Package ‘GPTreeO’

October 16, 2024

Type Package

Title Dividing Local Gaussian Processes for Online Learning Regression

Version 1.0.1

Maintainer Timo Braun <gptreeo.timo.braun@gmail.com>

Description We implement and extend the Dividing Local Gaussian Process algorithm by Lederer et al. (2020) <doi:10.48550/arXiv.2006.09446>. Its main use case is in online learning where it is used to train a network of local GPs (referred to as tree) by cleverly partitioning the input space. In contrast to a single GP, 'GPTreeO' is able to deal with larger amounts of data. The package includes methods to create the tree and set its parameter, incorporating data points from a data stream as well as making joint predictions based on all relevant local GPs.

License MIT + file LICENSE

Imports R6, hash, DiceKriging, mlegp

Suggests knitr, rmarkdown, spelling, testthat

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.1

Language en-US

NeedsCompilation no

Author Timo Braun [aut, cre] (<https://orcid.org/0009-0001-0965-8285>), Anders Kvellestad [aut] (<https://orcid.org/0000-0002-5267-7705>), Riccardo De Bin [ctb] (<https://orcid.org/0000-0002-7441-6880>)

Repository CRAN

Date/Publication 2024-10-16 15:20:02 UTC

Contents

CreateWrappedGP 2
GPNode 3
GPTree 7

WrappedDiceKrigingGP 15
WrappedGP 20
WrappedmlegpGP 24

Index **30**

CreateWrappedGP *Factory function called by GPNode to create the wrapper for a specified GP package*

Description

Factory function called by GPNode to create the wrapper for a specified GP package

Usage

```
CreateWrappedGP(
  wrapper,
  X,
  y,
  y_var,
  gp_control,
  init_covpars,
  retrain_buffer_length,
  add_buffer_in_prediction
)
```

Arguments

`wrapper`	A string specifying what GP implementation is used
`X`	Input data matrix with x_dim columns and at maximum Nbar rows. Is used to create the first iteration of the local GP.
`y`	Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored
`y_var`	Variance of the target variable; has to be a one-dimensional matrix or vector
`gp_control`	A list of GP implementation-specific options, passed directly to the wrapped GP implementation
`init_covpars`	Initial covariance parameters of the local GP
`retrain_buffer_length`	Only retrain when the number of buffer points or collected points exceeds this value
`add_buffer_in_prediction`	If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated.

Details

A detailed list of expected functions from GPTree and GPNode can be found in the comments of this file. Currently, GPs from the `DiceKriging` package (WrappedDiceKrigingGP) and `mlegp` package (WrappedmlegpGP) are implemented. The user can create their own wrapper using WrappedGP.

Value

The wrapper of the chosen GP package, containing the respective GP and information on the shared points and those stored in the buffer.

`GPNode` *R6 Class for the nodes / leaves in the GPTree tree*

Description

The nodes contain the local GP if they are leaves (at the end of a branch). Nodes that are just nodes contain information on how the input space was split. They are responsible for computing and updating the splitting probabilities. Also, the tree interacts with the local GPs through the nodes.

Currently, GPs from the `DiceKriging` package (WrappedDiceKrigingGP) and `mlegp` package (WrappedmlegpGP) are implemented. The user can create their own wrapper using WrappedGP.

Public fields

`key` A string like "0110100" to identify the node in the binary tree

`x_dim` Dimensionality of input points. It is set once the first point is received through the GPTree method `update`. It needs to be specified if `min_ranges` should be different from default.

`theta` Overlap ratio between two leafs in the split direction. The default value is 0.

`split_direction_criterion` A string that indicates which spitting criterion to use. The options are:

- `"max_spread"`: Split along the direction which has the largest data spread.
- `"min_lengthscale"`: split along the direction with the smallest length-scale hyperparameter from the local GP.
- `"max_spread_per_lengthscale"`: Split along the direction with the largest data spread relative to the corresponding GP length-scale hyperparameter.
- `"max_corr"`: Split along the direction where the input data is most strongly correlated with the target variable.
- `"principal_component"`: Split along the first principal component.

The default value is `"max_spread_per_lengthscale"`.

`split_position_criterion` A string indicating how the split position along the split direction should be set. Possible values are (`"mean"` and `"median"`). The default is `"mean"`.

`shape_decay` A string specifying how the probability function for a point to be assigned to the left leaf should fall off in the overlap region. The available options are a linear shape (`"linear"`), an exponential shape (`"exponential"`) or a Gaussian shape (`"gaussian"`). Another option is to select no overlap region. This can be achieved by selecting `"deterministic"` or to set `theta` to 0. The default is `"linear"`.

`prob_min_theta` Minimum probability after which the overlap shape gets truncated (either towards 0 or 1). The default value is 0.01.

`Nbar` Maximum number of data points for each GP in a leaf before it is split. The default value is 1000.

`min_ranges` Smallest allowed input data spread (per dimension) before node splitting stops. It is set to its default `min_ranges = rep(0.0, x_dim)` once the first point is received through the `update` method. `x_dim` needs to be specified by the user if it should be different from the default.

`is_leaf` If TRUE, this node a leaf, i.e the last node on its branch

`wrapped_gp` An instance of the WrappedGP type

`can_split` If TRUE for a given dimension, the leaf can be split along that dimension

`rotation_matrix` A rotation matrix, used for transforming the data

`shift` A shift, used for transforming the data

`use_pc_transform` TRUE if principal components transformation is used for node splitting

`x_spread` Vector of data spread for each dimension

`split_index` Index for the split dimension

`position_split` Position of the split along dimension split_index

`width_overlap` Width of overlap region along dimension split_index

`point_ids` IDs of the points assigned to this node

`residuals` Vector of residuals

`pred_errs` Vector of prediction uncertainties

`error_scaler` Scaling factor for the prediction error to ensure desired coverage

`use_n_residuals` Number of past residuals to use in calibrating the `error_scaler`

Methods

Public methods:

- `GPNode$new()`
- `GPNode$transform()`
- `GPNode$update_prob_pars()`
- `GPNode$get_prob_child_1()`
- `GPNode$register_residual()`
- `GPNode$update_empirical_error_pars()`
- `GPNode$delete_gp()`
- `GPNode$clone()`

Method `new()`**:** Create a new node object

Usage:

```
GPNode$new(
  key,
  x_dim,
  theta,
  split_direction_criterion,
  split_position_criterion,
  shape_decay,
  prob_min_theta,
  Nbar,
  wrapper,
  gp_control,
  retrain_buffer_length,
  add_buffer_in_prediction,
  min_ranges = NULL,
  is_leaf = TRUE
)
```

Arguments:

`key` A string like "0110100" to identify the node in the binary tree

`x_dim` Dimensionality of input points. It is set once the first point is received through the GPTree method `update`. It needs to be specified if `min_ranges` should be different from default.

`theta` Overlap ratio between two leafs in the split direction. The default value is 0.

`split_direction_criterion` A string that indicates which spitting criterion to use. The options are:

- `"max_spread"`: Split along the direction which has the largest data spread.
- `"min_lengthscale"`: split along the direction with the smallest length-scale hyperparameter from the local GP.
- `"max_spread_per_lengthscale"`: Split along the direction with the largest data spread relative to the corresponding GP length-scale hyperparameter.
- `"max_corr"`: Split along the direction where the input data is most strongly correlated with the target variable.
- `"principal_component"`: Split along the first principal component.

The default value is `"max_spread_per_lengthscale"`.

`split_position_criterion` A string indicating how the split position along the split direction should be set. Possible values are (`"mean"` and `"median"`). The default is `"mean"`.

`shape_decay` A string specifying how the probability function for a point to be assigned to the left leaf should fall off in the overlap region. The available options are a linear shape (`"linear"`), an exponential shape (`"exponential"`) or a Gaussian shape (`"gaussian"`). Another option is to select no overlap region. This can be achieved by selecting `"deterministic"` or to set `theta` to 0. The default is `"linear"`.

`prob_min_theta` Minimum probability after which the overlap shape gets truncated (either towards 0 or 1). The default value is 0.01.

`Nbar` Maximum number of data points for each GP in a leaf before it is split. The default value is 1000.

`wrapper` A string that indicates which GP implementation should be used. The current version includes wrappers for the packages `"DiceKriging"` and `"mlegp"`. The default setting is `"DiceKriging"`.

`gp_control` A `list` of control parameter that is forwarded to the wrapper. Here, the covariance function is specified. `DiceKriging` allows for the following kernels, passed as string: `"gauss"`, `"matern5_2"`, `"matern3_2"`, `"exp"`, `"powexp"` where `"matern3_2"` is set as default.

`retrain_buffer_length` Size of the retrain buffer. The buffer for a each node collects data points and holds them until the buffer length is reached. Then the GP in the node is updated with the data in the buffer. For a fixed `Nbar`, higher values for `retrain_buffer_length` lead to faster run time (less frequent retraining), but the trade-off is a temporary reduced prediction accuracy. We advise that the choice for `retrain_buffer_length` should depend on the chosen `Nbar`. By default `retrain_buffer_length` is set equal to `Nbar`.

`add_buffer_in_prediction` If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated. The default is `FALSE`.

`min_ranges` Smallest allowed input data spread (per dimension) before node splitting stops. It is set to its default `min_ranges = rep(0.0, x_dim)` once the first point is received through the GPTree method `update`. `x_dim` needs to be specified by the user if it should be different from the default.

`is_leaf` If TRUE, this node a leaf, i.e the last node on its branch.

`n_points_train_limit` Number of points at which a GP is created in the leaf

Returns: A new GPNode object. Contains the local GP in the field `wrapped_gp`, and information used for and related to splitting the node. If the node has been split, the local GP is removed.

Method `transform()`**:** Method to transform input data through a shift and a rotation. IS EXPECTED TO NOT BE CALLED BY THE USER

Usage:

```
GPNode$transform(X)
```

Arguments:

`X` Matrix with x points

Returns: The transformed X matrix

Method `update_prob_pars()`**:** Method to update the probability parameters (x_spread, can_split, split_index, position_split, width_overlap). IS EXPECTED TO NOT BE CALLED BY THE USER

Usage:

```
GPNode$update_prob_pars()
```

Method `get_prob_child_1()`**:** Method to compute the probability that a point x should go to child 1. IS EXPECTED TO NOT BE CALLED BY THE USER

Usage:

```
GPNode$get_prob_child_1(x)
```

Arguments:

`x` Single data point for which probability is computed; has to be a vector with length equal to x_dim

Returns: The probability that a point x should go to child 1

Method `register_residual()`**:** Method to register prediction performance

Usage:

```
GPNode$register_residual(x, y)
```

Arguments:

`x` Most recent single input data point from the data stream; has to be a vector with length equal to x_dim

`y` Target variable which has to be a one-dimensional matrix or a vector; any further columns will be ignored

Method `update_empirical_error_pars()`**:** Method for updating the empirical error parameters

Usage:

```
GPNode$update_empirical_error_pars()
```

Method `delete_gp()`**:** Method to delete the GP. IS EXPECTED TO NOT BE CALLED BY THE USER

Usage:

```
GPNode$delete_gp()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GPNode$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`GPTree()` for the main methods

`GPTree` *Tree structure storing all nodes containing local GPs*

Description

The base class which contains and where all parameters are set. Here, all information on how and when the splitting is carried out is stored. `wrapper` and `gp_control` specify the Gaussian process (GP) implementation and its parameters. Moreover, minimum errors and calibration of the predictions are specified here, too.

Essential methods

The following three methods are essential for the package. The remaining ones are mostly not expected to be called by the user.

- `GPTree$new()`: Creates a new tree with specified parameters
- `GPTree$update()`: Adds the information from the input point to the tree and updates local GPs
- `GPTree$joint_prediction()`: Computes the joint prediction for a given input point

Brief package functionality overview

The tree collects the information from all GPNodes which in turn contain the local GP. Currently, GPs from the DiceKriging package (WrappedDiceKrigingGP) and mlegp package (Wrappedm-legpGP) are implemented. The user can create their own wrapper using WrappedGP.

Public fields

`Nbar` Maximum number of data points for each GP in a leaf before it is split. The default value is 1000.

`retrain_buffer_length` Size of the retrain buffer. The buffer for a each node collects data points and holds them until the buffer length is reached. Then the GP in the node is updated with the data in the buffer. For a fixed `Nbar`, higher values for `retrain_buffer_length` lead to faster run time (less frequent retraining), but the trade-off is a temporary reduced prediction accuracy. We advise that the choice for `retrain_buffer_length` should depend on the chosen `Nbar`. By default `retrain_buffer_length` is set equal to `Nbar`.

`gradual_split` If TRUE, gradual splitting is used for splitting. The default value is TRUE.

`theta` Overlap ratio between two leafs in the split direction. The default value is 0.

`wrapper` A string that indicates which GP implementation should be used. The current version includes wrappers for the packages `"DiceKriging"` and `"mlegp"`. The default setting is `"DiceKriging"`.

`gp_control` A `list` of control parameter that is forwarded to the wrapper. Here, the covariance function is specified. `DiceKriging` allows for the following kernels, passed as string: `"gauss"`, `"matern5_2"`, `"matern3_2"`, `"exp"`, `"powexp"` where `"matern3_2"` is set as default.

`split_direction_criterion` A string that indicates which spitting criterion to use. The options are:

- `"max_spread"`: Split along the direction which has the largest data spread.
- `"min_lengthscale"`: split along the direction with the smallest length-scale hyperparameter from the local GP.
- `"max_spread_per_lengthscale"`: Split along the direction with the largest data spread relative to the corresponding GP length-scale hyperparameter.
- `"max_corr"`: Split along the direction where the input data is most strongly correlated with the target variable.
- `"principal_component"`: Split along the first principal component.

The default value is `"max_spread_per_lengthscale"`.

`split_position_criterion` A string indicating how the split position along the split direction should be set. Possible values are (`"median"` and `"mean"`). The default is `"median"`.

`shape_decay` A string specifying how the probability function for a point to be assigned to the left leaf should fall off in the overlap region. The available options are a linear shape (`"linear"`), an exponential shape (`"exponential"`) or a Gaussian shape (`"gaussian"`). Another option is to select no overlap region. This can be achieved by selecting `"deterministic"` or to set `theta` to 0. The default is `"linear"`.

`use_empirical_error` If TRUE, the uncertainty is calibrated using recent data points. The default value is TRUE.

The most recent 25 observations are used to ensure that the prediction uncertainty yields approximately 68 % coverage. This coverage is only achieved if `theta = 0` (also together with `gradual_split = TRUE`) is used. Nevertheless, the coverage will be closer to 68 % than it would be without calibration. The prediction uncertainties at the beginning are conservative and become less conservative with increasing number of input points.

`use_reference_gp` If TRUE, the covariance parameters determined for the GP in node 0 will be used for all subsequent GPs. The default is `FALSE`.

`min_abs_y_err` Minimum absolute error assumed for y data. The default value is 0.

`min_rel_y_err` Minimum relative error assumed for y data. The default value is `100 * .Machine$double.eps`.

`min_abs_node_pred_err` Minimum absolute error on the prediction from a single node. The default value is 0.

`min_rel_node_pred_err` Minimum relative error on the prediction from a single node. The default value is `100 * .Machine$double.eps`.

`prob_min_theta` Minimum probability after which the overlap shape gets truncated (either towards 0 or 1). The default value is 0.01.

`add_buffer_in_prediction` If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated. The default is `FALSE`.

`x_dim` Dimensionality of input points. It is set once the first point is received through the `update()` or `joint_prediction()` method. It needs to be specified if `min_ranges` should be different from default.

`min_ranges` Smallest allowed input data spread (per dimension) before node splitting stops. It is set to its default `min_ranges = rep(0.0, x_dim)` once the first point is received through the `update()` method. `x_dim` needs to be specified by the user if it should be different from the default.

`max_cond_num` Add additional noise if the covariance matrix condition number exceeds this value. The default is `NULL`.

`max_points` The maximum number of points the tree is allowed to store. The default value is `Inf`.

End of the user-defined input fields.

`nodes` A hash to hold the GP tree, using string keys to identify nodes and their position in the tree ("0", "00", "01", "000", "001", "010", "011", etc.)

`leaf_keys` Stores the keys ("0", "00", "01", "000", "001", "010", "011", etc.) for the leaves

`n_points` Number of points in the tree

`n_fed` Number of points fed to the tree

Methods

Public methods:

- `GPTree$new()`
- `GPTree$add_node()`
- `GPTree$get_marginal_point_prob()`
- `GPTree$update()`
- `GPTree$get_data_split_table()`

- GPTree$joint_prediction()
- GPTree$clone()

Method `new()`**:**

Usage:

```
GPTree$new(
  Nbar = 1000,
  retrain_buffer_length = Nbar,
  gradual_split = TRUE,
  theta = 0,
  wrapper = "DiceKriging",
  gp_control = list(covtype = "matern3_2"),
  split_direction_criterion = "max_spread_per_lengthscale",
  split_position_criterion = "median",
  shape_decay = "linear",
  use_empirical_error = TRUE,
  use_reference_gp = FALSE,
  min_abs_y_err = 0,
  min_rel_y_err = 100 * .Machine$double.eps,
  min_abs_node_pred_err = 0,
  min_rel_node_pred_err = 100 * .Machine$double.eps,
  prob_min_theta = 0.01,
  add_buffer_in_prediction = FALSE,
  x_dim = 0,
  min_ranges = NULL,
  max_cond_num = NULL,
  max_points = Inf
)
```

Arguments:

`Nbar` Maximum number of data points for each GP in a leaf before it is split. The default value is 1000.

`retrain_buffer_length` Size of the retrain buffer. The buffer for a each node collects data points and holds them until the buffer length is reached. Then the GP in the node is updated with the data in the buffer. For a fixed `Nbar`, higher values for `retrain_buffer_length` lead to faster run time (less frequent retraining), but the trade-off is a temporary reduced prediction accuracy. We advise that the choice for `retrain_buffer_length` should depend on the chosen `Nbar`. By default `retrain_buffer_length` is set equal to `Nbar`.

`gradual_split` If TRUE, gradual splitting is used for splitting. The default value is TRUE.

`theta` Overlap ratio between two leafs in the split direction. The default value is 0.

`wrapper` A string that indicates which GP implementation should be used. The current version includes wrappers for the packages `"DiceKriging"` and `"mlegp"`. The default setting is `"DiceKriging"`.

`gp_control` A `list` of control parameter that is forwarded to the wrapper. Here, the covariance function is specified. `DiceKriging` allows for the following kernels, passed as string: `"gauss"`, `"matern5_2"`, `"matern3_2"`, `"exp"`, `"powexp"` where `"matern3_2"` is set as default.

`split_direction_criterion` A string that indicates which spitting criterion to use. The options are:

- `"max_spread"`: Split along the direction which has the largest data spread.
- `"min_lengthscale"`: split along the direction with the smallest length-scale hyperparameter from the local GP.
- `"max_spread_per_lengthscale"`: Split along the direction with the largest data spread relative to the corresponding GP length-scale hyperparameter.
- `"max_corr"`: Split along the direction where the input data is most strongly correlated with the target variable.
- `"principal_component"`: Split along the first principal component.

The default value is `"max_spread_per_lengthscale"`.

`split_position_criterion` A string indicating how the split position along the split direction should be set. Possible values are (`"median"` and `"mean"`). The default is `"median"`.

`shape_decay` A string specifying how the probability function for a point to be assigned to the left leaf should fall off in the overlap region. The available options are a linear shape (`"linear"`), an exponential shape (`"exponential"`) or a Gaussian shape (`"gaussian"`). Another option is to select no overlap region. This can be achieved by selecting `"deterministic"` or to set `theta` to 0. The default is `"linear"`.

`use_empirical_error` If TRUE, the uncertainty is calibrated using recent data points. The default value is TRUE.

The most recent 25 observations are used to ensure that the prediction uncertainty yields approximately 68 % coverage. This coverage is only achieved if `theta = 0` (also together with `gradual_split = TRUE`) is used. Nevertheless, the coverage will be closer to 68 % than it would be without calibration. The prediction uncertainties at the beginning are conservative and become less conservative with increasing number of input points.

`use_reference_gp` If TRUE, the covariance parameters determined for the GP in node 0 will be used for all subsequent GPs. The default is `FALSE`.

`min_abs_y_err` Minimum absolute error assumed for y data. The default value is 0.

`min_rel_y_err` Minimum relative error assumed for y data. The default value is `100 * .Machine$double.eps`.

`min_abs_node_pred_err` Minimum absolute error on the prediction from a single node. The default value is 0.

`min_rel_node_pred_err` Minimum relative error on the prediction from a single node. The default value is `100 * .Machine$double.eps`.

`prob_min_theta` Minimum probability after which the overlap shape gets truncated (either towards 0 or 1). The default value is 0.01.

`add_buffer_in_prediction` If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated. The default is `FALSE`.

`x_dim` Dimensionality of input points. It is set once the first point is received through the `update` method. It needs to be specified if `min_ranges` should be different from default.

`min_ranges` Smallest allowed input data spread (per dimension) before node splitting stops. It is set to its default `min_ranges = rep(0.0, x_dim)` once the first point is received through the `update` method. `x_dim` needs to be specified by the user if it should be different from the default.

`max_cond_num` Add additional noise if the covariance matrix condition number exceeds this value. The default is `NULL`.

`max_points` The maximum number of points the tree is allowed to store. The default value is `Inf`.

Returns: A new GPTree object. Tree-specific parameters are listed in this object. The field `nodes` contains a hash with all GPNodes and information related to nodes. The nodes in turn contain the local GPs. Nodes that have been split no longer contain a GP.

Examples:

```
set.seed(42)
## Use the 1d toy data set from Higdon (2002)
X <- as.matrix(sample(seq(0, 10, length.out = 31)))
y <- sin(2 * pi * X / 10) + 0.2 * sin(2 * pi * X / 2.5)
y_variance <- rep(0.1**2, 31)

## Initialize a tree with Nbar = 15, retrain_buffer_length = 15, use_empirical_error = FALSE,
## and default parameters otherwise
gptree <- GPTree$new(Nbar = 15, retrain_buffer_length = 15, use_empirical_error = FALSE)

## For the purpose of this example, we simulate the data stream through a simple for loop.
## In actual applications, the input stream comes from e.g. a differential evolutionary scanner.
## We follow the procedure in the associated paper, thus letting the tree make a prediction
## first before we update the tree with the point.
for (i in 1:nrow(X)) {
y_pred_with_err = gptree$joint_prediction(X[i,], return_std = TRUE)
## Update the tree with the true (X,y) pair
gptree$update(X[i,], y[i], y_variance[i])
}

## In the following, we go over different initializations of the tree
## 1. The same tree as before, but using the package mlegp:
## Note: since the default for gp_control is gp_control = list(covtype = "matern3_2"),
## we set gp_control to an empty list when using mlegp.
gptree <- GPTree$new(Nbar = 15, retrain_buffer_length = 15, use_empirical_error = FALSE,
wrapper = "mlegp", gp_control = list())

## 2. Minimum working example:
gptree <- GPTree$new()

## 3. Fully specified example corresponding to the default settings
## Here, we choose to specify x_dim and min_ranges so that they correspond to the default values.
## If we do not specifiy them here, they will be automatically specified once
## the update or predict method is called.
gptree <- GPTree$new(Nbar = 1000, retrain_buffer_length = 1000,
gradual_split = TRUE, theta = 0, wrapper = "DiceKriging",
gp_control = list(covtype = "matern3_2"),
split_direction_criterion = "max_spread_per_lengthscale", split_position_criterion = "mean",
shape_decay = "linear", use_empirical_error = TRUE,
use_reference_gp = FALSE, min_abs_y_err = 0, min_rel_y_err = 100 * .Machine$double.eps,
min_abs_node_pred_err = 0, min_rel_node_pred_err = 100 * .Machine$double.eps,
prob_min_theta = 0.01, add_buffer_in_prediction = FALSE, x_dim = ncol(X),
min_ranges = rep(0.0, ncol(X)), max_cond_num = NULL, max_points = Inf)
```

Method `add_node()`: Add a new GPNode to the tree. IS EXPECTED TO NOT BE CALLED BY THE USER

Usage:

```
GPTree$add_node(key)
```

Arguments:

`key` Key of the new leaf

Method `get_marginal_point_prob()`: Marginal probability for point x to belong to node with given key. IS EXPECTED TO NOT BE CALLED BY THE USER

Usage:

```
GPTree$get_marginal_point_prob(x, key)
```

Arguments:

`x` Single input data point from the data stream; has to be a vector with length equal to x_dim

`key` Key of the node

Returns: Returns the marginal probability for point x to belong to node with given key

Method `update()`: Assigns the given input point x with target variable y and associated variance y_var to a node and updates the tree accordingly

Usage:

```
GPTree$update(x, y, y_var = 0, retrain_node = TRUE)
```

Arguments:

`x` Most recent single input data point from the data stream; has to be a vector with length equal to x_dim

`y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored

`y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

`retrain_node` If TRUE, the GP node will be retrained after the point is added.

Details: The methods takes care of both updating an existing node and splitting the parent node into two child nodes. It ensures that the each child node has at least `n_points_train_limit` in each GP. Further handling of duplicate points is also done here.

Method `get_data_split_table()`: Generates a table used to distribute data points from a node to two child nodes

Usage:

```
GPTree$get_data_split_table(current_node)
```

Arguments:

`current_node` The GPNode whose data should be distributed

Returns: A matrix object

Method `joint_prediction()`: Compute the joint prediction from all relevant leaves for an input point x

Usage:

`GPTree$joint_prediction(x, return_std = TRUE)`

Arguments:

`x` Single data point for which the predicted joint mean (and standard deviation) is computed; has to be a vector with length equal to x_dim

`return_std` If TRUE, the standard error of the prediction is returned

Details: We follow Eqs. (5) and (6) in this paper

Returns: The prediction (and its standard error) for input point x from this tree

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`GPTree$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `GPTree$new`
## ------------------------------------------------

set.seed(42)
## Use the 1d toy data set from Higdon (2002)
X <- as.matrix(sample(seq(0, 10, length.out = 31)))
y <- sin(2 * pi * X / 10) + 0.2 * sin(2 * pi * X / 2.5)
y_variance <- rep(0.1**2, 31)

## Initialize a tree with Nbar = 15, retrain_buffer_length = 15, use_empirical_error = FALSE,
## and default parameters otherwise
gptree <- GPTree$new(Nbar = 15, retrain_buffer_length = 15, use_empirical_error = FALSE)

## For the purpose of this example, we simulate the data stream through a simple for loop.
## In actual applications, the input stream comes from e.g. a differential evolutionary scanner.
## We follow the procedure in the associated paper, thus letting the tree make a prediction
## first before we update the tree with the point.
for (i in 1:nrow(X)) {
y_pred_with_err = gptree$joint_prediction(X[i,], return_std = TRUE)
## Update the tree with the true (X,y) pair
gptree$update(X[i,], y[i], y_variance[i])
}

## In the following, we go over different initializations of the tree
## 1. The same tree as before, but using the package mlegp:
## Note: since the default for gp_control is gp_control = list(covtype = "matern3_2"),
## we set gp_control to an empty list when using mlegp.
gptree <- GPTree$new(Nbar = 15, retrain_buffer_length = 15, use_empirical_error = FALSE,
wrapper = "mlegp", gp_control = list())

## 2. Minimum working example:
```

```
gptree <- GPTree$new()

## 3. Fully specified example corresponding to the default settings
## Here, we choose to specify x_dim and min_ranges so that they correspond to the default values.
## If we do not specifiy them here, they will be automatically specified once
## the update or predict method is called.
gptree <- GPTree$new(Nbar = 1000, retrain_buffer_length = 1000,
gradual_split = TRUE, theta = 0, wrapper = "DiceKriging",
gp_control = list(covtype = "matern3_2"),
split_direction_criterion = "max_spread_per_lengthscale", split_position_criterion = "mean",
shape_decay = "linear", use_empirical_error = TRUE,
use_reference_gp = FALSE, min_abs_y_err = 0, min_rel_y_err = 100 * .Machine$double.eps,
min_abs_node_pred_err = 0, min_rel_node_pred_err = 100 * .Machine$double.eps,
prob_min_theta = 0.01, add_buffer_in_prediction = FALSE, x_dim = ncol(X),
min_ranges = rep(0.0, ncol(X)), max_cond_num = NULL, max_points = Inf)
```

WrappedDiceKrigingGP *R6 class WrappedDiceKrigingGP*

Description

Contains the GP created by DiceKriging::km from the DiceKriging package

Public fields

- gp The DiceKriging GP object (DiceKriging::km in the DiceKriging manual)
- X_buffer Buffer matrix to collect x points until first GP can be trained
- y_buffer Buffer vector to collect y points until first GP can be trained
- y_var_buffer Buffer vector to collect variance of y points until first GP can be trained
- add_y_var Small additional variance used to keep the covariance matrix condition number under control
- n_points_train_limit Number of points needed before we can create the GP
- n_points The number of collected points belonging to this GP
- x_dim Dimensionality of input points
- gp_control A list of GP implementation-specific options, passed directly to the wrapped GP implementation
- init_covpars The initial covariance parameters when training the DiceKriging GP object in self@gp
- estimate_covpars If TRUE, the parameters are estimated by the package. Otherwise, the parameters from init_covpars are taken
- retrain_buffer_length Only retrain after this many new points have been added to the buffer
- retrain_buffer_counter Counter for the number of new points added since last retraining
- add_buffer_in_prediction If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated.

`X_shared` Matrix with x points that this GP shares with the GP in the sibling node

`y_shared` Vector of y points that this GP shares with the GP in the sibling node

`y_var_shared` Vector of y_var points that this GP shares with the GP in the sibling node

`n_shared_points` The number of own points shared with the GP in the sibling node

Methods

Public methods:

- `WrappedDiceKrigingGP$new()`
- `WrappedDiceKrigingGP$update_init_covpars()`
- `WrappedDiceKrigingGP$get_lengthscales()`
- `WrappedDiceKrigingGP$get_X_data()`
- `WrappedDiceKrigingGP$get_y_data()`
- `WrappedDiceKrigingGP$get_y_var_data()`
- `WrappedDiceKrigingGP$get_cov_mat()`
- `WrappedDiceKrigingGP$update_add_y_var()`
- `WrappedDiceKrigingGP$store_point()`
- `WrappedDiceKrigingGP$delete_buffers()`
- `WrappedDiceKrigingGP$train()`
- `WrappedDiceKrigingGP$predict()`
- `WrappedDiceKrigingGP$delete_gp()`
- `WrappedDiceKrigingGP$create_DiceKriging_gp()`
- `WrappedDiceKrigingGP$call_DiceKriging_predict()`
- `WrappedDiceKrigingGP$clone()`

Method `new()`**:** Create a new WrappedDiceKrigingGP object

Usage:

```
WrappedDiceKrigingGP$new(
  X,
  y,
  y_var,
  gp_control,
  init_covpars,
  retrain_buffer_length,
  add_buffer_in_prediction,
  estimate_covpars = TRUE,
  X_shared = NULL,
  y_shared = NULL,
  y_var_shared = NULL
)
```

Arguments:

`X` Input data matrix with x_dim columns and at maximum Nbar rows. Is used to create the first iteration of the local GP.

`y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored `y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

`gp_control` A list of GP implementation-specific options, passed directly to the wrapped GP implementation

`init_covpars` Initial covariance parameters of the local GP

`retrain_buffer_length` Only retrain when the number of buffer points or collected points exceeds this value

`add_buffer_in_prediction` If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated.

`estimate_covpars` If TRUE, the parameters are estimated by the package. Otherwise, the parameters from init_covpars are taken

`X_shared` Matrix with x points that this GP shares with the GP in the sibling node

`y_shared` Vector of y points that this GP shares with the GP in the sibling node

`y_var_shared` Vector of y_var points that this GP shares with the GP in the sibling node

Returns: A new WrappedDiceKrigingGP object. Besides the local GP, information on the shared points and those stored in the buffer are collected. For more information on the GP, consult the method DiceKriging::km in the `DiceKriging` package.

Method `update_init_covpars()`**:** Stores the initial covariance parameters (length-scales, standard deviation and trend coefficients) of the GP in the field `init_covpars`

Usage:

```
WrappedDiceKrigingGP$update_init_covpars()
```

Method `get_lengthscales()`**:** Retrieves the length-scales of the kernel of the local GP

Usage:

```
WrappedDiceKrigingGP$get_lengthscales()
```

Method `get_X_data()`**:** Retrieves the design matrix X

Usage:

```
WrappedDiceKrigingGP$get_X_data(include_shared = FALSE)
```

Arguments:

`include_shared` If TRUE, shared points between this GP and its sibling GP are included

Method `get_y_data()`**:** Retrieves the response

Usage:

```
WrappedDiceKrigingGP$get_y_data(include_shared = FALSE)
```

Arguments:

`include_shared` If TRUE, shared points between this GP and its sibling GP are included

Method `get_y_var_data()`**:** Retrieves the individual variances from the response

Usage:

```
WrappedDiceKrigingGP$get_y_var_data(include_shared = FALSE)
```

Arguments:

`include_shared` If TRUE, shared points between this GP and its sibling GP are included

Method `get_cov_mat()`**:** Retrieves the covariance matrix

Usage:

```
WrappedDiceKrigingGP$get_cov_mat()
```

Returns: the covariance matrix

Method `update_add_y_var()`**:** Method for updating add_y_var based on a bound for the covariance matrix condition number, based on this paper, Section 5.4

Usage:

```
WrappedDiceKrigingGP$update_add_y_var(max_cond_num)
```

Arguments:

`max_cond_num` Max allowed condition number

Method `store_point()`**:** Stores a new point into the respective buffer method

Usage:

```
WrappedDiceKrigingGP$store_point(
  x,
  y,
  y_var,
  shared = FALSE,
  remove_shared = TRUE
)
```

Arguments:

`x` Single input data point from the data stream; has to be a vector or row matrix with length equal to x_dim

`y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored

`y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

`shared` If TRUE, this point is shared between this GP and its sibling GP

`remove_shared` If TRUE, the last of the shared points is removed

Method `delete_buffers()`**:** Method for clearing the buffers

Usage:

```
WrappedDiceKrigingGP$delete_buffers()
```

Method `train()`**:** Method for (re)creating / (re)training the GP

Usage:

```
WrappedDiceKrigingGP$train(do_buffer_check = TRUE)
```

Arguments:

`do_buffer_check` If TRUE, only train the GP if the number of stored points is larger than retrain_buffer_length

Returns: TRUE if training was performed, otherwise FALSE

Method `predict()`**:** Method for prediction

Usage:

```
WrappedDiceKrigingGP$predict(x, return_std = TRUE)
```

Arguments:

`x` Single data point for which the predicted mean (and standard deviation) is computed; has to be a vector or row matrix with length equal to x_dim

`return_std` If TRUE, the standard error is returned in addition to the prediction

Returns: Prediction for input point x

Method `delete_gp()`**:** Method to delete the GP object in self$gp

Usage:

```
WrappedDiceKrigingGP$delete_gp()
```

Method `create_DiceKriging_gp()`**:** Method for calling the 'km' function in DiceKriging to create a GP object, stored in self$gp

Usage:

```
WrappedDiceKrigingGP$create_DiceKriging_gp(X, y, y_var)
```

Arguments:

`X` Input data matrix with x_dim columns and at maximum Nbar rows for the local GP.

`y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored

`y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

Returns: TRUE

Method `call_DiceKriging_predict()`**:** Method for calling the 'predict' function in DiceKriging

Usage:

```
WrappedDiceKrigingGP$call_DiceKriging_predict(x, use_gp = NULL)
```

Arguments:

`x` Single data point for which the predicted mean (and standard deviation) is computed; has to be a vector with length equal to x_dim

`use_gp` optional user-defined GP which is evaluated instead of the local GP

Returns: The predictions for x from the specified GP, by default the local GP

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
WrappedDiceKrigingGP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`WrappedGP` *R6 class WrappedGP*

Description

Contains the GP created by a user-defined GP package

Details

This is effectively a dummy wrapper based on the wrapper for the mlegp package (see WrappedmlegpGP). It contains a basic implementation of the wrapper. The vignette offers a tutorial on how to change this wrapper for the new GP package.

Public fields

`gp` The mlegp GP object (mlegp::mlegp in the `mlegp` manual)

`X_buffer` Buffer matrix to collect x points until first GP can be trained

`y_buffer` Buffer vector to collect y points until first GP can be trained

`y_var_buffer` Buffer vector to collect variance of y points until first GP can be trained

`add_y_var` Small additional variance used to keep the covariance matrix condition number under control

`n_points_train_limit` Number of points needed before we can create the GP

`n_points` The number of collected points belonging to this GP

`x_dim` Dimensionality of input points

`gp_control` A list of GP implementation-specific options, passed directly to the wrapped GP implementation

`init_covpars` The initial covariance parameters when training the mlegp GP object in self@gp

`estimate_covpars` If TRUE, the parameters are estimated by the package. Otherwise, the parameters from init_covpars are taken

`retrain_buffer_length` Only retrain after this many new points have been added to the buffer

`retrain_buffer_counter` Counter for the number of new points added since last retraining

`add_buffer_in_prediction` If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated.

`X_shared` Matrix with x points that this GP shares with the GP in the sibling node

`y_shared` Vector of y points that this GP shares with the GP in the sibling node

`y_var_shared` Vector of y_var points that this GP shares with the GP in the sibling node

`n_shared_points` The number of own points shared with the GP in the sibling node

Methods

Public methods:

- `WrappedGP$new()`
- `WrappedGP$update_init_covpars()`
- `WrappedGP$get_lengthscales()`
- `WrappedGP$get_X_data()`
- `WrappedGP$get_y_data()`
- `WrappedGP$get_y_var_data()`
- `WrappedGP$get_cov_mat()`
- `WrappedGP$update_add_y_var()`
- `WrappedGP$store_point()`
- `WrappedGP$delete_buffers()`
- `WrappedGP$delete_gp()`
- `WrappedGP$call_create_gp()`
- `WrappedGP$call_predict()`
- `WrappedGP$train()`
- `WrappedGP$predict()`
- `WrappedGP$clone()`

Method `new()`**:** Create a new WrappedmlegpGP object

Usage:

```
WrappedGP$new(
  X,
  y,
  y_var,
  gp_control,
  init_covpars,
  retrain_buffer_length,
  add_buffer_in_prediction,
  estimate_covpars = TRUE,
  X_shared = NULL,
  y_shared = NULL,
  y_var_shared = NULL
)
```

Arguments:

`X` Input data matrix with x_dim columns and at maximum Nbar rows. Is used to create the first iteration of the local GP.

`y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored

`y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

`gp_control` A list of GP implementation-specific options, passed directly to the wrapped GP implementation

`init_covpars` Initial covariance parameters of the local GP `retrain_buffer_length` Only retrain when the number of buffer points or collected points exceeds this value

`add_buffer_in_prediction` If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated.

`estimate_covpars` If TRUE, the parameters are estimated by the package. Otherwise, the parameters from init_covpars are taken

`X_shared` Matrix with x points that this GP shares with the GP in the sibling node

`y_shared` Vector of y points that this GP shares with the GP in the sibling node

`y_var_shared` Vector of y_var points that this GP shares with the GP in the sibling node

Returns: A new WrappedGP object. Besides the local GP, information on the shared points and those stored in the buffer are collected. For more information on the GP, consult the respective met in the GP package.

Method `update_init_covpars()`**:** Stores the initial covariance parameters (length-scales, standard deviation and trend coefficients) of the GP in the field `init_covpars`

Usage:

```
WrappedGP$update_init_covpars()
```

Method `get_lengthscales()`**:** Retrieves the length-scales of the kernel of the local GP

Usage:

```
WrappedGP$get_lengthscales()
```

Method `get_X_data()`**:** Retrieves the design matrix X

Usage:

```
WrappedGP$get_X_data(include_shared = FALSE)
```

Arguments:

`include_shared` If TRUE, shared points between this GP and its sibling GP are included

Method `get_y_data()`**:** Retrieves the response

Usage:

```
WrappedGP$get_y_data(include_shared = FALSE)
```

Arguments:

`include_shared` If TRUE, shared points between this GP and its sibling GP are included

Method `get_y_var_data()`**:** Retrieves the individual variances from the response

Usage:

```
WrappedGP$get_y_var_data(include_shared = FALSE)
```

Arguments:

`include_shared` If TRUE, shared points between this GP and its sibling GP are included

Method `get_cov_mat()`**:** Retrieves the covariance matrix

Usage:

```
WrappedGP$get_cov_mat()
```

Returns: the covariance matrix

Method `update_add_y_var()`**:** Method for updating add_y_var based on a bound for the covariance matrix condition number, based on this paper, Section 5.4

Usage:

```
WrappedGP$update_add_y_var(max_cond_num)
```

Arguments:

`max_cond_num` Max allowed condition number

Method `store_point()`**:** Stores a new point into the respective buffer method

Usage:

```
WrappedGP$store_point(x, y, y_var, shared = FALSE, remove_shared = TRUE)
```

Arguments:

`x` Single input data point from the data stream; has to be a vector or row matrix with length equal to x_dim

`y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored

`y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

`shared` If TRUE, this point is shared between this GP and its sibling GP

`remove_shared` If TRUE, the last of the shared points is removed

Method `delete_buffers()`**:** Method for clearing the buffers

Usage:

```
WrappedGP$delete_buffers()
```

Method `delete_gp()`**:** Method to delete the GP object in self$gp

Usage:

```
WrappedGP$delete_gp()
```

Method `call_create_gp()`**:** Method for calling the 'mlegp' function in mlegp to create a GP object, stored in self$gp

Usage:

```
WrappedGP$call_create_gp(X, y, y_var)
```

Arguments:

`X` Input data matrix with x_dim columns and at maximum Nbar rows for the local GP.

`y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored

`y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

Returns: TRUE

Method `call_predict()`**:** Method for calling the 'predict' function in mlegp

Usage:

```
WrappedGP$call_predict(x, use_gp = NULL)
```

Arguments:

`x` Single data point for which the predicted mean (and standard deviation) is computed; has to be a vector with length equal to x_dim

`use_gp` Optional user-defined GP which is evaluated instead of the local GP

Returns: The predictions for x from the specified GP, by default the local GP. The output needs to be a list with fields mean and sd for the prediction and prediction error, respectively.

Method `train()`**:** Method for (re)creating / (re)training the GP

Usage:

```
WrappedGP$train(do_buffer_check = TRUE)
```

Arguments:

`do_buffer_check` If TRUE, only train the GP if the number of stored points is larger than retrain_buffer_length

Returns: TRUE if training was performed, otherwise FALSE

Method `predict()`**:** Method for prediction

Usage:

```
WrappedGP$predict(x, return_std = TRUE)
```

Arguments:

`x` Single data point for which the predicted mean (and standard deviation) is computed; has to be a vector or row matrix with length equal to x_dim

`return_std` If TRUE, the standard error is returned in addition to the prediction

Returns: Prediction for input point x

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
WrappedGP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`WrappedmlegpGP` *R6 class WrappedmlegpGP*

Description

Contains the GP created by mlegp::mlegp from the `mlegp` package

Details

This package is by default not able to include individual uncertainties for input points. For this reason, all fields related to `y_var` are not used when updating the GP. No covariance kernel can be specified either. This implementation also assumes a vector for `y` (and not a matrix with multiple columns). Moreover, since no parameters can be specified for the GP, we will only update the GP parameters due to internal dependencies, but not use `init_covpars`.

Public fields

`gp` The mlegp GP object (mlegp::mlegp in the `mlegp` manual)

`X_buffer` Buffer matrix to collect x points until first GP can be trained

`y_buffer` Buffer vector to collect y points until first GP can be trained

`y_var_buffer` Buffer vector to collect variance of y points until first GP can be trained

`add_y_var` Small additional variance used to keep the covariance matrix condition number under control

`n_points_train_limit` Number of points needed before we can create the GP

`n_points` The number of collected points belonging to this GP

`x_dim` Dimensionality of input points

`gp_control` A list of GP implementation-specific options, passed directly to the wrapped GP implementation

`init_covpars` The initial covariance parameters when training the mlegp GP object in self@gp

`estimate_covpars` If TRUE, the parameters are estimated by the package. Otherwise, the parameters from init_covpars are taken

`retrain_buffer_length` Only retrain after this many new points have been added to the buffer

`retrain_buffer_counter` Counter for the number of new points added since last retraining

`add_buffer_in_prediction` If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated.

`X_shared` Matrix with x points that this GP shares with the GP in the sibling node

`y_shared` Vector of y points that this GP shares with the GP in the sibling node

`y_var_shared` Vector of y_var points that this GP shares with the GP in the sibling node

`n_shared_points` The number of own points shared with the GP in the sibling node

Methods

Public methods:

- `WrappedmlegpGP$new()`
- `WrappedmlegpGP$update_init_covpars()`
- `WrappedmlegpGP$get_lengthscales()`
- `WrappedmlegpGP$get_X_data()`
- `WrappedmlegpGP$get_y_data()`
- `WrappedmlegpGP$get_y_var_data()`
- `WrappedmlegpGP$get_cov_mat()`
- `WrappedmlegpGP$update_add_y_var()`
- `WrappedmlegpGP$store_point()`
- `WrappedmlegpGP$delete_buffers()`
- `WrappedmlegpGP$train()`
- `WrappedmlegpGP$predict()`
- `WrappedmlegpGP$delete_gp()`

- `WrappedmlegpGP$create_mlegp_gp()`
- `WrappedmlegpGP$call_mlegp_predict()`
- `WrappedmlegpGP$clone()`

Method `new()`**:** Create a new WrappedmlegpGP object

Usage:

```
WrappedmlegpGP$new(
  X,
  y,
  y_var,
  gp_control,
  init_covpars,
  retrain_buffer_length,
  add_buffer_in_prediction,
  estimate_covpars = TRUE,
  X_shared = NULL,
  y_shared = NULL,
  y_var_shared = NULL
)
```

Arguments:

`X` Input data matrix with x_dim columns and at maximum Nbar rows. Is used to create the first iteration of the local GP.

`y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored

`y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

`gp_control` A list of GP implementation-specific options, passed directly to the wrapped GP implementation

`init_covpars` Initial covariance parameters of the local GP

`retrain_buffer_length` Only retrain when the number of buffer points or collected points exceeds this value

`add_buffer_in_prediction` If TRUE, points in the data buffers are added to the GP before prediction. They are added into a temporarily created GP which contains the not yet included points. The GP in the node is not yet updated.

`estimate_covpars` If TRUE, the parameters are estimated by the package. Otherwise, the parameters from init_covpars are taken

`X_shared` Matrix with x points that this GP shares with the GP in the sibling node

`y_shared` Vector of y points that this GP shares with the GP in the sibling node

`y_var_shared` Vector of y_var points that this GP shares with the GP in the sibling node

Returns: A new WrappedmlegpGP object. Besides the local GP, information on the shared points and those stored in the buffer are collected. For more information on the GP, consult the method mlegp::mlegp in the `mlegp` package.

Method `update_init_covpars()`**:** Stores the initial covariance parameters (length-scales, standard deviation and trend coefficients) of the GP in the field `init_covpars`

Usage:

```
WrappedmlegpGP$update_init_covpars()
```

Method `get_lengthscales()`**:** Retrieves the length-scales of the kernel of the local GP

Usage:

```
WrappedmlegpGP$get_lengthscales()
```

Method `get_X_data()`**:** Retrieves the design matrix X

Usage:

```
WrappedmlegpGP$get_X_data(include_shared = FALSE)
```

Arguments:

`include_shared` If TRUE, shared points between this GP and its sibling GP are included

Method `get_y_data()`**:** Retrieves the response

Usage:

```
WrappedmlegpGP$get_y_data(include_shared = FALSE)
```

Arguments:

`include_shared` If TRUE, shared points between this GP and its sibling GP are included

Method `get_y_var_data()`**:** Retrieves the individual variances from the response

Usage:

```
WrappedmlegpGP$get_y_var_data(include_shared = FALSE)
```

Arguments:

`include_shared` If TRUE, shared points between this GP and its sibling GP are included

Method `get_cov_mat()`**:** Retrieves the covariance matrix

Usage:

```
WrappedmlegpGP$get_cov_mat()
```

Returns: the covariance matrix

Method `update_add_y_var()`**:** Method for updating add_y_var based on a bound for the covariance matrix condition number, based on this paper, Section 5.4

Usage:

```
WrappedmlegpGP$update_add_y_var(max_cond_num)
```

Arguments:

`max_cond_num` Max allowed condition number

Method `store_point()`**:** Stores a new point into the respective buffer method

Usage:

```
WrappedmlegpGP$store_point(x, y, y_var, shared = FALSE, remove_shared = TRUE)
```

Arguments:

`x` Single input data point from the data stream; has to be a vector or row matrix with length equal to x_dim `y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored

`y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

`shared` If TRUE, this point is shared between this GP and its sibling GP

`remove_shared` If TRUE, the last of the shared points is removed

Method `delete_buffers()`**:** Method for clearing the buffers

Usage:

```
WrappedmlegpGP$delete_buffers()
```

Method `train()`**:** Method for (re)creating / (re)training the GP

Usage:

```
WrappedmlegpGP$train(do_buffer_check = TRUE)
```

Arguments:

`do_buffer_check` If TRUE, only train the GP if the number of stored points is larger than retrain_buffer_length

Returns: TRUE if training was performed, otherwise FALSE

Method `predict()`**:** Method for prediction

Usage:

```
WrappedmlegpGP$predict(x, return_std = TRUE)
```

Arguments:

`x` Single data point for which the predicted mean (and standard deviation) is computed; has to be a vector or row matrix with length equal to x_dim

`return_std` If TRUE, the standard error is returned in addition to the prediction

Returns: Prediction for input point x

Method `delete_gp()`**:** Method to delete the GP object in self$gp

Usage:

```
WrappedmlegpGP$delete_gp()
```

Method `create_mlegp_gp()`**:** Method for calling the 'mlegp' function in mlegp to create a GP object, stored in self$gp

Usage:

```
WrappedmlegpGP$create_mlegp_gp(X, y, y_var)
```

Arguments:

`X` Input data matrix with x_dim columns and at maximum Nbar rows for the local GP.

`y` Value of target variable at input point x; has to be a one-dimensional matrix or a vector; any further columns will be ignored

`y_var` Variance of the target variable; has to be a one-dimensional matrix or vector

Returns: TRUE

Method `call_mlegp_predict()`**:** Method for calling the 'predict' function in mlegp

Usage:

```
WrappedmlegpGP$call_mlegp_predict(x, use_gp = NULL)
```

Arguments:

`x` Single data point for which the predicted mean (and standard deviation) is computed; has to be a vector with length equal to x_dim

`use_gp` Optional user-defined GP which is evaluated instead of the local GP

Returns: The predictions for x from the specified GP, by default the local GP. The output needs to be a list with fields mean and sd for the prediction and prediction error, respectively.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
WrappedmlegpGP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index

CreateWrappedGP, 2

DiceKriging::km, *15*, *17*

GPNode, 3, *8*, *12*
GPTree, *3*, *5*, *6*, 7
GPTree(), *7*

hash, *12*

mlegp::mlegp, *20*, *24–26*

WrappedDiceKrigingGP, *3*, *8*, 15
WrappedGP, *3*, *8*, 20
WrappedmlegpGP, *3*, *8*, *20*, 24