HPdclassifier – Distributed Classification

User Guide

Contents

User Guide

1. Introduction

The HPdclassifier is a package for distributed classifier algorithms. It is written based on the distributed R infrastructure developed by HP-Labs. At the current version, it contains only distributed random forest algorithm. The main functions of the this package are:

- hpdrandomForest: It runs randomForest in a distributed fashion.
- predictHPdRF: It provides distributed predict method for applying a random forest objects on a darray or a dframe.

2. hpdrandomForest

hpdrandomForest function is a distributed version of randomForest function which is available in randomForest package. Indeed, 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. Every R-executor should have access to the whole samples; therefore, the function is bounded by the physical memory of a single node of the cluster. In other words, hpdrandomForest does not scale in terms of memory usage, but aims to speedup randomForest because different parts of the forest are computed in parallel.

hpdrandomForest supports two interfaces similar to randomForest. Indeed, their input arguments are the same, but it adds two arguments *nExecutor* and *trace*, and removes two arguments *do.trace* and *keep.inbag*. The output of hpdrandomForest is also fully compatible with output of randomForest.

It should be noticed that the input dataset of hpdrandomForest can be ordinary R objects similar to randomForest; e.g., matrix and data frame. However, the size of these objects should not exceed 65MB. For bigger datasets darray or dframe should be used. Only the second interface, which does not have formula, can be used for inputs of type darray and dframe. The darray structure does not support categorical data, but it is more memory efficient in comparison to dframe. Therefore, the usage of darray is highly recommended when there is no categorical data.

More detail about the interface of the function and its input arguments can be find in the manual of the package.

2.1. Example 1

This classification example (Figure 1) uses a small data-frame called 'iris', which is available in the standard distribution of R (datasets package). The input dataset is of data.frame in this example.

```
> library(HPdclassifier)      # loading the library
Loading required package: distributedR
Loading required package: Rcpp
Loading required package: RInside
Loading required package: randomForest
> distributedR_start()     # starting the distributed environment
Workers registered - 1/1.
All 1 workers are registered.
[1] TRUE
> (ds <- distributedR_status())    # saving the status
         Workers Inst SysMem MemUsed DarrayQuota DarrayUsed
1 127.0.0.1:9090    7   7804    7134        3511          0
> nparts <- sum(ds$Inst)  # number of available distributed instances
> # running hpdrandomForest
iris.rf <- hpdrandomForest(Species ~ ., data=iris, importance=TRUE, proximity=TRUE,
nExecutor=nparts)
> print(iris.rf)

Call:
 hpdrandomForest(formula = Species ~ ., data=iris, importance=TRUE, proximity=TRUE,
nExecutor=nparts)
       Type of random forest: classification
             Number of trees: 504
No. of variables tried at each split: 2
```

Figure 1. example 1 for hpdrandomForest

2.2. Example 2

This example is similar to example 1, but we have used dframe as the type of input dataset. The dframe data structure should be only used for big datasets. A function provided in HPdata package can be used to load a dframe from a database. The script is displayed in Figure 2.

```
> library(HPdclassifier)     # loading the library
Loading required package: distributedR
Loading required package: Rcpp
Loading required package: RInside
Loading required package: randomForest
> distributedR_start()     # starting the distributed environment
Workers registered - 1/1.
All 1 workers are registered.
[1] TRUE
> (ds <- distributedR_status())    # saving the status
         Workers Inst SysMem MemUsed DarrayQuota DarrayUsed
1 127.0.0.1:9090    7   7804    7134        3511          0
> nparts <- sum(ds$Inst)  # number of available distributed instances
> # creating dframes
> dfx <- dframe(c(150,4),c(50,4))
> dfy <- dframe(c(150,1),c(50,1))
> # loading dframes
> foreach(i, 1:npartitions(dfx), function(dfxi=splits(dfx,i),dfyi=splits(dfy,i),id=i){
+    offset <- (id-1)*50
+    start <- 1+offset
+    end <- start + 49
+    dfxi <- data.frame(iris[start:end,-5])
+    dfyi <- data.frame(iris[start:end, 5])
+    update(dfxi)
+    update(dfyi)
+})
Progress: 100%
[1] TRUE
> # running hpdrandomForest
> myrf1 <- hpdrandomForest(x=dfx, y=dfy, nExecutor=nparts)
> print(myrf1)

Call:
 hpdrandomForest.defualt(x = dfx, y = dfx, nExecutor=nparts)
       Type of random forest: classification
             Number of trees: 504
No. of variables tried at each split: 2
```

Figure 2. example 2 for hpdrandomForest

2.3. Performance evaluation

The following experiments show the performance of hpdrandomForest function for different number of concurrent R-executors. The tests are performed for 2 different modes of the function: regression and classification. It should be noticed that the function can accept 3 different types for the input data: non-distributed (matrix or data-frame), darray, and dframe. When the size of the input data is smaller than 64MB, using a non-distributed data structure is recommended. When the input data is bigger than 64MB but non-categorical, using darray is recommended. For categorical input data which is bigger than 64MB,

dframe is recommended. The benchmarks cover all these different situations. The test scrip of each benchmark is also presented.

Specification of each node for experiments:

- 2 CPU per Node
- Core(s) per socket: 6
- CPU model: Intel® Xeon® X5650 (12MB Cache, 2.67GHz)
- 96GB RAM
- 10Gb/s Ethernet Network
- CentOS release 6.4 (Final)
- R version 3.0.1

Regression mode – input type matrix

For this set of tests, a dataset with 10e5 samples is synthesized. Each sample has 10 predictors and 1 response. All the attributes are numerical; therefore, the function runs in the regression mode. The specified number of trees in the forest is 500. The results for running the function on a single node and different number of instances (R-executors) are displayed in Table 1 and Figure 3. Moreover, the results of running the function on multiple nodes with 12 instances per node are displayed in Table 2 and Figure 4.

Table 1. hpdrandomForest on a single node (#workers = 1), Input type: matrix

#Workers	#Instances/worker	Time (sec)	Speedup
1	1	6797.732	1.00
1	2	3502.092	1.94
1	4	2129.511	3.19
1	8	2501.207	2.72

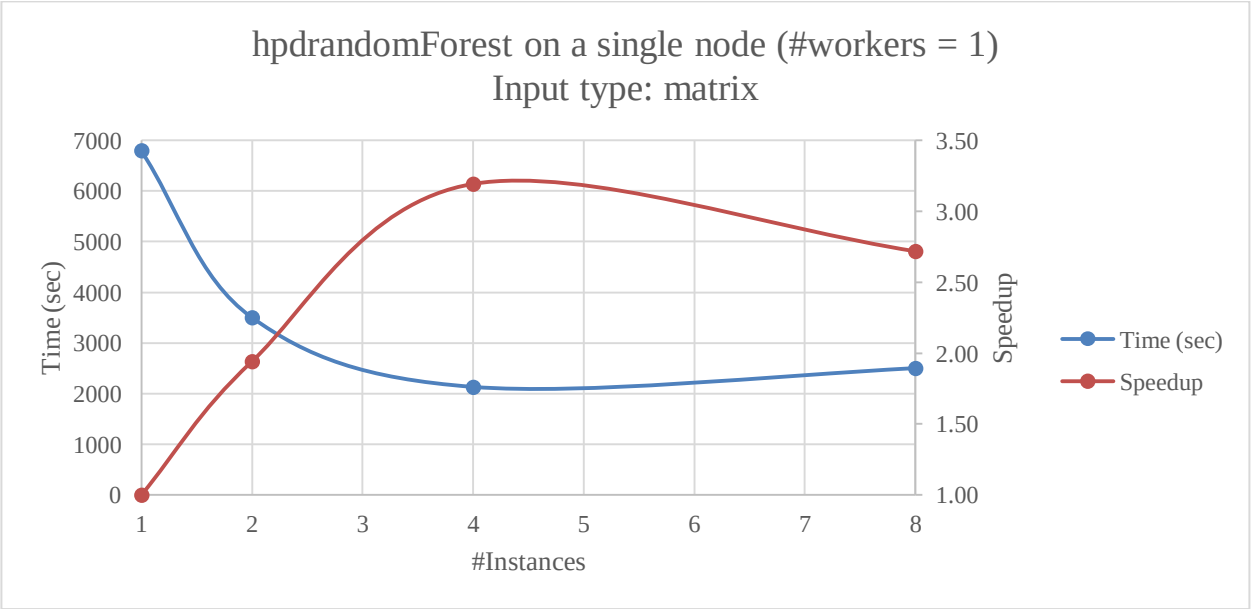

Figure 3. hpdrandomForest on a single node, Input type: matrix

Table 2. hpdrandomForest on multiple nodes (#Instances/worker = 12), Input type: matrix

#Workers	#Instances/worker	Time (sec)	Speedup
1	12	3073.064	1.00
2	12	1560.604	1.97
4	12	835.674	3.68
8	12	450.938	6.81
12	12	333.136	9.22

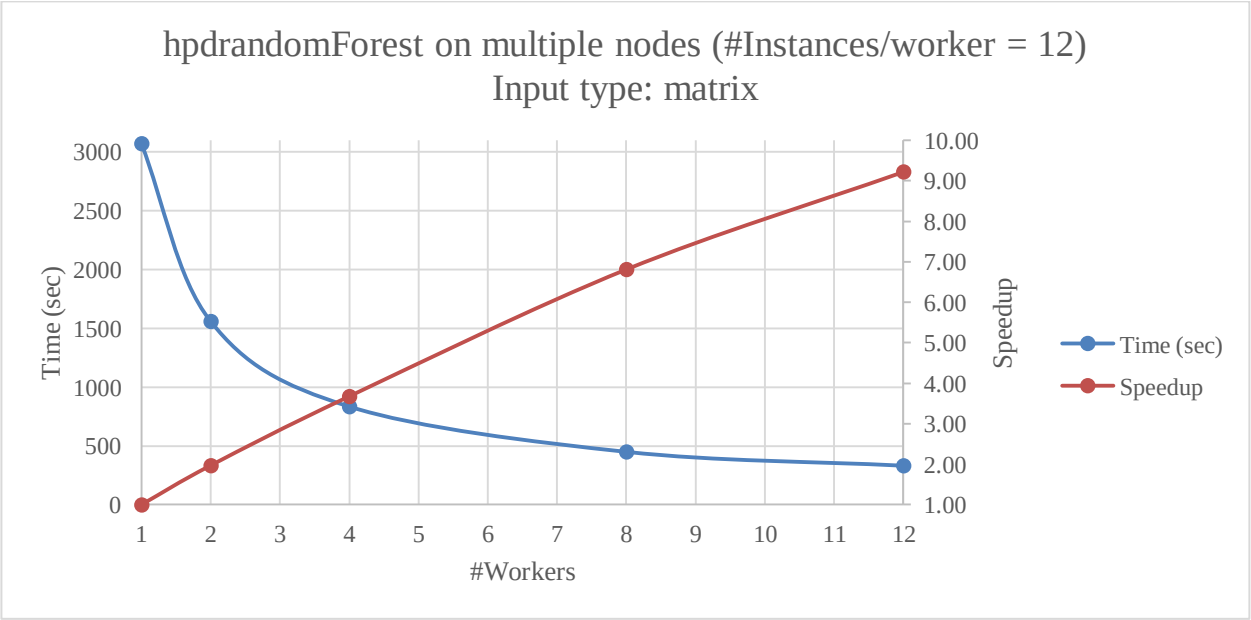

Figure 4. hpdrandomForest on multiple nodes, Input type: matrix

Regression mode – input type darray

For this set of tests, a dataset with 10e5 samples is synthesized. Each sample has 100 predictors and 1 response. All the attributes are numerical; therefore, the function runs in the regression mode. The specified number of trees in the forest is 150. The results of running the function on multiple nodes with 12 instances per node are displayed in Table 3 and Figure 5.

Table 3. hpdrandomForest on multiple nodes (#Instances/worker = 12), Input type: darray

#Workers	#Instances/worker	Time (sec)	Speedup
1	12	1921.868	1.00
2	12	1028.091	1.87
4	12	615.776	3.12
8	12	375.728	5.12

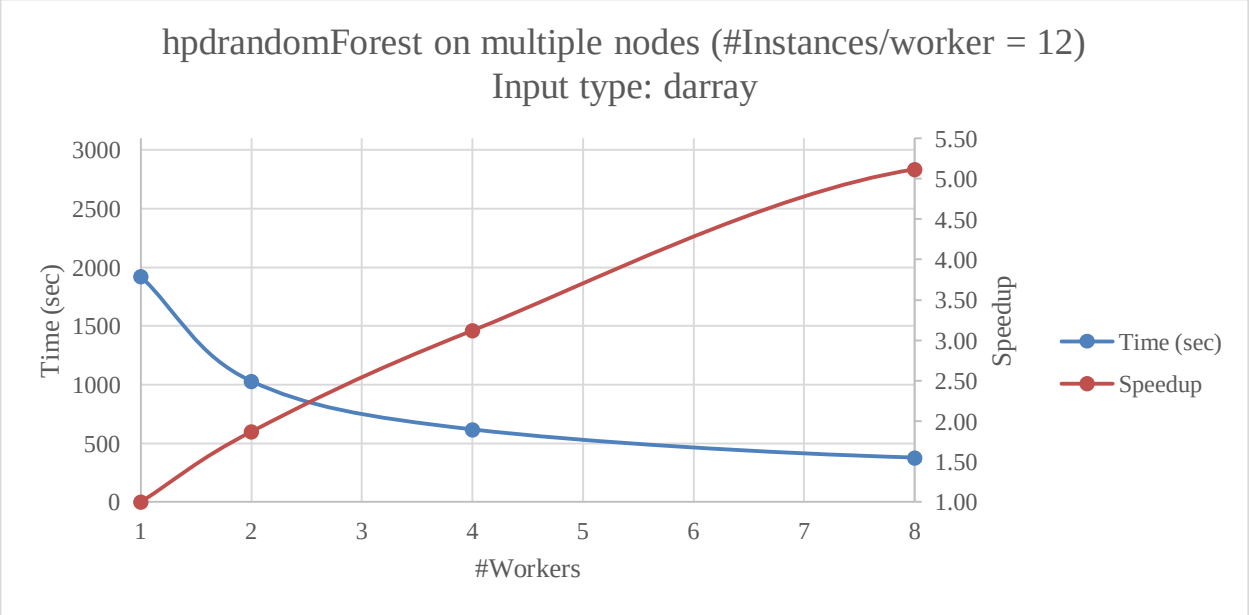

Figure 5. hpdrandomForest on multiple nodes, Input type: darray

Classification mode – input type dframe

For this set of tests, a dataset with 10e5 samples is synthesized. Each sample has 100 predictors and 1 response. The response is categorical; therefore, the function runs in the classification mode. The specified number of trees in the forest is 150. The results for running the function on a single node and different number of instances (R-executors) are displayed in Table 4 and Figure 6.

Table 4. hpdrandomForest on a single node (#worker = 1), Input type: dframe

#Workers	#Instances/worker	Time (sec)	Speedup
1	1	583.108	1.00
1	2	322.013	1.81
1	4	193.16	3.02
1	8	151.212	3.86

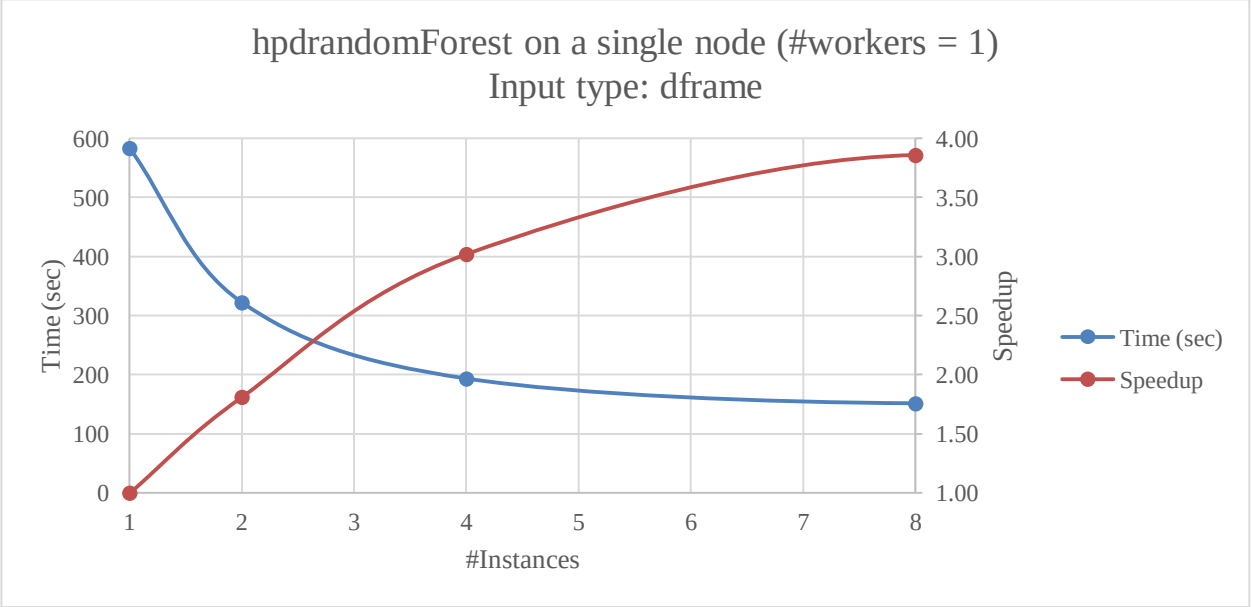

Figure 6. hpdrandomForest on a single node, Input type: dframe

3. predictHPdRF function

The predictHPdRF function provides a distributed predict method for applying a random forest objects on a new dataset stored in a darray or a dframe.

3.1. Example 1

This example shows the usage of the predictHPdRF function for predicting values of a set of samples stored in a darray.

```
> library(HPdclassifier)     # loading the library
Loading required package: distributedR
Loading required package: Rcpp
Loading required package: RInside
Loading required package: randomForest
> distributedR_start()    # starting the distributed environment
Workers registered - 1/1.
All 1 workers are registered.
[1] TRUE
> (ds <- distributedR_status())   # saving the status
         Workers Inst SysMem MemUsed DarrayQuota DarrayUsed
1 127.0.0.1:9090    7   7804    7134        3511          0
> nparts <- sum(ds$Inst)  # number of available distributed instances
>
> nSamples <- 100 # number of samples
> nAttributes <- 5 # number of attributes of each sample
> nSpllits <- 1 # number of splits in each darray
> # creating darrays
> dax <- darray(c(nSamples,nAttributes), c(round(nSamples/nSpllits),nAttributes))
> day <- darray(c(nSamples,1), c(round(nSamples/nSpllits),1))
> # loading darrays
> 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)
+})
Progress: 100%
[1] TRUE
> # running hpdrandomForest
> myrf <- hpdrandomForest(dax, day, nExecutor=nparts)
> # prediction
> dp <- predictHPdRF(myrf, dax)
```

Figure 7. An example for using darray with predictHPdRF

3.2. Example 2

This example shows the usage of the predictHPdRF function for predicting values of a set of samples stored in a dframe.

```
> library(HPdclassifier)     # loading the library
Loading required package: distributedR
Loading required package: Rcpp
Loading required package: RInside
Loading required package: randomForest
> distributedR_start()    # starting the distributed environment
Workers registered - 1/1.
All 1 workers are registered.
[1] TRUE
> (ds <- distributedR_status())   # saving the status
         Workers Inst SysMem MemUsed DarrayQuota DarrayUsed
1 127.0.0.1:9090    7   7804    7134        3511          0
> nparts <- sum(ds$Inst)  # number of available distributed instances
>
> nSamples <- 100 # number of samples
> nAttributes <- 5 # number of attributes of each sample
> nSpllits <- 4 # number of splits in each dframe
> # creating dframes
> dfx <- dframe(c(nSamples,nAttributes), c(round(nSamples/nSpllits),nAttributes))
> dfy <- dframe(c(nSamples,1), c(round(nSamples/nSpllits),1))
> blockSize <- dfx@blocks[1]
 > # loading dframes
> 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)
+})
 Progress: 100%
[1] TRUE
> # running hpdrandomForest
> myrf <- hpdrandomForest(dfx, dfy, nExecutor=nparts)
> # prediction
> dp <- predictHPdRF(myrf, dfx)
```

Figure 8. An example for using dframe with predictHPdRF