for other cases.

Usage

```
gp_energy(gp, include_prior = TRUE)
```

Arguments

`gp`	The fitted GP model.
`include_prior`	Whether to add log density of the prior to the result (in which case the result is -(log marginal likelihood + log prior))

Value

The energy value (negative log marginal likelihood).

References

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian processes for machine learning. MIT Press.

Examples

```
# Generate some toy data
set.seed(1242)
n <- 500
x <- matrix(rnorm(n * 3), nrow = n)
f <- sin(x[, 1]) + 0.5 * x[, 2]^2 + x[, 3]
y <- f + 0.5 * rnorm(n)
x <- data.frame(x1 = x[, 1], x2 = x[, 2], x3 = x[, 3])

# Basic usage
gp <- gp_init(cf_sexp(), lik_gaussian())
gp <- gp_fit(gp, x, y)
e <- gp_energy(gp)
```

`gp_fit` *Fit a GP model*

Description

Function `gp_fit` fits a GP model with the current hyperparameters. Notice that this function does not optimize the hyperparameters in any way, but only finds the analytical posterior approximation (depending on chosen `approx`) for the latent values with the current hyperparameters. For optimizing the hyperparameter values, see `gp_optim`.

Usage

```
gp_fit(gp, x, y, trials = NULL, offset = NULL, jitter = NULL, ...)
```

Arguments

`gp`	The gp model object to be fitted.
`x`	n-by-d matrix of input values (n is the number of observations and d the input dimension). Can also be a vector of length n if the model has only a single input.
`y`	Vector of n output (target) values.
`trials`	Vector of length n giving the number of trials for each observation in binomial (and beta binomial) model.
`offset`	Vector of constant values added to the latent values f_i (i = 1,...,n). For Poisson models, this is the logarithm of the exposure time in each observation.

`jitter`	Magnitude of diagonal jitter for covariance matrices for numerical stability. Default is 1e-6.
`...`	Currently ignored

Value

An updated GP model object.

References

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian processes for machine learning. MIT Press.

Examples

```
# Generate some toy data
set.seed(32004)
n <- 150
sigma <- 0.1
x <- rnorm(n)
ycont <- sin(3 * x) * exp(-abs(x)) + rnorm(n) * sigma
y <- rep(0, n)
y[ycont > 0] <- 1
trials <- rep(1, n)

# Fit the model using Laplace approximation (with the specified hyperparameters)
cf <- cf_sexp(lscale = 0.3, magn = 3)
gp <- gp_init(cf, lik_binomial())
gp <- gp_fit(gp, x, y, trials = trials)
```

`gp_init` *Initialize a GP model*

Description

Initializes a GP model with given covariance function(s) and likelihood. The model can then be fitted using `gp_fit`. For hyperparameter optimization, see `gp_optim`

Usage

```
gp_init(
  cfs = cf_sexp(),
  lik = lik_gaussian(),
  method = method_full(),
  approx = approx_laplace()
)
```

Arguments

`cfs`	The covariance function(s). Either a single covariance function or a list of them. See `cf`.
`lik`	Likelihood (observation model). See `lik`.
`method`	Method for approximating the covariance function. See `method`.
`approx`	Approximate inference method for Gaussian approximation for the posterior of the latent values. See `approx`.

Value

A GP model object that can be passed to other functions, for example when optimizing the hyperparameters or making predictions.

References

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian processes for machine learning. MIT Press.

Examples

```
# Full exact GP with Gaussian likelihood
gp <- gp_init(
  cfs = cf_sexp(),
  lik = lik_gaussian(),
  method = method_full()
)

# Binary classification model with EP approximation for the latent values
# and FITC sparse approximation to facilitate large datasets
gp <- gp_init(
  cfs = cf_sexp(),
  lik = lik_bernoulli(),
  approx = approx_ep(),
  method = method_fitc(num_inducing = 100)
)
```

`gp_loo` *Model assessment and comparison*

Description

Function `gp_loo` computes the approximate leave-one-out (LOO) cross-validation statistics for the given GP model with the current hyperparameters. Function `gp_compare` estimates the difference in the expected predictive accuracy of two or more GP models given their LOO statistics.

Usage

```
gp_loo(
  gp,
  x,
  y,
  quadrature = TRUE,
  quad_order = 11,
  draws = 4000,
  jitter = NULL,
  seed = NULL,
  ...
)

gp_compare(..., ref = NULL)
```

Arguments

`gp`	The gp model object to be fitted.
`x`	n-by-d matrix of input values (n is the number of observations and d the input dimension). Can also be a vector of length n if the model has only a single input.
`y`	Vector of n output (target) values.
`quadrature`	Whether to use deterministic Gauss-Hermite quadrature to estimate the required integrals. If FALSE, then Monte Carlo estimate is used.
`quad_order`	Order of the numerical quadrature (only applicable if `quadrature=TRUE`).
`draws`	Number of posterior draws to estimate the required integrals (only applicable if `quadrature=FALSE`).
`jitter`	Magnitude of diagonal jitter for covariance matrices for numerical stability. Default is 1e-6.
`seed`	Random seed.
`...`	For `gp_compare`, LOO statistics for the models to compare. For `gp_loo`, possible additional data that is required for LOO predictions (for example, argument `trials` in case of binomial likelihood).
`ref`	Index of the model against which to compare the other models (pairwise comparison for LOO difference). If not given, then the model with the best LOO is used as the reference for comparisons.

Value

`gp_loo` returns a list with LOO statistics. `gp_compare` returns a matrix with comparison statistics (LOO differences and stardard errors in the estimates).

References

Vehtari A., Mononen T., Tolvanen V., Sivula T. and Winther O. (2016). Bayesian Leave-One-Out Cross-Validation Approximations for Gaussian Latent Variable Models. Journal of Machine Learning Research 17(103):1-38.

Examples

```
# Generate some toy data
set.seed(32004)
n <- 50
sigma <- 0.1
x <- rnorm(n)
ycont <- sin(3 * x) * exp(-abs(x)) + rnorm(n) * sigma
y <- rep(0, n)
y[ycont > 0] <- 1
trials <- rep(1, n)

# Set up two models
gp1 <- gp_init(cf_sexp(), lik_binomial())
gp2 <- gp_init(cf_matern32(), lik_binomial())

# Optimize
gp1 <- gp_optim(gp1, x, y, trials = trials)
gp2 <- gp_optim(gp2, x, y, trials = trials)

# Compare
loo1 <- gp_loo(gp1, x, y, trials = trials)
loo2 <- gp_loo(gp2, x, y, trials = trials)
gp_compare(loo1, loo2)
```

`gp_optim` *Optimize hyperparameters of a GP model*

Description

This function can be used to optimize the hyperparameters of the model to the maximum marginal likelihood (or maximum marginal posterior if priors are used), using Nelder-Mead algorithm.

Usage

```
gp_optim(
  gp,
  x,
  y,
  tol = 1e-04,
  tol_param = 0.1,
  maxiter = 500,
  restarts = 1,
  verbose = TRUE,
  warnings = TRUE,
  ...
)
```

Arguments

`gp`	The gp model object to be fitted.
`x`	n-by-d matrix of input values (n is the number of observations and d the input dimension). Can also be a vector of length n if the model has only a single input.
`y`	Vector of n output (target) values.
`tol`	Relative change in the objective function value (marginal log posterior) after which the optimization is terminated. This will be passed to the function stats::optim as a convergence criterion.
`tol_param`	After the optimizer (Nelder-Mead) has terminated, the found hyperparameter values will be checked for convergence within tolerance tol_param. More precisely, if we perturb any of the hyperparameters by the amount tol_param or -tol_param, then the resulting log posterior must be smaller than the value with the found hyperparameter values. If not, then the optimizer will automatically attempt a restart (see argument restarts). Note: tol_param will be applied for the logarithms of the parameters (e.g. log length-scale), not for the native parameter values.
`maxiter`	Maximum number of iterations.
`restarts`	Number of possible restarts during optimization. The Nelder-Mead iteration can sometimes terminate prematurely before a local optimum is found, and this argument can be used to specify how many times the optimization is allowed to restart from where it left when Nelder-Mead terminated. By setting restarts > 0, one can often find local optimum without having to call gp_optim several times. Note: usually there is no need to allow more than a few (say 1-3) restarts; if the optimization does not converge with a few restarts, then one usually must try to reduce argument tol in order to achieve convergence. If this does not help either, then the optimization problem is usually ill-conditioned somehow.
`verbose`	If TRUE, then some information about the progress of the optimization is printed to the console.
`warnings`	Whether to print out some potential warnings (such as maximum number of iterations reached) during the optimization.
`...`	Further arguments to be passed to `gp_fit` that are needed in the fitting process, for example `trials` in the case of binomial likelihood.

Value

An updated GP model object.

References

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian processes for machine learning. MIT Press.

Examples

```
# Generate some toy data
```

```
set.seed(1242)
n <- 50
x <- matrix(rnorm(n * 3), nrow = n)
f <- sin(x[, 1]) + 0.5 * x[, 2]^2 + x[, 3]
y <- f + 0.5 * rnorm(n)
x <- data.frame(x1 = x[, 1], x2 = x[, 2], x3 = x[, 3])

# Basic usage
cf <- cf_sexp()
lik <- lik_gaussian()
gp <- gp_init(cf, lik)
gp <- gp_optim(gp, x, y)
```

gp_saveload   *Save and load a GP model*

Description

Convenience functions for saving and loading GP models.

Usage

```
gp_save(gp, filename)

gp_load(filename)
```

Arguments

`gp`	The gp model object to be saved.
`filename`	Where to save or load from.

Value

`gp_load` returns the loaded GP model object.

Examples

```
gp <- gp_init()

# fit the model (skipped here)

# save the model
filename <- file.path(tempdir(), 'gp.rda')
gp_save(gp, filename)

# load the model and remove the file
```

```
gp <- gp_load(filename)
file.remove(filename)
```

lik *Initialize likelihood*

Description

Functions for initializing the likelihood (observation model) which can then be passed to `gp_init`.

Usage

```
lik_gaussian(sigma = 0.5, prior_sigma = prior_logunif())

lik_bernoulli(link = "logit")

lik_binomial(link = "logit")

lik_betabinom(link = "logit", phi = 1, prior_phi = prior_logunif())

lik_poisson(link = "log")
```

Arguments

`sigma`	Initial value for the noise standard deviation.
`prior_sigma`	Prior for hyperparameter `sigma`. See `priors`.
`link`	Link function if the likelihood supports non-identity links. See Details for information about possible links for each likelihood.
`phi`	The over dispersion parameter for beta binomial likelihood.
`prior_phi`	Prior for hyperparameter `phi`. See `priors`.

Details

The supported likelihoods are:

`lik_gaussian` Gaussian likelihood. Has no links (uses identity link).

`lik_bernoulli` Bernoulli likelihood. Possible links: 'logit' or 'probit'.

`lik_binomial` Binomial likelihood. Possible links: 'logit' or 'probit'.

`lik_betabinom` Beta binomial likelihood. Possible links: 'logit' or 'probit'.

`lik_poisson` Poisson likelihood. Possible links: 'log'.

Value

The likelihood object.

Examples

```
# Basic usage
cf <- cf_sexp()
lik <- lik_binomial()
gp <- gp_init(cf, lik)
```

---

method *Initialize method or type of the model*

Description

Functions for initializing the method or type of the model, which can then be passed to `gp_init`. The supported methods are:

`method_full` Full GP, so full exact covariance function is used, meaning that the inference will be for the `n` latent function values (fitting time scales cubicly in `n`).

`method_fitc` Fully independent training (and test) conditional, or FITC, approximation (see Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2006). The fitting time scales `O(n*m^2)`, where n is the number of data points and m the number of inducing points `num_inducing`. The inducing point locations are chosen using the k-means algorithm.

`method_rf` Random features, that is, linearized GP. Uses random features (or basis functions) for approximating the covariance function, which means the inference time scales cubicly in the number of approximating basis functions `num_basis`. For stationary covariance functions random Fourier features (Rahimi and Recht, 2007) is used, and for non-stationary kernels using case specific method when possible (for example, drawing the hidden layer parameters randomly for `cf_nn`). For `cf_const` and `cf_lin` this means using standard linear model, and the inference is performed on the weight space (not in the function space). Thus if the model is linear (only `cf_const` and `cf_lin` are used), this will give a potentially huge speed-up if the number of features is considerably smaller than the number of data points.

Usage

```
method_full()

method_fitc(
  inducing = NULL,
  num_inducing = 100,
  bin_along = NULL,
  bin_count = 10,
  seed = 12345
)

method_rf(num_basis = 400, seed = 12345)
```

Arguments

inducing	Inducing points to use. If not given, then num_inducing points will be placed in the input space using a clustering algorithm.
num_inducing	Number of inducing points for the approximation. Will be ignored if the inducing points are given by the user.
bin_along	Either an index or a name of the input variable along which to bin the values before placing the inducing inputs. For example, if bin_along=3, then the input data is divided into bin_count bins along 3rd input variable, and each bin will have the same number inducing points (or as close as possible). This can sometimes be useful to ensure that inducing points are spaced evenly with respect to some particular variable, for example time in spatio-temporal models.
bin_count	The number of bins to use if bin_along given. Has effect only if bin_along is given.
seed	Random seed for reproducible results.
num_basis	Number of basis functions for the approximation.

Value

The method object.

References

Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. In Advances in Neural Information Processing Systems 20.

Quiñonero-Candela, J. and Rasmussen, C. E (2005). A unifying view of sparse approximate Gaussian process regression. Journal of Machine Learning Research 6:1939-1959.

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In Advances in Neural Information Processing Systems 18.

Examples

```
#' # Generate some toy data
# NOTE: this is so small dataset that in reality there would be no point
# use sparse approximation here; we use this small dataset only to make this
# example run fast
set.seed(1242)
n <- 50
x <- matrix(rnorm(n * 3), nrow = n)
f <- sin(x[, 1]) + 0.5 * x[, 2]^2 + x[, 3]
y <- f + 0.5 * rnorm(n)
x <- data.frame(x1 = x[, 1], x2 = x[, 2], x3 = x[, 3])

# Full exact GP with Gaussian likelihood
gp <- gp_init(cf_sexp())
gp <- gp_optim(gp, x, y)
```

```
# Approximate solution using random features (here we use a very small
# number of random features only to make this example run fast)
gp <- gp_init(cf_sexp(), method = method_rf(num_basis = 30))
gp <- gp_optim(gp, x, y)

# Approximate solution using FITC (here we use a very small
# number of incuding points only to make this example run fast)
gp <- gp_init(cf_sexp(), method = method_fitc(num_inducing = 10))
gp <- gp_optim(gp, x, y)
```

param *Get or set GP model parameters*

Description

get_param returns the current hyperparameters of the GP model in a vector. set_param can be used to set the parameters. Note that these functions are intended mainly for internal usage, and there is typically no need to use these functions directly but instead create a new GP model using gp_init.

Usage

```
get_param(object, ...)

set_param(object, param, ...)
```

Arguments

object	The model object.
...	Ignored currently.
param	The parameters to be set. Call first get_param to see the order in which the parameters should be given for a particular model. Notice that all positive parameters should be given in a log-scale.

Value

get_param returns the current hyperparameters and set_param the GP model structure with the new parameter values.

Examples

```
# Set up some model
gp <- gp_init(cf = cf_sexp(), lik = lik_gaussian())

# print out to see the parameter ordering
param <- get_param(gp)
```

```
print(param)

# set some new values
param_new <- log(c(0.1, 0.8, 0.3))
names(param_new) <- names(param)
gp <- set_param(gp, param_new)

# check the result
print(get_param(gp))
```

`priors` *Initialize prior for hyperparameter*

Description

Functions for initializing hyperparameter priors which can then be passed to `gp_init`. See section Details for the prior explanations.

Usage

```
prior_fixed()

prior_logunif()

prior_lognormal(loc = 0, scale = 1)

prior_half_t(df = 1, scale = 1)
```

Arguments

`loc`	Location parameter of the distribution
`scale`	Scale parameter of the distribution
`df`	Degrees of freedom

Details

The supported priors are:

`prior_fixed` The hyperparameter is fixed to its initial value, and is not optimized by `gp_optim`.

`prior_logunif` Improper uniform prior on the log of the parameter.

`prior_lognormal` Log-normal prior (Gaussian prior on the logarithm of the parameter).

`prior_half_t` Half Student-t prior for a positive parameter.

Value

The hyperprior object.

References

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian processes for machine learning. MIT Press.

Examples

```
# Quasi-periodic covariance function, with fixed period
cf1 <- cf_periodic(
  period = 5,
  prior_period = prior_fixed(),
  cf_base = cf_sexp(lscale = 2)
)
cf2 <- cf_sexp(lscale = 40)
cf <- cf1 * cf2
gp <- gp_init(cf)

# draw from the prior
set.seed(104930)
xt <- seq(-10, 10, len = 500)
plot(xt, gp_draw(gp, xt), type = "l")
```

Index