Package 'RGF'

April 14, 2018

Type Package

Title Regularized Greedy Forest

Version 1.0.2

Date 2018-04-14

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

BugReports https://github.com/mlampros/RGF/issues

URL https://github.com/mlampros/RGF

Description Regularized Greedy Forest wrapper of the 'Regularized Greedy Forest' <https://github.com/fukatani/rgf_python> 'python' package, which also includes a Multi-core implementation (FastRGF) <https://github.com/baidu/fast_rgf>.

License GPL-3

Copyright inst/COPYRIGHTS

SystemRequirements Python (>= 2.7), rgf_python, fast_rgf, scikit-learn (>= 0.19.0), scipy (>= 1.0.0), numpy (>= 1.14.0). Detailed installation instructions for each operating system can be found in the README file.

Depends R(>= 3.2.3)

Imports reticulate, R6, Matrix

Suggests testthat, covr, knitr, rmarkdown

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Lampros Mouselimis [aut, cre],
Ryosuke Fukatani [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)

Repository CRAN

Date/Publication 2018-04-14 12:46:53 UTC

R topics documented:

`dgCMatrix_2scipy_sparse`
conversion of an R dgCMatrix to a scipy sparse matrix

Description

conversion of an R dgCMatrix to a scipy sparse matrix

Usage

```
dgCMatrix_2scipy_sparse(dgc_mat_object)
```

Arguments

`dgc_mat_object`	an R sparse matrix of type *dgCMatrix*

Details

This function allows the user to convert an R *dgCMatrix* to a scipy sparse matrix (*scipy.sparse.csc_matrix*). This is useful because the Regularized Greedy Forest algorithm accepts besides an R dense matrix also python sparse matrices as input.

The dgCMatrix class is a class of sparse numeric matrices in the compressed, sparse, *column-oriented format*. In this implementation the non-zero elements in the columns are sorted into increasing row order. dgCMatrix is the "standard" class for sparse numeric matrices in the *Matrix* package.

References

https://stat.ethz.ch/R-manual/R-devel/library/Matrix/html/dgCMatrix-class.html, https://docs.scipy.org/doc/scipy/reference/g

Examples

```
if (reticulate::py_available() && reticulate::py_module_available("scipy")) {

  if (Sys.info()["sysname"] != 'Darwin') {

    library(RGF)

    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 = dgCMatrix_2scipy_sparse(dgcM)

    print(res$shape)
  }
}
```

FastRGF_Classifier *A Fast Regularized Greedy Forest classifier*

Description

A Fast Regularized Greedy Forest classifier

Usage

```
# 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)
```

Arguments

x	an R matrix (object) or a Python sparse matrix (object) of shape c(n_samples, n_features). The training input samples. The sparse matrix should be a Python

sparse matrix. The helper functions *mat_2scipy_sparse* and *dgCMatrix_2scipy_sparse* allow the user to convert an R matrix / R dgCMatrix to a scipy sparse matrix.

`y` a vector of shape c(n_samples). The target values (real numbers in regression).

`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 selected. Meant for being used with sparse data (Original name: discretize.sparse.min_occrrences.)

`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.)

Format

An object of class `R6ClassGenerator` of length 24.

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/fukatani/rgf_python/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

```
FastRGF_Classifier$new(n_estimators = 500, max_depth = 6, max_leaf = 50, tree_gain_ratio = 1.0, min

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

References

https://github.com/fukatani/rgf_python, *Tong Zhang, FastRGF: Multi-core Implementation of Regularized Greedy Forest (https://github.com/baidu/fast_rgf)*

Examples

```
if (reticulate::py_available() && 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)
}
```

FastRGF_Regressor *A Fast Regularized Greedy Forest regressor*

Description

A Fast Regularized Greedy Forest regressor

Usage

```
# 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)
```

Arguments

x	an R matrix (object) or a Python sparse matrix (object) of shape c(n_samples, n_features). The training input samples. The sparse matrix should be a Python sparse matrix. The helper functions *mat_2scipy_sparse* and *dgCMatrix_2scipy_sparse* allow the user to convert an R matrix / R dgCMatrix to a scipy sparse matrix.
y	a vector of shape c(n_samples). The target values (real numbers in regression).
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.)

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 selected. Meant for being used with sparse data (Original name: discretize.sparse.min_occrrences.)
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.)

Format

An object of class `R6ClassGenerator` of length 24.

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/fukatani/rgf_python/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

```
FastRGF_Regressor$new(n_estimators = 500, max_depth = 6, max_leaf = 50, tree_gain_ratio = 1.0, min_

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

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

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

References

https://github.com/fukatani/rgf_python, *Tong Zhang, FastRGF: Multi-core Implementation of Regularized Greedy Forest (https://github.com/baidu/fast_rgf)*

Examples

```
if (reticulate::py_available() && 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)
}
```

---

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

```
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

```
if (reticulate::py_available() && 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)
}
```

RGF_Classifier *Regularized Greedy Forest classifier*

Description

Regularized Greedy Forest classifier

Usage

```
# 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)
```

Arguments

x	an R matrix (object) or a Python sparse matrix (object) of shape c(n_samples, n_features). The training input samples. The sparse matrix should be a Python sparse matrix. The helper functions *mat_2scipy_sparse* and *dgCMatrix_2scipy_sparse* allow the user to convert an R matrix / R dgCMatrix to a scipy sparse matrix.
y	a vector of shape c(n_samples). The target values (class labels in classification).
sample_weight	a vector of shape c(n_samples) or NULL. Individual weights for each sample.
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 int 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.
calc_prob	a character string. One of *"sigmoid"* or *"softmax"*. Method of probability calculation.

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

Format

An object of class `R6ClassGenerator` of length 24.

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/fukatani/rgf_python/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

```
RGF_Classifier$new(max_leaf = 1000, test_interval = 100, algorithm = "RGF", loss = "Log", reg_depth

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

References

https://github.com/fukatani/rgf_python, *Rie Johnson and Tong Zhang, Learning Nonlinear Functions Using Regularized Greedy Forest*

Examples

```
if (reticulate::py_available() && 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)
}
```

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/fukatani/rgf_python/is*
for more details.

References

https://github.com/fukatani/rgf_python

Examples

```
## Not run:
library(RGF)

RGF_cleanup_temp_files()

## End(Not run)
```

RGF_Regressor *Regularized Greedy Forest regressor*

Description

Regularized Greedy Forest regressor

Usage

```
# 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)
```

Arguments

x	an R matrix (object) or a Python sparse matrix (object) of shape c(n_samples, n_features). The training input samples. The sparse matrix should be a Python sparse matrix. The helper functions *mat_2scipy_sparse* and *dgCMatrix_2scipy_sparse* allow the user to convert an R matrix / R dgCMatrix to a scipy sparse matrix.
y	a vector of shape c(n_samples). The target values (real numbers in regression).
sample_weight	a vector of shape c(n_samples) or NULL. Individual weights for each sample.
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 int 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.

Format

An object of class `R6ClassGenerator` of length 24.

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/fukatani/rgf_python/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

```
RGF_Regressor$new(max_leaf = 500, test_interval = 100, algorithm = "RGF", loss = "LS", reg_depth =

--------------
```

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

References

https://github.com/fukatani/rgf_python, *Rie Johnson and Tong Zhang, Learning Nonlinear Functions Using Regularized Greedy Forest*

Examples

```
if (reticulate::py_available() && 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)
}
```

Index