Package 'HPdata'

August 1, 2014

Type Package

Title Distributed Data Package

Version 0.7.0

Date 2014-01-22

Author Arash Fard

Maintainer Arash Fard <afard@vertica.com>

Depends R (>= 3.0.1), distributedR, MatrixHelper

LinkingTo Rcpp

Description It provides several functions to load distributed data structures available in distributedR. It is written based on the infrastructure created in HP-Lab for distributed computing in R.

License GPL (>= 2) | file LICENSE

R topics documented:

`HPdata-package` *Distributed Data Package*

Description

HPdata 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.

Details

Package:	HPdata
Type:	Package
Version:	0.7.0
Date:	2014-01-22

Main Functions:

- 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.
- file2dgraph: It loads an adjaceny matrix to a darray from an edgelist strored in a set of files.
- 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.

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.

`db2darray` *A simple loader for darray*

Description

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'.

Usage

```
db2darray(tableName, features = list(...), conf, nSplits)
```

Arguments

`tableName`	it is the name of the table in the database in string format.
`features`	the list of the name of columns corresponding to the features of a sample.
`conf`	the name of configuration in ODBC.INI file for connecting to the database.
`nSplits`	this optional argument specifies the desired number of splits (partitions) in the distributed array. When it is not specified, it will become equal to the number of active instances in the distributed R environment. It should be noted that the number of splits in the returned result might not be exactly the same as nSplits. Please read the details for more information.

Details

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. When a positive integer number is specified for nSplits, the number of samples in each partition of the distributed structure will be calculated as rowsInBlock <- ceiling(nobs/nSplits) where nobs is the number of observations (number of rows) in the table. 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.

Value

X	the darray of samples

Author(s)

Arash Fard <afard@vertica.com>

Examples

```
 ## Not run:
    require(HPdata)
    distributedR_start()
    # Assuming that samples are stored in a table named "samples",
    # and the names of the columns are "col1", "col2", "col3", and "col4".
    loadedSamples <- db2darray ("samples", list("col1", "col2",
    "col3", "col4"), conf="RDev")

## End(Not run)
```

---

db2darrays *A Simple Data Loader which loads two aligned darrays from Database*

Description

db2darrays function is a simple function to load a labeled dataset (a set of labeled samples) from a table in Vertica database to a pair of darrays which correspond to responses and predictors of a predictive model. It is assumed that samples (including responses and predictors) 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'.

Usage

```
db2darrays(tableName, resp = list(...), pred = list(...), conf,
            nSplits)
```

Arguments

tableName	it is the name of the table in the database in string format.
resp	the list of the name of columns corresponding to responses.
pred	the list of the name of columns corresponding to predictors.
conf	the name of configuration in ODBC.INI file for connecting to the database.

`nSplits` this optional argument specifies the desired number of splits (partitions) in the distributed arrays. When it is not specified, it will become equal to the number of active instances in the distributed R environment. It should be noted that the number of splits in the returned result might not be exactly the same as nSplits. Please read the details for more information.

Details

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. When a positive integer number is specified for nSplits, the number of samples in each partition of the distributed structure will be calculated as rowsInBlock <- ceiling(nobs/nSplits) where nobs is the number of observations (number of rows) in the table. 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.

Value

`Y` the darray of responses

`X` the darray of predictors

Author(s)

Arash Fard

Examples

```
## Not run:
   require(HPdata)
   distributedR_start()
   # Assuming 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.
   loadedData <- db2darrays ("mortgage", list("def"), list("mltvspline1",
   "mltvspline2", "agespline1", "agespline2", "hpichgspline",
   "ficospline"), conf="RDev")

## End(Not run)
```

---

db2dframe *A simple loader for dframe*

Description

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'.

Usage

```
db2dframe(tableName, features = list(...), conf, nSplits)
```

Arguments

tableName	it is the name of the table in the database in string format.
features	the list of the name of columns corresponding to the features of a sample.
conf	the name of configuration in ODBC.INI file for connecting to the database.
nSplits	this optional argument specifies the desired number of splits (partitions) in the distributed frame. When it is not specified, it will become equal to the number of active instances in the distributed R environment. It should be noted that the number of splits in the returned result might not be exactly the same as nSplits. Please read the details for more information.

Details

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. When a positive integer number is specified for nSplits, the number of samples in each partition of the distributed structure will be calculated as rowsInBlock <- ceiling(nobs/nSplits) where nobs is the number of observations (number of rows) in the table. 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.

Value

X	the dframe of samples

Author(s)

Arash Fard <afard@vertica.com>

Examples

```
## Not run:
   require(HPdata)
   distributedR_start()
   # Assuming that samples are stored in a table named "samples",
   # and the names of the columns are "col1", "col2", "col3", and "col4".
   loadedSamples <- db2dframe ("samples", list("col1", "col2",
   "col3", "col4"), conf="RDev")

## End(Not run)
```

---

db2dgraph *A simple graph loader from a database*

Description

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.

Usage

```
db2dgraph(tableName, from, to, conf, weight, nSplits)
```

Arguments

tableName	it is the name of the table in the database in string format.
from	the name of the column in the table which stores the source vertices
to	the name of the column in the table which stores the target vertices
conf	the name of configuration in ODBC.INI file for connecting to the database
weight	the name of the column in the table which stores the weights of the edges (it is an optional argument)
nSplits	this optional argument specifies the desired number of splits (partitions) in the distributed array. When it is not specified, it will become equal to the number of active instances in the distributed R environment. It should be noted that the number of splits in the returned result might not be exactly the same as nSplits. Please read the details for more information.

Details

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. When a positive integer number is specified for nSplits, the number of samples in each partition of the distributed structure will be calculated as verticesInSplit <- ceiling(nVertices/nSplits) where nVertices is the number of vertices. As for building the distributed data structure, the value of verticesInSplit will be used, the final number of partitions will be ceiling(nVertices/verticesInSplit) which might be different from nSplits.

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.

Value

X	the sparse darray of adjacency matrix
W	the sparse darray of weights on edges

Author(s)

Arash Fard <afard@vertica.com>

Examples

```
 ## Not run:
    require(HPdata)
    distributedR_start()
    # Assuming that unweighted edge-list is stored in a table named "graph",
    # and the names of the columns are "x", "y".
    dgraphDB <- db2dgraph("graph", from="x", to="y", conf="RDev")

## End(Not run)
```

db2matrix *A simple loader for loading a matrix from a database*

Description

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.

Usage

```
db2matrix(tableName, features = list(...), conf)
```

Arguments

tableName	it is the name of the table in the database in string format.
features	the list of the name of columns corresponding to the features of a sample.
conf	the name of configuration in ODBC.INI file for connecting to the database.

Value

X	the matrix

Author(s)

Arash Fard <afard@vertica.com>

Examples

```
 ## Not run:
    # Assuming that centers are stored in a table named "centers",
    # and the names of the columns are "col1", "col2", "col3", and "col4".
    loadedCenters <- db2matrix ("centers", list("col1", "col2",
    "col3", "col4"), conf="RDev")

## End(Not run)
```

file2dgraph *A simple graph loader from a set of files*

Description

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.

Usage

```
file2dgraph(pathPrefix, nVertices, verticesInSplit, isWeighted)
```

Arguments

pathPrefix	the path and prefix to the files. It should be an absolute path and the same on all nodes of the cluster.
nVertices	the total number of vertices in the graph
verticesInSplit	the number of vertices considered in each split file
isWeighted	it should be FALSE when the there is no weight information in the input files

Details

The split files should be partitioned edge-lists of a graph. The vertices should be partitioned among the file in such a way that nFiles == ceiling(nVertices / verticesInSplit) where nFiles is the number of files, nVertices is the total number of vertices, and verticesInSplit is the number of vertice in each file. The last file may have less vertices. The ID of the vertices starts from 0. The maximum ID number must be equal to `nVertices - 1`. All missed ID numbers between the first ID (0) and the maximum ID will be assumed as vertices without any connected edge.

`isWeighted` must be carefully set TRUE or FALSE. Either all the edges should be weighed or none of them. When there is no weight, the return value for `W` will be `NULL`.

Any comment or extra character in the lines will generate error.

All the nodes of the cluster should have access to all the split files, whether they are on a shared storage or they are distributed among the nodes. Nevertheless, only a portion of split files will be used by R-executors of a node. When the files are distributed among the nodes, file2dgraph function will be robust against corruption of the files which are not used by any R-executors. It should be noticed that this function does not detect the possible inconsistency of the distributed files, but it is assumed that they are the same from point of view of all the nodes.

Value

X	the sparse darray of adjacency matrix
W	the sparse darray of weights on edges

Author(s)

Arash Fard <afard@vertica.com>

Examples

```
## Not run:
   require(HPdata)
   distributedR_start()

   # Wiki-Vote graph can be found in SNAP's repository
   # http://snap.stanford.edu/data/index.html
   ret <- splitGraphFile(inputFile="
               /home/afard/Downloads/graph/Wiki-Vote.txt",
               nSplits=7, outputPath="/home/afard/temp/test", isNFS=TRUE)

   dgraphF <- file2dgraph(ret$pathPrefix, ret$nVertices,
```

```
                                ret$verticesInSplit, ret$isWeighted)

## End(Not run)
```

`splitGraphFile` *A simple graph splitter*

Description

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

Usage

```
splitGraphFile(inputFile, nSplits, outputPath, isNFS = FALSE)
```

Arguments

`inputFile`	the input file which contains the complete adjacency list of a graph. An absolute path should be used.
`nSplits`	the desired number of partitions in the graph. It may change based on the total number of vertices. It will be translated to the number of split files.
`outputPath`	it is the path for storing the output split files, it should be similar for master and workers. An absolute path should be used.
`isNFS`	TRUE indicates that outputPath is on NFS; therefore, only one copy of files will be sent to the destination. When it is FALSE (defualt) a copy of the files will be sent to every active node in the pool.

Details

Master should have read and write access to `inputPath`. Moreover, it should have write access to `outputPath`. When `outputPath` is not on NFS (the default assumption), Master distributes the files among all the workers which are active in the distributedR environment. `outputPath` should be already available on all the nodes of the cluster.

The input file should be a text file containing the edge-list of a graph. Each edge should appear in a new line. Each edge should be represented by two integer numbers. The first integer would be the ID of the source vertex, and the second integer would be the ID of the target vertex. Edges can have floating point numbers for their weights which should appear right after the target vertex. These numbers should appear at the beginning of a line and delimited by space. Anything after these numbers will be ignored. Moreover, any line which starts with a non-number word will be assumed as a comment line and will be ignored. Existence of weight on the first edge determines if it the graph is weighted or not. Therefore, if the first edge appeared in the file does not have weight, weights on any other edges will be ignored. On the other hand, when the first edge has a weight, all other edges will be assumed weighted. In this case, the weight of the edges where it is not explicitly specified would be 1.

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.

The number of files at the result might not be exactly the same as desired number, but it will be close to it. The final number of files can be determined by the following routine. When 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 nSplits. The files will be indexed from `0` to `nFiles - 1`.

Value

`pathPrefix`	the prefix name of the split files
`nVertices`	the total number of vertices in the graph
`verticesInSplit`	the number of vertices considered in each split file
`nFiles`	the number of split files
`isWeighted`	it indicates that there is weight on edges when it is TRUE

Author(s)

Arash Fard <afard@vertica.com>

Examples

```
## Not run:
   require(HPdata)
   distributedR_start()

   # Wiki-Vote graph can be found in SNAP's repository
   # http://snap.stanford.edu/data/index.html
   ret <- splitGraphFile(inputFile=
           "/graphs/Wiki-Vote.txt", nSplits=7,
           outputPath="/home/afard/temp/test", isNFS=TRUE)

## End(Not run)
```

Index