HPdata – Data Manipulation for distributedR

User Guide

Contents

HPdata – Data Manipulation for distributedR

User Guide

1. Introduction

The HPdata is a package which encapsulate all data related functions - data loading, data preparation etc for distributed R environment. It is written based on the infrastructure created in HP-Labs for distributed computing in R. The main functions of the this package are:

- db2darray: It is an example for loading a set of unlabeled samples stored in a table to a darray.
- db2darrays: It is an example for loading a set of labeled samples stored in a table to two darrays.
- db2matrix: It is an example for loading a set of unlabeled samples stored in a table to a matrix.
- db2dframe: It is an example for loading a set of samples stored in a table to a dframe.
- db2dgraph: It loads an adjaceny matrix to a darray from an edgelist strored in a database.
- splitGraphFile: It is an example for splitting an edge-list file of a graph, and distributing the results among the active nodes of a cluster system.
- file2dgraph: It loads an adjaceny matrix to a darray from an edgelist strored in a set of files.

2. db2darray

db2darray function is an example for loading a set of unlabeled samples stored in a table to a darray. It is assumed that samples are stored in a single table, and the table contains a column called 'rowid'. It is also assumed that 'rowid' starts from 0, and there is no missed 'rowid'. There is an optional argument to specify the desired number of splits (partitions) in the distributed array. However, the number of partitions at the result might not be exactly the same as desired number, but it will be close to it. The final number of partitions can be determined by the following routine:

rowsInBlock <- ceiling(nobs/nSplits), where nobs is the number of observations (number of rows) in the table, and nSplits is the specified number of splits. As for building the distributed data structure, the value of rowsInBlock will be used, the final number of partitions will be ceiling(nobs/rowsInBlock) which clearly might be different from nSplits. A warning message will show the selected number of splits when it is different from the desired number. The default value for nSplits when it is not specified is the number of active R-executors in the distributedR environment.

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

- **Example**

As an example, assume that samples are stored in a table named *mortgage*; while 6 columns should be used for loading to a darray: *mltvspline1*, *mltvspline2*, *agespline1*, *agespline2*, *hpichgspline*, and *ficospline* (see Table 1). Also assume that the name of configuration for the database connection is "*Test"*. The command to load all the samples stored in this table is:

```
loadedSamples <- db2darray ("mortgage", list("mltvspline1", "mltvspline2",
"agespline1", "agespline2", "hpichgspline", "ficospline"), conf="Test")
```

The darray for samples will become available as `loadedSamples` (see Figure 1). Partitions (blocks) of the darray are row-wise and their number is the same as the number of active instances (R-executors) when nSplits is not specified.

Table 1. Example of a table in the database called mortgage

rowid	def	mltvspline1	mltvspline2	agespline1	agespline2	hpichgspline	ficospline
0	1	0.760777	0.006632	0.948052	0.906403	0.058021	0.960328
1	0	0.135741	0.205449	0.516031	0.013455	0.827438	0.659125
2	0	0.021796	0.138996	0.862165	0.034211	0.150524	0.345917
3	1	0.271257	0.543280	0.940978	0.891880	0.993050	0.000160
4	1	0.986207	0.053896	0.119611	0.646744	0.819753	0.663289

```
> loadedSamples <- db2darray ("mortgage", list("mltvspline1", "mltvspline2", "agespline1",
"agespline2", "hpichgspline", "ficospline"), conf="Test")
progress: 100%
> getpartition(loadedSamples)
     mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
[1,]    0.760777    0.006632   0.948052   0.906403     0.058021   0.960328
[2,]    0.135741    0.205449   0.516031   0.013455     0.827438   0.659125
[3,]    0.021796    0.138996   0.862165   0.034211     0.150524   0.345917
[4,]    0.271257    0.543280   0.940978   0.891880     0.993050   0.000160
[5,]    0.986207    0.053896   0.119611   0.646744     0.819753   0.663289
```

Figure 1. An example for db2darray

3. db2darrays

The db2darrays function loads a dataset (a set of samples) from a table in HP Vertica to a pair of R darrays which correspond to responses and predictors of a predictive model. The samples stored in the database (including responses and predictors) must be stored in a single table, and the table must contain a column called rowid. The column rowid must start from 0, and be contiguous across all values in the column. For example, rowid for a table with five rows would contain the values 0, 1, 2, 3, 4 (see the example).

Similar to db2darray, there is an optional input argument called nSplits to specify the desired number of splits. The behavior of the function for selecting the final number of partitions is similar to the behavior of db2darray function which was explained in the previous section.

- **Example**

As an example, assume that samples are stored in a table named *mortgage*; the name of response column is *def*, and the names of predictive columns are *mltvspline1*, *mltvspline2*, *agespline1*, *agespline2*, *hpichgspline*, and *ficospline* (see Table 1). Also assume that the name of configuration for the database connection is "*Test"*. The command to load all the samples stored in this table is to a pair of darrays with 2 partitions is:

```
loadedData <- db2darrays ("mortgage", list("def"), list("mltvspline1", "mltvspline2",
"agespline1", "agespline2", "hpichgspline", "ficospline"), conf="Test")
```

The darrays for response and predictors are available as `loadedData$Y` and `loadedData$X` respectively (see Figure 2). The number of blocks and their size are exactly the same in both darrays. In fact, each row in Y corresponds to the response of a row in X with exactly the same row position. Moreover, blocks are row-wise partitioned.

```
> loadedData <- db2darrays ("mortgage", list("def"), list("mltvspline1", "mltvspline2",
"agespline1", "agespline2", "hpichgspline", "ficospline")"), conf="Test", nSplits=2)
progress: 100%
> names(loadedData)
[1] "Y" "X"
> getpartition(loadedData$X)
      mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
 [1,]    0.760777    0.006632   0.948052   0.906403     0.058021   0.960328
 [2,]    0.135741    0.205449   0.516031   0.013455     0.827438   0.659125
 [3,]    0.021796    0.138996   0.862165   0.034211     0.150524   0.345917
 [4,]    0.271257    0.543280   0.940978   0.891880     0.993050   0.000160
 [5,]    0.986207    0.053896   0.119611   0.646744     0.819753   0.663289
```

Figure 2. An example for db2darrays

4. db2matrix

db2matrix function is an example for loading a set of unlabeled samples stored in a table to a matrix. It is assumed that samples are stored in a single table. All the rows of the table will be read, and each row will be a sample.

- **Example**

Let's assume that two samples are stored in a table called *centers*, and the table has 6 columns corresponding to different features of the samples. Each row of this table is read as a sample. Table 2 and Figure 3 show an example for this function.

Table 2. Example of a table in the database for db2matrix

mltvspline1	mltvspline2	agespline1	agespline2	hpichgspline	ficospline
0.76	0.006	0.948	0.906	0.058	0.960
0.135	0.205	0.516	0.013	0.827	0.659

```
> loadedCenters <- db2matrix ("centers", list("mltvspline1", "mltvspline2", "agespline1",
"agespline2", "hpichgspline", "ficospline"), conf="Test")
> loadedCenters
      mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
 [1,]    0.76        0.006      0.948      0.906        0.058      0.960328
 [2,]    0.135       0.205      0.516      0.013        0.827      0.659125
```

Figure 3. An example for db2matrix

5. db2dframe

db2dframe function is an example for loading a set of samples stored in a table to a dframe. It is assumed that samples are stored in a single table, and the table contains a column called 'rowid'. It is also assumed that 'rowid' starts from 0, and there is no missed 'rowid'.

Similar to db2darray, there is an optional input argument called nSplits to specify the desired number of splits. The behavior of the function for selecting the final number of partitions is similar to the behavior of db2darray function which was explained in the previous section.

- **Example**

db2dframe function is very similar to db2array except that it stores data in a dframe data structure instead of darray; consequently, it can store also categorical data. Let's consider a table in database similar to the previous examples, but strings *"Yes"* and *"No"* for the value of its *def* column (see Table 3).

Table 3. Example of a table in the database with categorical data

rowid	def	mltvspline1	mltvspline2	agespline1	agespline2	hpichgspline	ficospline
0	Yes	0.760777	0.006632	0.948052	0.906403	0.058021	0.960328
1	No	0.135741	0.205449	0.516031	0.013455	0.827438	0.659125
2	No	0.021796	0.138996	0.862165	0.034211	0.150524	0.345917
3	Yes	0.271257	0.543280	0.940978	0.891880	0.993050	0.000160
4	Yes	0.986207	0.053896	0.119611	0.646744	0.819753	0.663289

Figure 4 shows an example for loading the table into a dframe.

```
> loadedSamples <- db2dframe ("mortgage", list("def", "mltvspline1", "mltvspline2",
"agespline1", "agespline2", "hpichgspline", "ficospline"), conf="Test")
progress: 100%
> is.dframe(loadedSamples)
[1] TRUE
> getpartition(loadedSamples)
      def    mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
 1     Yes    0.760777    0.006632    0.948052   0.906403     0.058021   0.960328
 2     No     0.135741    0.205449    0.516031   0.013455     0.827438   0.659125
 3     No     0.021796    0.138996    0.862165   0.034211     0.150524   0.345917
 4     Yes    0.271257    0.543280    0.940978   0.891880     0.993050   0.000160
 5     Yes    0.986207    0.053896    0.119611   0.646744     0.819753   0.663289
```

Figure 4. An example for db2dframe

6. db2dgraph

db2dgraph function is an example for loading an adjacency matrix from a table of edge-list stored in a database. It is assumed that edge-list is stored in a single table. It is also assumed that the ID of vertices in the table starts from zero. As the returned darray will be column-wise partitioned, it would be more efficient when there is projection (index) on the 'to' column of the table.

Similar to db2darray, there is an optional input argument called nSplits to specify the desired number of splits. Therefore, the number of splits in the result might be different from the requested number.

The ID of the vertices starts from 0. The maximum ID number will be used to find the number of vertices in the graph. All missed ID numbers between the first ID (0) and the maximum ID will be assumed as vertices without any connected edge.

- **Example**

Let's assume that there is a weighted graph stored in the *wgraph* table of the database similar to

Table 4. Figure 5 shows how db2dgraph function can be used to read the table into a pair of sparse darrays.

Table 4. An example of a weighted graph in the database

x	y	w
8	7	2.5
2	6	3.1
3	5	1.2
4	7	2.2

```
> dg <- db2dgraph (tableName="wgraph", from="x", to="y", weight="w", conf="Test")

progress: 100%

> getpartitio(dg$X)

[1] . . . . . .
[2] . . . . . .
[3] . . . . . 1 .
[4] . . . . 1 . .
[5] . . . . . 1 .
[6] . . . . . .
[7] . . . . . .
[8] . . . . . .
[9] . . . . . 1 .

> getpartition(dg$W)

[1] . . . . . .   .   .
[2] . . . . . .   .   .
[3] . . . . . .  3.1  .
[4] . . . . . 1.2 .   .
[5] . . . . . .   .  2.2 .
[6] . . . . . .   .   .
[7] . . . . . .   .   .
[8] . . . . . .   .   .
[9] . . . . . .   .  2.5 .
```

Figure 5. An example for db2dgraph

7. splitGraphFile

splitGraphFile function is an example for splitting an edge-list file of a graph, and distributing the results among the active nodes of a cluster system. The result of this function can be fed to file2dgraph function to create a darray for adjacency matrix. The input edge-list is assumed zero-based; i.e., the id of the first edge is zero. The total number of vertices will be calculated according to the maximum vertex id

found in the input file. Moreover, a dummy vertex will be assumed for any missed vertex id; i.e., a vertex with no connected edge.

The function first creates a temporary folder at the same path of the input file for generating split files, and then sends the files to the nodes of the clusters where each an active worker is available in the distributedR environment. Therefore, the account of the user who calls this function must have read and write access to both input path and output path.

The number of files at the result might not be exactly the same as requested number, but it will be close to it. The final number of files can be determined by the following routine. Assuming that a positive integer number is specified for *nSplits*, the number of vertices in each split file will be calculated as *verticesInSplit <- ceiling(nVertices/nSplits)* where *nVertices* is the number of vertices in the input graph. Then, the final number of files will be *nFiles <- ceiling(nVertices/verticesInSplit)* which might be different from the input *nSplits*. The files will be indexed from *0* to *(nFiles – 1)*.

- **Example**

Assume that file small.txt contains edge-list of a small directed graph (see Figure 6-a), and is located in the local address "/home/verticaUser/graphSource". Let's also assume that we want to split this file into 2 partitions and store the resulted files in an NFS directory with this path: "/graphSplits". The process is displayed in Figure 7. The name of the split files will be small.txt0, small.txt2. The content of these files is also displayed in Figure 6-b&c.

```
# The original
# small graph
0 1
1 2
2 6
2 7
3 4
3 6
5 3
5 4
6 7
7 1
```

a) small.txt

```
0 1
1 2
5 3
7 1
```

b) small.txt0

```
2 6
2 7
3 4
3 6
5 4
6 7
```

c) small.txt1

Figure 6. An example for splitting a file

```
> ret <- splitGraphFile(inputFile="/home/verticaUser/GraphSource/small.txt",

outputPath="/graphSplits", nSplits=2, isNFS=TRUE)

> ret

$pathPrefix
[1] "/graphSplits/small.txt"

$nVertices
[1] 8

$verticesInSplit
[1] 4

$nFiles
[1] 2

$isWeighted
[1] FALSE
```

Figure 7. An example for splitGraphFile

8. file2dgraph

file2dgraph function is an example for loading an adjacency matrix from an edge-list which is split in a set of files. It is assumed that edge-list is properly split among nfiles. Split files should correspond to column-wise partitioned adjacency matrix; i.e., each file will be used to load one partition of such a matrix. The files should be available from the same path from all the nodes; i.e., they should be located either on NFS or on the same path in different nodes of the cluster. Moreover, *isWeighted* input argument should be set properly according to the graph stored in the files.

- **Example**

The output of splitGraphFile function can be fed to the input arguments of file2dgraph function. Let's use the results of the previous example to create the distributed adjacency matrix. The procedure is displayed in Figure 8. Dimensions of the resulted darray will be equal to the number of vertices. Moreover, the sum of its elements will be equal to the number of edges because each edge is represented with an element of value 1.

```
> dg <- file2dgraph(pathPrefix=ret$pathPrefix, nVertices=ret$nVertices,

verticesInSplit=ret$verticesInSplit, isWeighted=ret$isWeighted)

Openning files:
/graphSplits/small.txt0
…until …
/graphSplits/small.txt1
Progress: 100%

> getpartition(dg$X)
8 * 8 sparse Matrix of class "dgCMatrix"

[1,] . 1 . . . . . .
[2,] . . 1 . . . . .
[3,] . . . . . . 1 1
[4,] . . . . 1 . 1 .
[5,] . . . . . . . .
[6,] . . . 1 1 . . .
[7,] . . . . . . . 1
[8,] . 1 . . . . . .

> dg$W
NULL

> dim(dg$X)
[1] 8 8

> sum(dg$X)
[1] 10
```

Figure 8. An example for file2dgraph