

# HPdclassifier – Distributed Classification

## User Guide

---

### Contents

1. Introduction .....	2
2. hpdrandomForest .....	2
2.1. Example 1 .....	3
2.2. Example 2 .....	3
3. predictHPdRF function .....	4
3.1. Example 1 .....	4
3.2. Example 2 .....	6

### 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.

# HPdclassifier – Distributed Classification

## User Guide

---

### 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.

# HPdclassifier – Distributed Classification

## User Guide

```
> 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
> myrfl <- hpdrandomForest(x=dfx, y=dfy, nExecutor=nparts)
> print(myrfl)

Call:
hpdrandomForest.default(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

### 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.

# HPdclassifier – Distributed Classification

## User Guide

---

```
> 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 3. 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 4. An example for using `dframe` with `predictHPdRF`