Package 'HPdclassifier'

March 25, 2014

Type Package

Title Distributed classifiers for Big Data

Version 0.5.0

Date 2014-01-22

Author Arash Fard

Maintainer Arash Fard <afard@vertica.com>

Depends R (>= 3.0.1), distributedR, randomForest

Description It provides distributed algorithms for learning classifiers. It is written based on the infrastructure created in HP-Labs for distributed computing in R.

License GPL (>= 2)

R topics documented:

`HPdclassifier-package`

Distributed algorithms for classifiers

Description

HPdclassifier provides several distributed algorithms for classifiers. It is written based on the infrastructure created in HP-Labs for distributed computing in R.

Details

Package:	HPdclassifier
Type:	Package
Version:	0.4.0
Date:	2014-01-22

Main Functions:

- hpdrandomForest: It is a distributed function for randomForest
- predictHPdRF: distributed predict method for applying a random forest objects on a darray or a dframe

Author(s)

Arash Fard <afard@vertica.com>

References

1. Using R for Iterative and Incremental Processing. Shivaram Venkataraman, Indrajit Roy, Alvin AuYoung, Rob Schreiber. HotCloud 2012, Boston, USA.

`hpdrandomForest` *Distributed randomForest*

Description

hpdrandomForest function runs `randomForest` function of randomForest package in a distributed fashion.

Description

`hpdrandomForest` calls several instances of `randomForest` distributed across a cluster system. Therefore, the master distributes the input data among all R-executors of the distributedR environment, and trees on different sub-sections of the forest are created simultaneously. At the end, all these trees are combined to result a single forest.

The interface of `hpdrandomForest` is similar to `randomForest`. Indeed it adds two arguments `nExecutor` and `trace`, and removes several other arguments `do.trace`, `keep.inbag` `proximity`, and `oob.prox`. Its returned result is also completely compatible to the result of `randomForest`.

Usage

```
## S3 method for class 'formula'
hpdrandomForest(formula, data=NULL, ..., subset, na.action=na.fail,
                                        nExecutor, trace=FALSE)
## Default S3 method:
hpdrandomForest(x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
         mtry=if (!is.null(y) && !is.factor(y) && !is.dframe(y))
         max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
         replace=TRUE, classwt=NULL, cutoff, strata,
         sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),
         nodesize = if (!is.null(y) && !is.factor(y) &&
         !is.dframe(y)) 5 else 1,
         maxnodes = NULL,
         importance=FALSE, localImp=FALSE, nPerm=1,
         norm.votes=TRUE,
         keep.forest=!is.null(y) && is.null(xtest),
         corr.bias=FALSE, nExecutor, trace=FALSE, ...)
```

Arguments

data	an optional data frame containing the variables in the model. By default the variables are taken from the environment which hpdrandomForest is called from.
subset	an index vector indicating which rows should be used. (NOTE: If given, this argument must be named.)
na.action	A function to specify the action to be taken if NAs are found. (NOTE: If given, this argument must be named.)
formula	a formula describing the model to be fitted.
x	when a data frame or a matrix of predictors assigned to x, its size should not be bigger than 64MB. For bigger datasets, darray or dframe should be used. darray is more memory efficient than dframe, but it does not support categorical data. Therefore, using darray is highly recommended when there is no categorical data.
y	a response vector. If a factor, classification is assumed, otherwise regression is assumed. If omitted, hpdrandomForest will run in unsupervised mode. When x is a distributed structure (darray or a dframe), y should be also a distributed structure with a single column.
xtest	a data frame or matrix (like x) containing predictors for the test set. When x is a darray or a dframe, it should be of the same type.
ytest	response for the test set. Its type should be consistent with y. Moreover, it should have a single column.
ntree	Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.
mtry	Number of variables randomly sampled as candidates at each split. Note that the default values are different for classification (sqrt(p) where p is number of variables in x) and regression (p/3)
replace	Should sampling of cases be done with or without replacement?
classwt	Priors of the classes. Need not add up to one. Ignored for regression.
cutoff	(Classification only) A vector of length equal to number of classes. The 'winning' class for an observation is the one with the maximum ratio of proportion of votes to cutoff. Default is 1/k where k is the number of classes (i.e., majority vote wins).
strata	A (factor) variable that is used for stratified sampling.
sampsize	Size(s) of sample to draw. For classification, if sampsize is a vector of the length the number of strata, then sampling is stratified by strata, and the elements of sampsize indicate the numbers to be drawn from the strata.
nodesize	Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time). Note that the default values are different for classification (1) and regression (5).
maxnodes	Maximum number of terminal nodes trees in the forest can have. If not given, trees are grown to the maximum possible (subject to limits by nodesize). If set larger than maximum possible, a warning is issued.
importance	Should importance of predictors be assessed?
localImp	Should casewise importance measure be computed? (Setting this to TRUE will override importance.)

nPerm	Number of times the OOB data are permuted per tree for assessing variable importance. Number larger than 1 gives slightly more stable estimate, but not very effective. Currently only implemented for regression.
norm.votes	If TRUE (default), the final result of votes are expressed as fractions. If FALSE, raw vote counts are returned (useful for combining results from different runs). Ignored for regression.
keep.forest	If set to FALSE, the forest will not be retained in the output object. If xtest is given, defaults to FALSE.
corr.bias	perform bias correction for regression? Note: Experimental. Use at your own risk.
...	optional parameters to be passed to the low level function.
nExecutor	a positive integer number indicating the number of tasks for running the function. To have optimal performance, it is recommended to have this number smaller than the number of R-executors in the distributedR environment. It cannot be bigger than ntree.
trace	when this argument is true, intermediate steps of the progress are displayed.

Value

An object of class randomForest. The result is consistent with the result of the combine function in randomForest package.

Note

When ntree is not big enough in comparison to nExecutor, some of the returned predicted values may become NULL.

Returned values for err.rate, votes, and oob.times are valid only for classification type.

Author(s)

Arash Fard <afard@vertica.com>

References

Breiman, L. (2001), *Random Forests*, Machine Learning 45(1), 5-32.

Breiman, L (2002), "Manual On Setting Up, Using, And Understanding Random Forests V3.1", http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf.

Random Forests V4.6-7, http://cran.r-project.org/web/packages/randomForest/randomForest.pdf.

Examples

```
 ## Not run:

library(HPdclassifier)
distributedR_start()
drs <- distributedR_status()
nparts <- sum(drs$Ins)

## Classification:
##data(iris)
iris.rf <- hpdrandomForest(Species ~ ., data=iris, importance=TRUE,
```

```
                          nExecutor=nparts)
print(iris.rf)

## The `unsupervised' case:
iris.urf <- hpdrandomForest(iris[, -5], nExecutor=nparts)
MDSplot(iris.urf, iris$Species)

## stratified sampling: draw 20, 30, and 20 of the species to grow each tree.
(iris.rf2 <- hpdrandomForest(iris[1:4], iris$Species,
                          sampsize=c(20, 30, 20), nExecutor=nparts))

## Regression:
## data(airquality)
ozone.rf <- hpdrandomForest(Ozone ~ ., data=airquality, mtry=3,
                         importance=TRUE, na.action=na.omit, nExecutor=nparts)
print(ozone.rf)
## Show "importance" of variables: higher value mean more important:
round(importance(ozone.rf), 2)

## "x" can be a matrix instead of a data frame:
x <- matrix(runif(5e2), 100)
y <- gl(2, 50)
(myrf <- hpdrandomForest(x, y, nExecutor=nparts))
(predict(myrf, x))

## "complicated" formula:
(swiss.rf <- hpdrandomForest(sqrt(Fertility) ~ . - Catholic + I(Catholic < 50),
                          data=swiss, nExecutor=nparts))
(predict(swiss.rf, swiss))
## Test use of 32-level factor as a predictor:
x <- data.frame(x1=gl(32, 5), x2=runif(160), y=rnorm(160))
(rf1 <- hpdrandomForest(x[-3], x[[3]], ntree=10, nExecutor=nparts))

## Grow no more than 4 nodes per tree:
(treesize(hpdrandomForest(Species ~ ., data=iris, maxnodes=4, ntree=30,
                                  nExecutor=nparts)))

distributedR_shutdown()

## End(Not run)
```

`predictHPdRF` *distributed predict method for applying a random forest objects on a darray or a dframe*

Description

Prediction of distributed test data using random forest.

Usage

```
predictHPdRF(object, newdata, trace=FALSE)
```

Arguments

object	an object of class randomForest, as that created by the function randomForest or hpdrandomForest.
newdata	a darray or a dframe containing new data. darray is highly recommended when there is no categorial data

Value

It returns a darray or a dframe of predicted classes. The type of returned value will be darray unless there is any categorical data.

Author(s)

Arash Fard <afard@vertica.com>

References

Breiman, L. (2001), *Random Forests*, Machine Learning 45(1), 5-32.

See Also

hpdrandomForest

Examples

```
 ## Not run:
# example for darray
library(HPdclassifier)
distributedR_start()
drs <- distributedR_status()
nparts <- sum(drs$Ins)

nSamples <- 100
nAttributes <- 5
nSplits <- 1

dax <- darray(c(nSamples,nAttributes), c(round(nSamples/nSplits),nAttributes))
day <- darray(c(nSamples,1), c(round(nSamples/nSplits),1))

foreach(i, 1:npartitions(dax), function(x=splits(dax,i),y=splits(day,i),id=i){
    x <- matrix(runif(nrow(x)*ncol(x)), nrow(x),ncol(x))
    y <- matrix(runif(nrow(y)), nrow(y), 1)
    update(x)
    update(y)
})

(myrf1 <- hpdrandomForest(dax, day, nExecutor=nparts))
dp <- predictHPdRF(myrf1, dax)

# example for dframe
nSamples <- 100
nAttributes <- 5
nSplits <- 4

dfx <- dframe(c(nSamples,nAttributes), c(round(nSamples/nSplits),nAttributes))
```

```
dfy <- dframe(c(nSamples,1), c(round(nSamples/nSpllits),1))
blockSize <- dfx@blocks[1]

foreach(i, 1:npartitions(dfx), function(xi=splits(dfx,i),yi=splits(dfy,i),
  blockSize=blockSize,nAttributes=nAttributes){
     xi <- data.frame( matrix(runif(blockSize*nAttributes,1,10),
           nrow=blockSize,ncol=nAttributes) )
     yi <- data.frame( gl(2, blockSize/2) )
     update(xi)
     update(yi)
})

(myrf2 <- hpdrandomForest(dfx, dfy, nExecutor=nparts))
fp2 <- predictHPdRF(myrf2, dfx)


## End(Not run)
```

Index