Package ‘RGF’

May 7, 2021

Type Package
Title Regularized Greedy Forest
Version 1.0.8
Date 2021-05-07

BugReports https://github.com/RGF-team/rgf/issues

URL https://github.com/RGF-team/rgf/tree/master/R-package

Description

License MIT + file LICENSE

SystemRequirements Python (>= 3.6), rgf_python, scikit-learn (>= 0.18.0), scipy, numpy. Detailed installation instructions for each operating system can be found in the README file.

Depends R(>= 3.2.0)
Imports reticulate, R6, Matrix
Suggests testthat, covr, knitr, rmarkdown

Encoding UTF-8
RoxygenNote 7.1.1
VignetteBuilder knitr

NeedsCompilation no

Author Lampros Mouselimis [aut, cre] (<https://orcid.org/0000-0002-8024-1546>), Ryosuke Fukatani [cph] (Author of the python wrapper of the 'Regularized Greedy Forest' machine learning algorithm), Nikita Titov [cph] (Author of the python wrapper of the 'Regularized Greedy Forest' machine learning algorithm), Tong Zhang [cph] (Author of the 'Regularized Greedy Forest' and of the Multi-core implementation of Regularized Greedy Forest machine learning algorithm), Rie Johnson [cph] (Author of the 'Regularized Greedy Forest' machine learning algorithm)
FastRGF_Classifier

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>
Repository CRAN
Date/Publication 2021-05-07 16:10:07 UTC

R topics documented:

- FastRGF_Classifier ........................................ 2
- FastRGF_Regressor .......................................... 5
- mat_2scipy_sparse ........................................ 8
- RGF_Classifier ............................................. 9
- RGFCleanup_temp_files ................................. 13
- RGF_Regressor .............................................. 14
- TO_scipy_sparse ........................................... 17

Index

---

FastRGF_Classifier A Fast Regularized Greedy Forest classifier

Description

A Fast Regularized Greedy Forest classifier
A Fast Regularized Greedy Forest classifier

Usage

```r
# init <- FastRGF_Classifier$new(n_estimators = 500, max_depth = 6,
# max_leaf = 50, tree_gain_ratio = 1.0,
# min_samples_leaf = 5, loss = "LS", l1 = 1.0,
# l2 = 1000.0, opt_algorithm = "rgf",
# learning_rate = 0.001, max_bin = NULL,
# min_child_weight = 5.0, data_l2 = 2.0,
# sparse_max_features = 80000,
# sparse_min_occurences = 5,
# calc_prob = "sigmoid", n_jobs = 1,
# verbose = 0)
```

Details

the `fit` function builds a classifier from the training set (x, y).
the `predict` function predicts the class for x.
the `predict_proba` function predicts class probabilities for x.
the `cleanup` function removes tempfiles used by this model. See the issue https://github.com/RGF-team/rgf/issues/75, which explains in which cases the `cleanup` function applies.
the `get_params` function returns the parameters of the model.
the `score` function returns the mean accuracy on the given test data and labels.
Methods

```r
FastRGF_Classifier$new(n_estimators = 500, max_depth = 6, max_leaf = 50, tree_gain_ratio = 1.0, min_samples_leaf = 5, ...

--------------
fit(x, y, sample_weight = NULL)
--------------
predict(x)
--------------
predict_proba(x)
--------------
cleanup()
--------------
get_params(deep = TRUE)
--------------
score(x, y, sample_weight = NULL)
--------------
```

Super class

`RGF::Internal_class` -> `FastRGF_Classifier`

Methods

Public methods:
- `FastRGF_Classifier$new`
- `FastRGF_Classifier$clone`

Method `new()`:

Usage:
```r
FastRGF_Classifier$new(
  n_estimators = 500,
  max_depth = 6,
  max_leaf = 50,
  tree_gain_ratio = 1,
  min_samples_leaf = 5,
  loss = "LS",
  l1 = 1,
  l2 = 1000,
  opt_algorithm = "rgf",
  learning_rate = 0.001,
  max_bin = NULL,
  min_child_weight = 5,
  data_l2 = 2,
  sparse_max_features = 80000,
```
FastRGF_Classifier

spars_min_occurences = 5,
calc_prob = "sigmoid",
_n_jobs = 1,
verbose = 0
)

Arguments:
n_estimators an integer. The number of trees in the forest (Original name: forest.ntrees.)
max_depth an integer. Maximum tree depth (Original name: dtree.max_level.)
max_leaf an integer. Maximum number of leaf nodes in best-first search (Original name:
dtree.max_nodes.)
tree_gain_ratio a float. New tree is created when leaf-nodes gain < this value * estimated
    gain of creating new tree (Original name: dtree.new_tree_gain_ratio.)
min_samples_leaf an integer or float. Minimum number of training data points in each leaf
    node. If an integer, then consider min_samples_leaf as the minimum number. If a float,
    then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the
    minimum number of samples for each node (Original name: dtree.min_sample.)
loss a character string. One of "LS" (Least squares loss), "MODLS" (Modified least squares
    loss) or "LOGISTIC" (Logistic loss) (Original name: dtree.loss.)
l1 a float. Used to control the degree of L1 regularization (Original name: dtree.lamL1.)
l2 a float. Used to control the degree of L2 regularization (Original name: dtree.lamL2.)
opt_algorithm a character string. Either "rgf" or "epsilon-greedy". Optimization method for
    training forest (Original name: forest.opt.)
learning_rate a float. Step size of epsilon-greedy boosting. Meant for being used with
    opt_algorithm = "epsilon-greedy" (Original name: forest.stepsize.)
max_bin an integer or NULL. Maximum number of discretized values (bins). If NULL, 65000
    is used for dense data and 200 for sparse data (Original name: discretize.(sparse/dense).max_buckets.)
min_child_weight a float. Minimum sum of data weights for each discretized value (bin)
    (Original name: discretize.(sparse/dense).min_bucket_weights.)
data_l2 a float. Used to control the degree of L2 regularization for discretization (Original
    name: discretize.(sparse/dense).lamL2.)
sparse_max_features an integer. Maximum number of selected features. Meant for being
    used with sparse data (Original name: discretize.sparse.max_features.)
sparse_min_occurences an integer. Minimum number of occurrences for a feature to be se-
    lected. Meant for being used with sparse data (Original name: discretize.sparse.min_occurrences.)
calc_prob a character string. Either "sigmoid" or "softmax". Method of probability calculation
n_jobs an integer. The number of jobs to run in parallel for both fit and predict. If -1, all CPUs
    are used. If -2, all CPUs but one are used. If < -1, (n_cpus + 1 + n_jobs) are used (Original
    name: set.nthreads.)
verbose an integer. Controls the verbosity of the tree building process (Original name: set.verbose.)

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

Usage:
FastRGF_Classifier$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
FastRGF_Regressor

References


Examples

```r
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("rgf.sklearn")) {
      library(RGF)
      set.seed(1)
      x = matrix(runif(100000), nrow = 100, ncol = 1000)
      y = sample(1:2, 100, replace = TRUE)
      fast_RGF_class = FastRGF_Classifier$new(max_leaf = 50)
      fast_RGF_class$fit(x, y)
      preds = fast_RGF_class$predict_proba(x)
    }
  }
}, silent = TRUE)
```

FastRGF_Regressor  A Fast Regularized Greedy Forest regressor

Description

A Fast Regularized Greedy Forest regressor

Usage

```r
# init <- FastRGF_Regressor$new(n_estimators = 500, max_depth = 6,
# max_leaf = 50, tree_gain_ratio = 1.0,
# min_samples_leaf = 5, l1 = 1.0,
# l2 = 1000.0, opt_algorithm = "rgf",
# learning_rate = 0.001, max_bin = NULL,
# min_child_weight = 5.0, data_l2 = 2.0,
# sparse_max_features = 80000,
# sparse_min_occurences = 5,
# n_jobs = 1, verbose = 0)
```
Details

the `fit` function builds a regressor from the training set (x, y).
the `predict` function predicts the regression target for x.
the `cleanup` function removes tempfiles used by this model. See the issue https://github.com/RGF-team/rgf/issues/75, which explains in which cases the `cleanup` function applies.
the `get_params` function returns the parameters of the model.
the `score` function returns the coefficient of determination (R^2) for the predictions.

Methods

```r
FastRGF_Regressor$new(n_estimators = 500, max_depth = 6, max_leaf = 50, tree_gain_ratio = 1.0, min_samples_leaf = 5, l1 = ..., min_child_weight = 5.0, data_l2 = 2.0, sparse_max_features = 80000, sparse_min_occurences = 5, n_jobs = 1, verbose = 0)
```

---

```r
fit(x, y, sample_weight = NULL)
```

---

```r
predict(x)
```

---

```r
cleanup()
```

---

```r
get_params(deep = TRUE)
```

---

```r
score(x, y, sample_weight = NULL)
```

---

Super class

`RGF::Internal_class` -> `FastRGF_Regressor`

Methods

Public methods:

- `FastRGF_Regressor$new()`
- `FastRGF_Regressor$clone()`

Method `new()`:

Usage:

```r
FastRGF_Regressor$new(n_estimators = 500, max_depth = 6, max_leaf = 50, tree_gain_ratio = 1.0, min_samples_leaf = 5, l1 = 1,
```
l2 = 1000,
opt_algorithm = "rgf",
learning_rate = 0.001,
max_bin = NULL,
min_child_weight = 5,
data_l2 = 2,
sparse_max_features = 80000,
sparse_min_occurences = 5,
n_jobs = 1,
verbose = 0
)

Arguments:
n_estimators an integer. The number of trees in the forest (Original name: forest.ntrees.)
max_depth an integer. Maximum tree depth (Original name: dtree.max_level.)
max_leaf an integer. Maximum number of leaf nodes in best-first search (Original name: dtree.max_nodes.)
tree_gain_ratio a float. New tree is created when leaf-nodes gain < this value * estimated
gain of creating new tree (Original name: dtree.new_tree_gain_ratio.)
min_samples_leaf an integer or float. Minimum number of training data points in each leaf
node. If an integer, then consider min_samples_leaf as the minimum number. If a float,
then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the
minimum number of samples for each node (Original name: dtree.min_sample.)
11 a float. Used to control the degree of L1 regularization (Original name: dtree.lambda1.)
12 a float. Used to control the degree of L2 regularization (Original name: dtree.lambda2.)
opt_algorithm a character string. Either "rgf" or "epsilon-greedy". Optimization method for
training forest (Original name: forest.opt.)
learning_rate a float. Step size of epsilon-greedy boosting. Meant for being used with
opt_algorithm = "epsilon-greedy" (Original name: forest.stepsize.)
max_bin an integer or NULL. Maximum number of discretized values (bins). If NULL, 65000
is used for dense data and 200 for sparse data (Original name: discretize.(sparse/dense).max_buckets.)
min_child_weight a float. Minimum sum of data weights for each discretized value (bin)
(Original name: discretize.(sparse/dense).min_bucket_weights.)
data_l2 a float. Used to control the degree of L2 regularization for discretization (Original
name: discretize.(sparse/dense).lambda2.)
sparse_max_features an integer. Maximum number of selected features. Meant for being
used with sparse data (Original name: discretize.sparse.max_features.)
sparse_min_occurences an integer. Minimum number of occurrences for a feature to be se-
lected. Meant for being used with sparse data (Original name: discretize.sparse.min_occurences.)
n_jobs an integer. The number of jobs to run in parallel for both fit and predict. If -1, all CPUs
are used. If -2, all CPUs but one are used. If < -1, (n_CPUS + 1 + n_jobs) are used (Original
name: set.nthreads.)
verbose an integer. Controls the verbosity of the tree building process (Original name: set.verbose.)

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

Usage:
FastRGF_Regressor$clone(deep = FALSE)

Arguments:

depth Whether to make a deep clone.

References


Examples

```r
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("rgf.sklearn")) {
      library(RGF)
      set.seed(1)
      x = matrix(runif(100000), nrow = 100, ncol = 1000)
      y = runif(100)
      fast_RGF_regr = FastRGF_Regressor$new(max_leaf = 50)
      fast_RGF_regr$fit(x, y)
      preds = fast_RGF_regr$predict(x)
    }
  }
}, silent = TRUE)
```

mat_2scipy_sparse

conversion of an R matrix to a scipy sparse matrix

Description

conversion of an R matrix to a scipy sparse matrix

Usage

```r
mat_2scipy_sparse(x, format = "sparse_row_matrix")
```

Arguments

- `x` a data matrix
- `format` a character string. Either "sparse_row_matrix" or "sparse_column_matrix"
Details

This function allows the user to convert an R matrix to a scipy sparse matrix. This is useful because the Regularized Greedy Forest algorithm accepts only python sparse matrices as input.

References

https://docs.scipy.org/doc/scipy/reference/sparse.html

Examples

```r
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("scipy")) {
      library(RGF)
      set.seed(1)
      x = matrix(runif(1000), nrow = 100, ncol = 10)
      res = mat_2scipy_sparse(x)
      print(dim(x))
      print(res$shape)
    }
  }
}, silent = TRUE)
```

RGF_Classifier

Regularized Greedy Forest classifier

Description

Regularized Greedy Forest classifier

Usage

```r
# init <- RGF_Classifier$new(max_leaf = 1000, test_interval = 100,
#                            algorithm = "RGF", loss = "Log", reg_depth = 1.0,
#                            l2 = 0.1, sl2 = NULL, normalize = FALSE,
#                            min_samples_leaf = 10, n_iter = NULL,
#                            n_tree_search = 1, opt_interval = 100,
#                            learning_rate = 0.5, calc_prob = "sigmoid",
#                            n_jobs = 1, memory_policy = "generous",
#                            verbose = 0, init_model = NULL)
```
Details

the **fit** function builds a classifier from the training set \((x, y)\).

the **predict** function predicts the class for \(x\).

the **predict_proba** function predicts class probabilities for \(x\).

the **cleanup** function removes tempfiles used by this model. See the issue https://github.com/GRF-{}-team/rgf/issues/75, which explains in which cases the **cleanup** function applies.

the **get_params** function returns the parameters of the model.

the **score** function returns the mean accuracy on the given test data and labels.

the **feature_importances** function returns the feature importances for the data.

the **dump_model** function currently prints information about the fitted model in the console.

the **save_model** function saves a model to a file from which training can do warm-start in the future.

Methods

**RGF_Classifier$new**(max_leaf = 1000, test_interval = 100, algorithm = "RGF", loss = "Log", reg_depth = 1.0, l2 = 0.1, sl2 = 0.1, test_split = 0.1, learning_rate = 0.5, n_jobs = 1, memory_policy = "generous", verbose = 0, init_model = NULL)

--------------

**fit**(*x*, *y*, sample_weight = NULL)

--------------

**predict**(*x*)

--------------

**predict_proba**(*x*)

--------------

**cleanup**()

--------------

**get_params**(deep = TRUE)

--------------

**score**(*x*, *y*, sample_weight = NULL)

--------------

**feature_importances**()

--------------

**dump_model**()

--------------

**save_model**(filename)

--------------

Super class

**RGF::Internal_class** -> **RGF_Classifier**
Methods

Public methods:

- RGF_Classifier$new()
- RGF_Classifier$clone()

Method new():

Usage:

```r
RGF_Classifier$new(
  max_leaf = 1000,
  test_interval = 100,
  algorithm = "RGF",
  loss = "Log",
  reg_depth = 1,
  l2 = 0.1,
  sl2 = NULL,
  normalize = FALSE,
  min_samples_leaf = 10,
  n_iter = NULL,
  n_tree_search = 1,
  opt_interval = 100,
  learning_rate = 0.5,
  calc_prob = "sigmoid",
  n_jobs = 1,
  memory_policy = "generous",
  verbose = 0,
  init_model = NULL
)
```

Arguments:

- `max_leaf` an integer. Training will be terminated when the number of leaf nodes in the forest reaches this value.
- `test_interval` an integer. Test interval in terms of the number of leaf nodes.
- `algorithm` a character string specifying the Regularization algorithm. One of "RGF" (RGF with L2 regularization on leaf-only models), "RGF_Opt" (RGF with min-penalty regularization) or "RGF_Sib" (RGF with min-penalty regularization with the sum-to-zero sibling constraints).
- `loss` a character string specifying the Loss function. One of "LS" (Square loss), "Expo" (Exponential loss) or "Log" (Logistic loss).
- `reg_depth` a float. Must be no smaller than 1.0. Meant for being used with the algorithm RGF Opt or RGF Sib. A larger value penalizes deeper nodes more severely.
- `l2` a float. Used to control the degree of L2 regularization.
- `sl2` a float or NULL. Override L2 regularization parameter l2 for the process of growing the forest. That is, if specified, the weight correction process uses l2 and the forest growing process uses sl2. If NULL, no override takes place and l2 is used throughout training.
- `normalize` a boolean. If True, training targets are normalized so that the average becomes zero.
- `min_samples_leaf` an integer or a float. Minimum number of training data points in each leaf node. If an integer, then consider min_samples_leaf as the minimum number. If a float,
then \texttt{min_samples_leaf} is a percentage and \texttt{ceil(min_samples_leaf * n_samples)} are the minimum number of samples for each node.

\texttt{n_iter} an integer or NULL. The number of iterations of coordinate descent to optimize weights. If NULL, 10 is used for loss = "LS" and 5 for loss = "Exp" or "Log".

\texttt{n_tree_search} an integer. The number of trees to be searched for the nodes to split. The most recently grown trees are searched first.

\texttt{opt_interval} an integer. Weight optimization interval in terms of the number of leaf nodes. For example, by default, weight optimization is performed every time approximately 100 leaf nodes are newly added to the forest.

\texttt{learning_rate} a float. Step size of Newton updates used in coordinate descent to optimize weights.

\texttt{calc_prob} a character string. One of "sigmoid" or "softmax". Method of probability calculation.

\texttt{n_jobs} an integer. The number of jobs (threads) to use for the computation. The substantial number of the jobs depends on \texttt{classes} (The number of classes when \texttt{fit} is performed). If \texttt{classes} = 2, the substantial max number of the jobs is 1. If \texttt{classes} > 2, the substantial max number of the jobs is the same as \texttt{classes}. If \texttt{n_jobs} = 1, no parallel computing code is used at all regardless of \texttt{classes}. If \texttt{n_jobs} = -1 and \texttt{classes} >= number of CPU, all CPUs are used. For \texttt{n_jobs} = -2, all CPUs but one are used. For \texttt{n_jobs} below -1, (\texttt{n_cpus} + 1 + \texttt{n_jobs}) are used.

\texttt{memory_policy} a character string. One of "conservative" (it uses less memory at the expense of longer runtime. Try only when with default value it uses too much memory) or "generous" (it runs faster using more memory by keeping the sorted orders of the features on memory for reuse). Memory using policy.

\texttt{verbose} an integer. Controls the verbosity of the tree building process.

\texttt{init_model} either NULL or a character string, optional (default=NULL). Filename of a previously saved model from which training should do warm-start. If model has been saved into multiple files, do not include numerical suffixes in the filename. \texttt{NOTE}: Make sure you haven’t forgotten to increase the value of the max_leaf parameter regarding to the specified warm-start model because warm-start model trees are counted in the overall number of trees.

**Method** \texttt{clone()}: The objects of this class are cloneable with this method.

**Usage:**

\texttt{RGF_Classifier$clone(deep = FALSE)}

**Arguments:**

\texttt{deep} Whether to make a deep clone.

**References**

https://github.com/RGF-team/rgf/tree/master/python-package, Rie Johnson and Tong Zhang, Learning Nonlinear Functions Using Regularized Greedy Forest

**Examples**

```r
try({

```
if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("rgf.sklearn")) {
        library(RGF)
        set.seed(1)
        x = matrix(runif(1000), nrow = 100, ncol = 10)
        y = sample(1:2, 100, replace = TRUE)
        RGF_class = RGF_Classifier$new(max_leaf = 50)
        RGF_class$fit(x, y)
        preds = RGF_class$predict_proba(x)
    }
}, silent = TRUE)

RGF_cleanup_temp_files

Delete all temporary files of the created RGF estimators

Description

Delete all temporary files of the created RGF estimators

Usage

RGF_cleanup_temp_files()

Details

This function deletes all temporary files of the created RGF estimators. See the issue https://github.com/RGF-team/rgf/issues/75 for more details.

References

https://github.com/RGF-team/rgf/tree/master/python-package

Examples

```r
## Not run:
library(RGF)
RGF_cleanup_temp_files()
## End(Not run)
```
RGF_Regressor

Regularized Greedy Forest regressor

Description

Regularized Greedy Forest regressor

Usage

```r
# init <- RGF_Regressor$new(max_leaf = 500, test_interval = 100, 
#   algorithm = "RGF", loss = "LS", reg_depth = 1.0, 
#   l2 = 0.1, sl2 = NULL, normalize = TRUE, 
#   min_samples_leaf = 10, n_iter = NULL, 
#   n_tree_search = 1, opt_interval = 100, 
#   learning_rate = 0.5, memory_policy = "generous", 
#   verbose = 0, init_model = NULL)
```

Details

the `fit` function builds a regressor from the training set \((x, y)\).
the `predict` function predicts the regression target for \(x\).
the `cleanup` function removes tempfiles used by this model. See the issue https://github.com/rgf-team/rgf/issues/75, which explains in which cases the `cleanup` function applies.
the `get_params` function returns the parameters of the model.
the `score` function returns the coefficient of determination \((R^2)\) for the predictions.
the `feature_importances` function returns the feature importances for the data.
the `dump_model` function currently prints information about the fitted model in the console
the `save_model` function saves a model to a file from which training can do warm-start in the future.

Methods

```r
RGF_Regressor$new(max_leaf = 500, test_interval = 100, algorithm = "RGF", loss = "LS", reg_depth = 1.0, l2 = 0.1, sl2 = NULL, normalize = TRUE, min_samples_leaf = 10, n_tree_search = 1, opt_interval = 100, learning_rate = 0.5, memory_policy = "generous", verbose = 0, init_model = NULL)
```

```r
fit(x, y, sample_weight = NULL)
```

```r
predict(x)
```

```r
cleanup()
```

```r
get_params(deep = TRUE)
```
score(x, y, sample_weight = NULL)

feature_importances()

dump_model()

save_model(filename)

Super class

RGF::Internal_class -> RGF_Regressor

Methods

Public methods:

- RGF_Regressor$new()
- RGF_Regressor$clone()

Method new():

Usage:

RGF_Regressor$new(
  max_leaf = 500,
  test_interval = 100,
  algorithm = "RGF",
  loss = "LS",
  reg_depth = 1,
  l2 = 0.1,
  sl2 = NULL,
  normalize = TRUE,
  min_samples_leaf = 10,
  n_iter = NULL,
  n_tree_search = 1,
  opt_interval = 100,
  learning_rate = 0.5,
  memory_policy = "generous",
  verbose = 0,
  init_model = NULL
)

Arguments:

max_leaf an integer. Training will be terminated when the number of leaf nodes in the forest reaches this value.

test_interval an integer. Test interval in terms of the number of leaf nodes.
algorithm a character string specifying the Regularization algorithm. One of "RGF" (RGF with L2 regularization on leaf-only models), "RGF_Opt" (RGF with min-penalty regularization) or "RGF_Sib" (RGF with min-penalty regularization with the sum-to-zero sibling constraints).

loss a character string specifying the Loss function. One of "LS" (Square loss), "Expo" (Exponential loss) or "Log" (Logistic loss).

reg_depth a float. Must be no smaller than 1.0. Meant for being used with the algorithm RGF Opt or RGF Sib. A larger value penalizes deeper nodes more severely.

l2 a float. Used to control the degree of L2 regularization.

sl2 a float or NULL. Override L2 regularization parameter l2 for the process of growing the forest. That is, if specified, the weight correction process uses l2 and the forest growing process uses sl2. If NULL, no override takes place and l2 is used throughout training.

normalize a boolean. If True, training targets are normalized so that the average becomes zero.

min_samples_leaf an integer or a float. Minimum number of training data points in each leaf node. If an integer, then consider min_samples_leaf as the minimum number. If a float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

n_iter an integer or NULL. The number of iterations of coordinate descent to optimize weights. If NULL, 10 is used for loss = "LS" and 5 for loss = "Expo" or "Log".

n_tree_search an integer. The number of trees to be searched for the nodes to split. The most recently grown trees are searched first.

opt_interval an integer. Weight optimization interval in terms of the number of leaf nodes. For example, by default, weight optimization is performed every time approximately 100 leaf nodes are newly added to the forest.

learning_rate a float. Step size of Newton updates used in coordinate descent to optimize weights.

memory_policy a character string. One of "conservative" (it uses less memory at the expense of longer runtime. Try only when with default value it uses too much memory) or "generous" (it runs faster using more memory by keeping the sorted orders of the features on memory for reuse). Memory using policy.

verbose an integer. Controls the verbosity of the tree building process.

init_model either NULL or a character string, optional (default=NULL). Filename of a previously saved model from which training should do warm-start. If model has been saved into multiple files, do not include numerical suffixes in the filename. NOTE: Make sure you haven’t forgotten to increase the value of the max_leaf parameter regarding to the specified warm-start model because warm-start model trees are counted in the overall number of trees.

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

Usage:

RGF_Regressor$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.
References

https://github.com/rgf-team/rgf/tree/master/python-package, Rie Johnson and Tong Zhang, Learning Nonlinear Functions Using Regularized Greedy Forest

Examples

```r
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("rgf.sklearn")) {
      library(RGF)
      set.seed(1)
      x = matrix(runif(1000), nrow = 100, ncol = 10)
      y = runif(100)
      RGF_regr = RGF_Regressor$new(max_leaf = 50)
      RGF_regr$fit(x, y)
      preds = RGF_regr$predict(x)
    }
  }
}, silent = TRUE)
```

TO_scipy_sparse

conversion of an R sparse matrix to a scipy sparse matrix

Description

conversion of an R sparse matrix to a scipy sparse matrix

Usage

```r
TO_scipy_sparse(R_sparse_matrix)
```

Arguments

- `R_sparse_matrix`
  
an R sparse matrix. Acceptable input objects are either a `dgCMatrix` or a `dgRMatrix`.

Details

This function allows the user to convert either an R \( \text{dgCMatrix} \) or a \( \text{dgRMatrix} \) to a scipy sparse matrix (\text{scipy.sparse.csc_matrix} or \text{scipy.sparse.csr_matrix}). This is useful because the \text{RGF} package accepts besides an R dense matrix also python sparse matrices as input.

The \text{dgCMatrix} class is a class of sparse numeric matrices in the compressed, sparse, column-oriented format. The \text{dgRMatrix} class is a class of sparse numeric matrices in the compressed, sparse, row-oriented format.

References


Examples

```r
try({
  if (reticulate::py_available(initialize = FALSE)) {
    if (reticulate::py_module_available("scipy")) {
      if (Sys.info()["sysname"] != "Darwin") {
        library(RGF)

        # 'dgCMatrix' sparse matrix
        #--------------------------
        data = c(1, 0, 2, 0, 0, 3, 4, 5, 6)
        dgcM = Matrix::Matrix(
          data = data,
          nrow = 3,
          ncol = 3,
          byrow = TRUE,
          sparse = TRUE
        )

        print(dim(dgcM))

        res = TO_scipy_sparse(dgcM)

        print(res$shape)

        # 'dgRMatrix' sparse matrix
        #--------------------------
        dgrM = as(dgcM, "RsparseMatrix")

        print(dim(dgrM))
      }
    }
  }
})
```
res_dgr = TO_scipy_sparse(dgrM)
print(res_dgr$shape)
Index

FastRGF_Classifier, 2
FastRGF_Regressor, 5

mat_2scipy_sparse, 8

RGF::Internal_class, 3, 6, 10, 15
RGF_Classifier, 9
RGF_cleanup_temp_files, 13
RGF_Regressor, 14

TO_scipy_sparse, 17