Oracle R Advanced Analytics for Hadoop 2.7.0

November 23, 2016

`hadoop.exec` *Executes mapReduce functions written in R on Hadoop cluster.*

Description

Invokes Hadoop engine and sends mapper and reducer R functions for execution. If input data does not reside in HDFS then first copies data into HDFS. Prepares user's mapReduce scripts for execution in distributed Hadoop environment and invokes Hadoop engine monitoring its log for errors and failures.

Usage

```
hadoop.exec(dfs.id, out.name = NULL, mapper = NULL,
  reducer = NULL, combiner = NULL, export = NULL,
  init = NULL, final = NULL, job.name = NULL,
  config = NULL, cleanup = FALSE, overwrite = FALSE,
  attach = TRUE, tmp.result = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier of the input data. This is a special ORCH object returned by hdfs.attach and other functions which represents a directory in HDFS. Or it can be a string with HDFS-compliant path relative to the current working directory.
`out.name`	Output HDFS directory name or an HDFS object identifier of the output data. Note that the output directory must not exist when Hadoop job is submitted otherwise the job will fail. If the output directory is not specified a temporary one will be created in HDFS "/tmp".
`mapper`	Optional mapper function written in the R language. The function must accept two values: "key" and "value", names of the arguments do not matter. Prototype is: mapper = function(k,v) . If mapper function is not specified or NULL then reduce-only job will be executed.
`combiner`	Optional combiner function written in the R language. The function must accept two values: "key" and "value", names of the arguments do not matter. Prototype is: reducer = function(k,v) .

`reducer`	Optional reducer function written in the R language. The function must accept two values: "key" and "value", names of the arguments do not matter. Prototype is: reducer = function(k,v) . If reducer function is not specified or NULL then map-only job will be executed.
`export`	Exported R objects. This argument copies the specified R objects (e.g. from R user's R session) into server side of running mapReduce jobs (e.g. Hadoop cluster side so it is available to the mapReduce jobs during execution). See orch.export and examples for more details.
`init`	Optional job initialization function. Called once before any user's mapReduce functions and allows the user to do any initial preparation, initialization, or memory allocation required for map or reduce functions logic. The function should not accept any arguments. Prototype is: init = function() .
`final`	Optional job finalization function. Called once after all user's mapReduce functions and allows the user to do final data processing, or memory de-allocation required by mapReduce logic. It is allowed to output keyValues in final function too, see orch.keyvals. The function should not accept any arguments. Prototype is: final = function() .
`job.name`	Optional name of this mapReduce job. By default Hadoop's job ID will be used as the job name. It is advised to always give some meaningful name to facilitate locating your job in Hadoop run logs.
`config`	Optional mapReduce advanced configuration class. This argument allows the user to fine-tune various aspects of mapReduce job in order to achieve better performance or change behavior of ORCH mapReduce driver. This argument is an instance of the "mapred.config" class, and so it has this format: config = new("mapred.config", param1, param2,...).
`cleanup`	Will run a cleanup procedure after the mapReduce job is finished succesfully which will remove all empty "part" files and all Hadoop log files.
`overwrite`	Allows overwriting of HDFS objects with the same name. By default overwrite is disabled for safety of data.
`attach`	Enable or disable automatic attachment of the result of the Hadoop job. If disabled then the returned HDFS object identifier will be pointing to HDFS directory without ORCH metadata.
`tmp.result`	This argument tell the function that mapReduce job result is not final, is not intended to be returned to the user, and will not be used between R sessions. The result is temporary and will be removed in this R session. This option disables writing of ORCH metadata to HDFS and keeps it in memory cached only.

Details

This function provides core functionality for Hadoop MapReduce execution. It does not provide any data management and conversion facilities and requires that data is already present in HDFS before execution. Input can only be an HDFS object and results are stored back to HDFS only and never converted back into original input data formats as it's done in hadoop.run.

Compared to hadoop.run, this function is designed to be used for optimization of multi-stage mapReduce jobs when output of this job is not the final result and will be used as input for the next stage. It lowers the overhead of data conversion and management procedures when it's not required by an R workflow.

Value

Resulting HDFS object identifier if everything worked correctly, otherwise returns NULL if execution has failed for any reason.

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

hadoop.run mapred.config orch.dryrun orch.debug orch.keyval orch.keyvals hdfs.put hdfs.push hdfs.upload

Examples

```
# Filter cars with with "dist" > 30 and "speed" > 14 in mapper
# and get mean "speed" and "dist" in reducer.

# Put cars data in HDFS
cars.dfs <- hdfs.put(cars)
x <- hadoop.exec(
    cars.dfs,
    mapper = function(key, val) {
        for (i in 1:nrow(val)) {
            x <- val[i,]
            if (x$dist > 30 && x$speed > 14) {
                orch.keyval(key[i], x)
            }
        }
    },
    reducer = function(key, vals) {
        orch.keyval(key, c(mean(vals$speed), mean(vals$dist)))
    },
    config = new("mapred.config",
        map.tasks = 1,
        reduce.tasks = 1
        )
)

# Get result in R
res <- hdfs.get(x)
print(res)
# Cleanup
hdfs.rm(cars.dfs)
```

`hadoop.jobs` *Allows to inspect the Hadoop cluster load.*

Description

Allows to inspect the Hadoop cluster load.

Usage

```
hadoop.jobs(verbose = FALSE)
```

Arguments

`verbose`	If FALSE then returns a limited set of information about running jobs: • JobId: Hadoop job name as specified by a user. • State: Job state, normally "RUNNING". • StartTime: When the job was started. • UserName: Job owner name. • Priority: Job priority. If TRUE then returns all job attributes as they are returned by the presently running version of Hadoop. This also means that attributes and their names can differ depending on the Hadoop version.

Value

List of running jobs and their attributes as a data.frame object. Refer to `verbose` for more information about the returned value content.

`hadoop.run` *Executes mapReduce functions written in R on Hadoop cluster.*

Description

Invokes Hadoop engine and sends mapper and reducer R functions for execution. If input data does not reside in HDFS then first copies data into HDFS. Prepares user's mapReduce scripts for execution in distributed Hadoop environment and invokes Hadoop engine monitoring the Hadoop log for errors and failures. If execution was successful then reads data back from HDFS into R memory if input data was in-memory R object, or pushes it back to Oracle database or Hive depending on where original data is located.

Usage

```
hadoop.run(data, out.name = NULL, mapper = NULL,
  reducer = NULL, combiner = NULL, export = NULL,
  init = NULL, final = NULL, job.name = NULL,
  config = NULL, cleanup = FALSE, overwrite = FALSE)
```

Arguments

`data`	Input data object, can be one of the following types: • ORCH HDFS object identifier This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS-compliant directory path relative to the current working directory. • Oracle R Enterprise frame ore.frame Both RDBMS and HIVE tables/views exposed as ore.frame objects are accepted. • Object of R class "data.frame" • Object of R class "matrix" • Object of R class "list" • Object of R class "vector"
`out.name`	Output HDFS directory name or an HDFS object identifier of the output data. Note that the output directory must not exist when Hadoop job is submitted otherwise the job will fail. If the output directory is not specified a temporary one will be created in HDFS "/tmp".
`mapper`	Optional mapper function written in the R language. The function must accept two values: "key" and "value", names of the arguments do not matter. Prototype is: mapper = function(k,v) . If mapper function is not specified or NULL then reduce-only job will be executed.
`combiner`	Optional combiner function written in the R language. The function must accept two values: "key" and "value", names of the arguments do not matter. Prototype is: reducer = function(k,v) . Combiner function gets executed on the same host as each mapper and receives the same data as reducer but from each local individual mapper. Its output is the same as mapper output and will be feed to next reduce stage. Note that combiner can not be used without reduce and will be ignored at this case if its specified
`reducer`	Optional reducer function written in the R language. The function must accept two values: "key" and "value", names of the arguments do not matter. Prototype is: reducer = function(k,v) . If reducer function is not specified or NULL then map-only job will be executed.
`export`	Exported R objects. This argument allows the user to copy some of client side R objects (e.g. from R user's R session) into server side of running mapReduce jobs (e.g. Hadoop cluster side so it is available to the mapReduce jobs during execution). See orch.export and examples for more details.
`init`	Optional job initialization function. Called once before any user's mapReduce functions and allows to do any initial preparation, initialization, or memory allocation required for map or reduce functions logic. The function should not accept any arguments. Prototype is: init = function() .
`final`	Optional job finalization function. Called once after all user's mapReduce functions and allows to do final data processing, or memory de-allocation required by mapReduce logic. It is allowed to output keyValues in final function too, see orch.keyvals. The function should not accept any arguments. Prototype is: final = function() .
`job.name`	Optional name of this mapReduce job. By default Hadoop's job ID will be used as a job name. It is advised to always give some meaningful name to easier locate your job in Hadoop run logs. Note that this argument, if used, overrides job.name in `config`.

`config`	Optional mapReduce advanced configuration class. This argument allows to fine-tune various aspects of mapReduce job and in order to achieve better performance or change behavior of ORCH mapReduce driver. This argument is an instance of the "mapred.config" class, and so it has this format: config = new("mapred.config", param1, param2,...).
`cleanup`	Will run a cleanup procedure after the mapReduce job is finished succesfully which will remove all empty "part" files and all Hadoop log files.
`overwrite`	Allows overwriting of HDFS objects with the same name. By default overwrite is disabled for safety of data.

Value

Result in the same format as input data. For example, the results for HDFS input data are kept in HDFS, and the results for ore.frame input data are copied into the connected database. If any error or failur occured during execution which prevented successful output of the result then will be returned.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hadoop.exec mapred.config orch.dryrun orch.debug orch.keyval orch.keyvals hdfs.put hdfs.push hdfs.upload

Examples

```
# Filter cars with with "dist" > 30 and "speed" > 14 in mapper
# and get mean "speed" and "dist" in reducer.

# Use cars dataset from R memory
x <- hadoop.run(
   cars,
    mapper = function(key, val) {
        for (i in 1:nrow(val)) {
            x <- val[i,]
            if (x$dist > 30 && x$speed > 14) {
                orch.keyval(key[i], x)
            }
        }
    },
    reducer = function(key, vals) {
        orch.keyval(key, c(mean(vals$speed), mean(vals$dist)))
    },
    config = new("mapred.config",
        map.tasks = 1,
        reduce.tasks = 1
        )
)

# See Result
print(x)
```

`hdfs.attach` *Brings an HDFS object into ORCH environment.*

Description

Attaches "unmanaged" HDFS files in a directory to ORCH framework by loading the metadata describing contents of the file (number, types and names of data columns in the files in the directory) if the metadata is already present or by discovering metadata of the file by intelligent sampling of file contents. As a result returns HDFS object identifier of the HDFS attached directory if successful else NULL if metadata for the files in the HDFS directory could not be determined.

Usage

```
hdfs.attach(dfs.name, key.sep = .orch.env$key.sep,
  value.sep = .orch.env$val.sep, key = NULL,
  force = FALSE, trim = FALSE, data.frame = FALSE,
  na.strings = NULL, silent = FALSE)
```

Arguments

`dfs.name`	HDFS directory name or HDFS path relative to the current working directory. Alternatively the user can use an HDFS object identifier returned by hdfs.attach() from a prior invocation in case if it need to be re-attached (see `force` argument).
`key.sep`	Key field separator character, "\t" is system default. If key separator is specified incorrectly then key field will be concatenated with the first value field and there will be **no** key identified. Key separator can have the same value as value separator.
`value.sep`	Value field separator character, "," is system default. If key separator is specified incorrectly then all value fields will be concatenated together. Value separator can have the same value as key separator.
`key`	Key column index, NULL = auto-detect. The key column in HDFS has to be the first one but it can be mapped to any column in the original data. This value controls key column position in a data.frame when ORCH reads or samples the data.
`force`	TRUE to overwrite HDFS object metadata. If the HDFS object was previously attached and has metadata stored alongside already this arguments allows to re-attach it again.
`trim`	TRUE to ignore tailing empty fields. In case if HDFS data is suspected to have empty tailing columns like ",,," this option allows to detect and exclude such redundant columns for the data description in metadata and its structure.
`data.frame`	If TRUE enforces the class of the attached HDFS data to be "data.frame". Otherwise the class can be automatically recognized as "vector", or "matrix", or "data.frame".

`na.strings`	Character vector with strings that represent NA values in the attached data set. If this argument is not specified then "NA" and "" strings will be treated as NA values by default.
`silent`	Do not print information messages to console. Do not print final attach summary at the end of the run.

Details

By default, data files in HDFS are not usable in ORCH until they are attached and ORCH knows the structure of data in the files. Note that to use files in ORCH the user must first place them in a separate HDFS directory. The path to the directory should be specified as input to hdfs.attach(). If the data does not have ORCH metadata stored with it then ORCH samples portions of the data from the file(s) in this directory, parses them and determines the data structure. ORCH then generates a special metadata object that contains the discovered structure with ORCH-specific system data and stores it alongside with the original data in a new file called __ORCHMETA__.

If data has non-standard format (non-comma delimited) delimiters must be specified as a "hint" via argument `key.sep` and `value.sep`. ORCH will create HDFS object's metadata with the user specified delimiters stored in there, the content of the HDFS object attached will not be changed in any way. If you specify incorrect set of delimiters then attach may fail. If you do not specify the delimiters then the current defaults ("\t" for key delimiter and "," for values delimiter) will be used.

hdfs.attach() will create a new __ORCHMETA__ file (if not already present) in the same directory from where files are loaded into ORCH environment. This file contains metadata for the data files.

Value

HDFS object identifier if HDFS data was attached successfully, otherwise NULL if transfer or data structure recognition error occurred.

Note

Use this function to attach a text file to your R environment, just as you might attach a data.frame. Oracle R Connector for Hadoop does not support processing of attached non-structured files. Nonetheless, you can attach a non-structured file, download it to your local computer, and use it as desired. Alternatively, you can attach the file for use as input to a Hadoop application.

The function performance may drop when attaching large HDFS files with long records due to inherent limitations of the Hadoop command-line interface. When size of one record is larger then 1KB then sampling will fall back to streaming larger parts of HDFS files to retrieve several full records with valid structure. If input data contains many invalid or incomplete records the function may try to resample larger portions of the input data set in order to discover the structure.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.exists hdfs.ls hdfs.put hdfs.get hdfs.describe hdfs.meta

Examples

```
# Upload cars to HDFS
dfs1 <- hdfs.put(cars, dfs.name="cars_w_meta")

# Write cars data to localfile
tmpf <- tempfile(tmpdir='/tmp')
write.csv(cars, row.names=F, file=tmpf)

dfs2 <- hdfs.upload(tmpf, dfs.id="cars_wo_meta", header=TRUE,
                    attach=FALSE)

# Meta data exists
cars.dfs1 <- hdfs.attach(dfs1)
head(hdfs.get(cars.dfs1))

# Meta data missing so Sampling will be done
cars.dfs2 <- hdfs.attach(dfs2)
head(hdfs.get(cars.dfs2))

# Cleanup
hdfs.rm(cars.dfs1)
hdfs.rm(cars.dfs2)
```

`hdfs.cache` *Controls ORCH HDFS cache behavior.*

Description

Allows to fine-tune behavior of ORCH HDFS cache system. Normally the user does not need to use this function as system comes pre-configured with the most optimal options for most run environments.

Usage

```
hdfs.cache(onoff, disable = NULL, enable = NULL,
  ttl = NULL, ctl = NULL)
```

Arguments

`onoff`	Globally disable or enables HDFS caching. If not specified then will not change the current setting which allows to set fine-tuning options of the cache.
`disable`	Allows to disable caching of only one specific HDFS object. Can be an HDFS object identifier or HDFS-compliant path(s) as a character vector. This option must be set when an external change of the HDFS object by another user or 3rd party process is expected. Note that it **does not** recursively disables caching of child directories.

`enable`	Enables caching of previously disabled HDFS object. Can be an HDFS object identifier or HDFS-compliant path(s) as a character vector. To enable caching of all HDFS objects specify "*". Note that it **does not** recursively enables caching of child directories.
`ttl`	Sets time to live (TTL) configuration parameter of the cache in seconds. Each cached entry is allowed to live this amount of time after which it will be automatically deleted. -1 will revert the value to its default.
`ctl`	Sets clicks to live (CTL) configuration parameter of the cache in number of access attempts. Each cached entry is allowed to be accessed so many times and after exceeding this number will be automatically deleted. -1 will revert the value to its default.

Value

Always return the current state of the HDFS cache. If `onoff` argument was specified then it will return it invisibly.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.sync

`hdfs.cd` *Changes current HDFS working directory.*

Description

ORCH supports the notion of current working directory in HDFS. Every HDFS path when used with ORCH function is considered to be relative to the current working directory. Upon ORCH startup the current working directory is automatically set to the user's home in HDFS, which is "<root>/user/<user>". HDFS user name is the same as clients OS user name. HDFS root is normally "/" but can be changed via hdfs.setroot.

Usage

```
hdfs.cd(dfs.path)
```

Arguments

`dfs.path`	The new HDFS path is considered absolute if it starts with a "/" or relative to the user's home directory if it starts with "~". Otherwise the path is treated relative to the current path. See the function description for more details. Absolute path always uses current ORCH root as a reference point, see hdfs.root.

Details

ORCH current working directory is similar to a unix shell notation of working directory and accepts path with a number of special symbols. ORCH HDFS path compiler will walk through the user's path and denote each special character into a sub-path converting it into an absolute HDFS path.

Same as a unix shell ORCH allows three different types of HDFS paths:

- relative: If HDFS path starts with a resource name (file or directory) or from a "." then this path is treated as relative and will be concatenated to the current working directory to form the absolute HDFS path.
- absolute: If HDFS path starts with a divider ("/" symbol) then this path is treated as absolute and will be concatenated to the current HDFS root (see hdfs.root) to form the absolute HDFS path.
- home: If HDFS path starts with a home shortcut ("~" symbol) then this path is treated as relative to the user's home directory and will be concatenated to the HDFS user's home (<root>/user/<user>) to form the absolute HDFS path.

Same as a unix shell ORCH allows to use in an HDFS path special strings:

- / Parent and child directory and/or file divider. Directory and file names can not contain "/" symbol.
- . Identifies child directory, must be used at one token between parent and child dividers ("/" symbol). Directory and file names can contain "." symbol but in conjunction with other characters only. E.g. you can not name an HDFS file ".". Path "a/./b" is equivalent to "a/b".
- .. Identifies parent directory, must be used at one token between parent and child dividers ("/" symbol). Directory and file names can contain ".." symbol but in conjunction with other characters only. E.g. you can not name an HDFS file "..". Path "a/b/../c" is equivalent to "a/c".
- ~ Identifies user's home directory and can be used only as the very first symbol of an HDFS path. Directory and file names can include "~" symbol without any limitation, e.g. you can name an HDFS file as "~". Path "~/a" is equivalent to "/user/<user>/a".

Value

Current absolute HDFS path if the directory was set successfully. NULL will be returned if non-existing path is specified in `dfs.path`.

Note

Hadoop has no notion of "current working directory". This concept is entirely implemented and supported by ORCH only. ORCH closely follows Unix shell cd/pwd commands design to make navigation and access to HDFS resources easier for an R user.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.root hdfs.pwd hdfs.ls

`hdfs.cleanInput` *Clean ORCH HDFS objects*

Description

This function is used to clean ORCH HDFS objects by either removing bad/invalid values or replacing them with default values.

Usage

```
hdfs.cleanInput(input, config = NULL, tmpdir = "/tmp",
                            replace = TRUE, replace.val = NULL)
```

Arguments

`input`	ORCH HDFS identifier representing the input HDFS file to be cleaned
`config`	The mapred.config parameter used in hadoop.run. Default is `NULL`.
`tmpdir`	Character string specifying the HDFS directory path to store temporary results. These results are removed after the end of the function execution. Default is `"/tmp"`.
`replace`	Logical value to indicate if value replacement operation is to be performed. Default is `TRUE`. When `FALSE`, record removal is performed.
`replace.val`	When `replace = TRUE`, user can specify the default values in replace.val for replacement. This is a data.frame object with column names corresponding to the scalar data types supported in ORCH. See examples for usage. For default value of replace.val (`NULL`), replace.val uses: `data.frame("numeric"=0, "integer"=0, "logical" =FALSE, "character" = "", "factor" = as.factor(""))` When `replace = FALSE`, this argument is ignored.

Details

In ORCH, if for any data point in the input as.<columntype>(data) generates `NA`, it is considered to be dirty/invalid.

This function returns a cleaned ORCH HDFS object obtained by either replacing the invalid values (`replace = TRUE`) or removing corrupt records (`replace = FALSE`). After the end of the function execution, following statistics are displayed to show the impact of the cleaning operation:

1. Number of cells replaced when `replace = TRUE`
2. Number of rows removed when `replace = FALSE`
3. Precentage of cells replaced when `replace = TRUE`
4. Percentage of rows removed when `replace = FALSE`
5. Total number of input rows

Using cleaned input data before processing might result in significant performance improvements over data containing NA/missing values. It has been frequently observed that ORCH map-reduce jobs run at least 6-7 times faster on clean input data as compared to the unclean version. Note, all the performance improvements are based on the assumption that the execution time of the map-reduce job is not dominated by the user's map and reduce R scripts.

Value

ORCH HDFS identifier representing the cleaned ouput

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

See Also

`orch.fromHive` `orch.sample`

Examples

```
# create a data.frame with some invalid values
tmp1 <- data.frame(c1=c(1,2,3,4,5,6), c2=c(1,2,3,NA,NA,6))
# move the data.frame into HDFS
x11 <- hdfs.put(tmp1)
# clean the input by replacement of NAs with 0
y11 <- hdfs.cleanInput(x11)
# print the cleaned output
print(hdfs.get(y11))
# clean the input by removing the records with NA
y12 <- hdfs.cleanInput(x11, replace = FALSE)
# print the cleaned output
print(hdfs.get(y12))
# create a data.frame with some invalid values
tmp2 <- data.frame(c1=c(1,NA,NA,4,5,6),
                   c2=c("abc","def","efg",NA,NA,"xyz"), stringsAsFactors=FALSE)
# move the data.frame into HDFS
x21 <- hdfs.put(tmp2)
# clean the input by replacing numeric NAs with -1
# and character NAs with "abc"
y21 <- hdfs.cleanInput(x21, replace.val = data.frame(numeric=-1, character="abc",
                      stringsAsFactors=FALSE))
# print the cleaned output
print(hdfs.get(y21))
```

`hdfs.cp` *Copies HDFS directories and files.*

Description

Copies an existing HDFS file or directory located at `dfs.src` path relative to the current working directory to the specified by `dfs.dst` destination HDFS directory. If the destination directory already exists then the source object will be copied in there preserving its original name. If the destination directory does not exist then the source file or directory will be copied under the new directory name. This function is equivalent to "hadoop fs -cp" shell command.

Usage

```
hdfs.cp(dfs.src, dfs.dst, overwrite = FALSE,
  force = FALSE)
```

Arguments

`dfs.src`	HDFS source file or directory name in the current working directory, or its relative path, or an absolute HDFS-compliant path. See hdfs.cd for more details about HDFS path specification.
`dfs.dst`	HDFS destination file or directory name in the current working directory, or its relative path, or an absolute HDFS-compliant path.
`overwrite`	Enable replacing of HDFS directory and/or file if already exist. By default replacing is disabled.
`force`	Set this argument to TRUE to disable confirmation of copying a source with wildcard(*) in it, disable HDFS I/O check errors, and do not return result.

Value

TRUE if file or directory was copied successfully, FALSE if there was an error. In case of failure HDFS state may not be consistent, the destination data may be partially deleted (only if `overwrite` == TRUE) and only a portion of the source data may be copied. If `force` is set to TRUE then the function returns the result invisibly.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.mv hdfs.rmdir hdfs.mkdir

`hdfs.cwd` *Returns current working HDFS relative path.*

Description

ORCH support a notion of current working directory in HDFS. Every HDFS path when used with ORCH function is considered to be relative to the current working directory. Upon ORCH startup the current working directory is automatically set to the user's home in HDFS, which is "<root>/user/<user>". HDFS user name is the same as clients OS user name. HDFS root is normally "/" but can be changed via hdfs.setroot.

Usage

```
hdfs.cwd()
```

Value

Current working HDFS relative path or NULL if HDFS is not functional or not connected. The returned path will not include the current HDFS root (see hdfs.root) and will be reative to this root path.

Note

Hadoop has no notion of "current working directory". This concept is entirely implemented and supported by ORCH only. ORCH closely follows Unix shell cd/pwd commands design to make navigation and access to HDFS resources easier for an R user.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.pwd hdfs.cd hdfs.root hdfs.ls

`hdfs.delim` *Gets or sets default key and value fields separators.*

Description

Returns the currently configured or sets a new value of the system wide default key separator and values separator. The key separator is used in HDFS text based files to separate key field from value fields. The value separator is used in HDFS text based files to separate individual value fields from each other. Examples of input data that use the key and/or value separator are:

- key<key.sep>value1<value.sep>value2...- Data with a key and N values.
- <key.sep>value1<value.sep>value2...- Data with an empty key and N values.
- value1<value.sep>value2...- Data without a key and N values.
- key<key.sep>value- Data with a key and 1 value.
- value- Data without a key and 1 value.
- key- Data with a key and no values.

Usage

```
hdfs.delim(key.sep, value.sep)
```

Arguments

`key.sep`	Optionally a new key separator value to set. Must be one character. If not specified then the function will only return the current value set.
`value.sep`	Optionally a new value separator value to set. Must be one character. If not specified then the function will only return the current value set.

Details

Keep in mind that the key/value separators can be altered at the time of writing data to HDFS for each specific object. The key/value separators are stored in HDFS object's metadata and default system-wide settings are not used at the time of reading this object back from HDFS in ORCH. These default settings are used only when user does not specify the key/value separators explicitly in the function call for the following operations:

- Writing a new dataset to HDFS.
- Attaching existing HDFS data which does not have any metadata.
- Attaching existing HDFS data with metadata missing key/value separators.

Value

Currently configured system-wide key and value separators are a vector of two character values. Upon ORCH startup the key separator is set to a tabulation character "\t" and the values separator is set to a comma character ",".

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

hdfs.keysep hdfs.valuesep

`hdfs.describe` *Describes known characteristics of an HDFS object.*

Description

Returns a data.frame with extensive description of an HDFS object's attributes. If the object does not exist or has no metadata attached (i.e., hdfs.attach was not executed on the directory) then NULL will be returned. The resulting data frame will have two columns: NAME - name of the characteristic, and VALUE - its value

Usage

```
hdfs.describe(dfs.id)
```

Arguments

`dfs.id`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively user can pass a string with HDFS compliant directory path relative to the current working directory.

Details

Reported ORCH metadata characteristics are:

- path: Absolute HDFS path to the described object.
- origin: description of the HDFS object origin.
- class: R class corresponding to HDFS data, e.g. data.frame.
- types: list of data type names for each column.
- names: vector of known column names.
- dim: number of rows (or -1 if unknown) and columns.
- categorized: TRUE if "factor" columns are stored as indexes.
- has.key: TRUE if the data has key column.
- key.column: index and name of a column containing keys.
- empty.key: TRUE if the data has "" key.
- has.rownames: TRUE if rownames are stored with data.
- key.sep: delimiter used as a separator between key and values.
- value.sep: delimiter used as a separator between values.

- quoted: quoting symbol used when parsing fields or FALSE.
- pristine: TRUE if data has no invalid fields.
- trimmed: TRUE if number of columns in data can be less than "dim".

"Pristine" attribute defines the data as:

- a) Every row has the same number of columns.
- b) All missing values are represented either as "NA" or "".
- c) There are no non numeric values in numeric columns.

"Trimmed" attribute defines the data as:

- a) Number of "physical" columns stored in HDFS files is larger than the logical one stored in metadata.
- b) Columns are "hidden" in the logical view from user's perspective. ORCH will ignore "hidden" columns.
- c) "Hidden" columns contain no data i.e., then are blank strings("") in the HDFS files.

Value

A data frame containing the description, or NULL if the HDFS object does not exist or does not have any ORCH metadata associated with.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.meta hdfs.attach hdfs.levels

`hdfs.dim` *Returns number of rows and columns of an HDFS object.*

Description

Equivalent to R's dim() function for HDFS objects. Dimensions are typically stored in ORCH metadata alongside an HDFS object, which enables the function to return known values directly. If the dimensions are unknown, then this function tries to identify them. It downloads the data set to the client's R memory **if** the file is small enough or executes a mapReduce job for large data sets. After the function counts the number of rows and columns, it updates the ORCH metadata for this HDFS object to preserve the discovered values. Then it does not need to repeat the same counting process the next time the function is invoked.

Usage

```
hdfs.dim(dfs.id, force = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier to inspect. This is a special ORCH object returned by hdfs.attach and other functions that access HDFS, which represents an HDFS directory. Alternatively, it can be a string with an HDFS-compliant directory path relative to the current working directory.
`force`	Controls whether a mapReduce job (to determine dimensions) runs without confirmation. This parameter must be set to TRUE if your R script invokes `hadoop.run` and is run in batch mode, with unattended execution. Otherwise, the progress will be halted for user confirmation. `force` implicitly enables silent execution.

Value

Vector c(rows, columns). If any of the values is unknown for any reason (job failure, unrecognized format, etc.), then it will have value NA.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.nrow hdfs.ncol hdfs.meta

`hdfs.download` *Downloads an HDFS file or directory to the local file system.*

Description

This is the simplest and fastest possible way to transfer data from HDFS to a local storage. This function copies HDFS directory's `dfs.id` part-files into one local file specified by `filename` combining all data files into one. All non-data containing files like ORCH metadata, Hadoop's system files "_SUCCESS", ".checksum" and other known files that do not contain any data will be ignored unless `all` argument is set to TRUE.

Usage

```
hdfs.download(dfs.id, filename = NULL, dfs.file = NULL,
  all = FALSE, overwrite = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory.
`filename`	Optional local file path and name which will receive content of the `dfs.id` HDFS directory.
`dfs.file`	HDFS file name(s) to download. If not specified or NULL then all data files from `dfs.id` directory will be downloaded as a bulk. User can specify one of several files to download as a character vector.
`all`	Download all files including system (e.g. starting with "_", "."). Be aware that this may corrupt data structure by embedding ORCH metadata and Hadoop's system data into data files.
`overwrite`	If TRUE the replaces `filename` local file if already exists. Otherwise an error will occur if the file already present.

Value

local file name of the downloaded data if operation has finished successfully. Otherwise NULL if error occurred.

Attention

Use this function with caution since the entire contents of an HDFS directory are brought into your local file system. Considering ability of HDFS to store vast amounts of data you may exhaust your hard drive's free space.

Warning

Specifying download files list in `dfs.file` argument may significantly downgrade the function performance as each file will be downloaded separately and not as a bulk directory download.

Note

Data files do not need to be named according to Hadoop's mapReduce convention "part-(m-)?(r-)?[0-9]5" to be picked up by the download function. Every file in the directory with a name starting not with "_" or "." is considered a data file and will be picked up (unless `all` argument is set to TRUE). The downloaded data is formatted as-is in HDFS.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.upload hdfs.put hdfs.get

`hdfs.exists` *Checks if an HDFS object exists.*

Description

Verifies validity of the HDFS object identifier `dfs.id` or existence of an HDFS directory path that is specified as a string in `dfs.id` argument. If HDFS object exists then it can be safely used with any of ORCH public API functions which access HDFS data like hadoop.run, hdfs.get, etc.

Usage

```
hdfs.exists(dfs.id)
```

Arguments

`dfs.id`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory.

Details

HDFS data may be referenced at the same time by several HDFS object identifiers in ORCH. If one of the objects gets deleted with hdfs.rm, or it's directory gets removed with hdfs.rmdir, or the referred HDFS resource gets (re)moved outside of ORCH by a 3rd party process then the HDFS object identifiers may become invalid and would refer to non-existing HDFS data. This is one example of a situation where you'd need to check the validity of the HDFS object using hdfs.exists.

Value

TRUE if data exists and valid or FALSE if data does not exists or in case of a failure.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.attach hdfs.rm hdfs.rmdir hdfs.ls

`hdfs.fromHive` *Converts a HIVE table to a dfs identifier in ORCH.*

Description

This function is used to convert an ORE-HIVE table represented by an `ore.frame` object to an HDFS object compatible with ORCH APIs. It converts an `ore.frame` that points to a HIVE table into a HDFS identifier used by ORCH. The function will convert the HIVE table metadata into ORCH metadata and if needed may materialize HIVE query into a physical HDFS data set.

Usage

```
hdfs.fromHive(table, out.table = NULL, overwrite = FALSE)
```

Arguments

`table`	ore.frame object or a character string representing a HIVE table.
`out.table`	Optional table name for the staging table. See the function description for more details. If this argument is not specified then a temporary HIVE table will be created to hold the staged data.
`overwrite`	Overwrite the ORCH metadata. If ORCH metadata file already exists in the HDFS directory pointed by `table`, it is not overwritten when `overwrite` = FALSE.

Details

Currently, only non-partitioned HIVE tables are supported for conversion. Partitioned tables are stored as a collection of sub-directories which does not correspond to ORCH data storage model. Therefore, using a partitioned HIVE table as input would result in an error.

Value

Returns the HDFS object representing the input ORE-HIVE table. This HDFS object is consumable by ORCH.

Attention

A HIVE staging table is created if `table` object does not represent a physical HIVE table (e.g. transformed ore.frames, views, etc.). User can optionally pass in a name using `outtabname` for the staging table (if created) to be used as an ORE-HIVE table for further processing.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.toHive

Examples

```
# Put the cars dataset into HDFS.
ore.create(cars, table="cars1")

# Create the dfs.id object from the HIVE table.
z <- hdfs.fromHive(cars1)

# hdfs.* functions can be used on this object
print(hdfs.get(z))

# Remove created Hive tables.
ore.drop(table="cars1")
hdfs.exists(z)
```

`hdfs.fromRData` *Converts an HDFS binary object into plain HDFS text object.*

Description

This function executes a mapReduce job that consumes an HDFS directory containing the special ORCH binary format and which was already attached to ORCH (see hdfs.attach) and outputs the same data but in a plain text file format.

Usage

```
hdfs.fromRData(dfs.id, out.name = NULL,
  overwrite = FALSE, parts = NULL, key.sep = NULL,
  value.sep = NULL, silent = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier of the input data to be converted. This is a special ORCH object returned by hdfs.attach and other functions which represents a directory in HDFS. Or it can be a string with HDFS-compliant path relative to the current working directory.
`out.name`	Output HDFS directory name or an HDFS object identifier of the output converted plain text data. Note that the output directory must not exist otherwise the function will fail. See `overwrite` for more details. If the output directory is not specified a temporary one will be created in HDFS "/tmp".
`overwrite`	Allows overwriting of the output HDFS directory if already exists with the same name. By default overwrite is disabled for safety of data manipulations.
`parts`	Number of desired output "part" files. This option directly controls the size of each "part" file which will approximately equal to the total output size / number of "part" files. The function will try to satisfy the specified requirement but does not guarantee it due to the Hadoop jobs execution restrictions and input file

	format limitations. If not specified it will rely on Hadoop's default behavior and will generate a part file per each input part file.
`key.sep`	Key field separator character. If not specified then the original separator (the one used in text data prior to binary conversion) stored in the input HDFS object metadata will be used. If not available then the default ORCH global key separator will be used ("\t" by default).
`value.sep`	Value fields separator character. If not specified then the original separator (the one used in text data prior to binary conversion) stored in the input HDFS object metadata will be used. If not available then the default ORCH global value separator will be used ("," by default).
`silent`	Do not print information messages to console. Do not print final attach summary at the end of the run.

Details

The binary format is readable by ORCH Hadoop jobs only and gives the advantage of fastest achievable data read and write throughput in ORCH R mapReduce jobs. Data can be loaded directly into R memory in the mapper or the reducer without any parsing or conversion of text into R objects.

Value

HDFS object identifier if data was successfully converted to the plain text format, otherwise NULL if any conversion error has occurred.

Note

Output of the function is always pristine by definition because the ORCH binary format can contain only pristine data.

See Also

hdfs.toRData

`hdfs.get` *Copies data from HDFS into R in-memory object.*

Description

Copies data from HDFS into R in-memory object. Reads ORCH metadata with all meta files like levels data and restores all attributes including column names, data types, row names, factor levels, etc. If the data originated from R environment, i.e., data was put in HDFS using hdfs.put, these attributes are available. Otherwise, if data has originated from another source and was automatically attached via hdfs.attach generic reverse-engineered object attributes like "val1", "val2" for columns names and default data type "data.frame" will be assigned. Users can also update the metadata using hdfs.meta to avoid generic column names.

Usage

```
hdfs.get(dfs.id)
```

Arguments

`dfs.id`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory.

Value

A data.frame object in memory in the local R environment containing the imported data set, or NULL if the operation has failed for some reason.

Note

If the HDFS file contents can comfortably fit into an in-memory R data frame object then simply use hdfs.get(). Otherwise you must resort to first fetching the HDFS files into local file system and then reading chunks of the file into memory as desired. See hdfs.download for more details.

Key and value separators specification is not required when calling this function because it is stored together with the data itself and will be retrieved automatically from its ORCH metadata.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.put hdfs.download hdfs.upload hdfs.meta hdfs.describe

Examples

```
x <- hdfs.put(cars)
y <- hdfs.get(x)
all(y == cars)
all(names(y) == names(cars))
```

`hdfs.head` *Reads unformatted head of an HDFS object.*

Description

Returns the first `n` rows of the specified HDFS file without any parsing. If HDFS object contains many part-files then the function will retrieve the head portion or the whole file from the first (lexicographically sorted by name) part-file, if number of lines retrieved is less than `n` then head portion of the next file (and so on) will be appended.

Usage

```
hdfs.head(dfs.id, n = 0L)
```

Arguments

`dfs.id`	HDFS object identifier to get heading data for. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS-compliant directory path relative to the current working directory.
`n`	Number of rows to return. Must be >= 0. If 0 is specified (default value) then the function will return a default head portion of one first part-file which will give the fastest possible execution time. The default size is the whole part file if it's small enough (<=100KB) or an arbitrary head portion if it's too large (>100KB).

Details

The function performance will degrade based on two factors - number of part files in the input HDFS directory (e.g. HDFS object) and size of each part file. Performance approximately linearly degrades with increase of number of HDFS data files and with size increase of each data file. Although after reaching approximately 100KB, file size increase will not significantly affect the runtime anymore.

Value

Character vector of the specified length `n`. The length can be less than `n` if the specified number of lines can not be retrieved for some reason. NULL is returned if the object does not exist or an error has occurred. If the HDFS directory has no non-empty data files then a 0-size character vector will be returned.

Note

The function is designed to behave as close as possible to Unix shell "head" utility but inherits limitation of Hadoop's HDFS API. There is no equivalent command in Hadoop command line interface.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.tail hdfs.sample hdfs.get hdfs.download

hdfs.id *Creates a new ORCH HDFS object identifier.*

Description

Converts an HDFS string path to R "dfs.id" objects. If `dfs.x` is malformed and contains an invalid HDFS path, or the specified HDFS path does not exist (except if `force` is TRUE) then returns NULL.

Usage

```
hdfs.id(dfs.x, absolute = FALSE, force = FALSE)
```

Arguments

`dfs.x`	HDFS relative or absolute path as a string. If HDFS object identifier is given then will just check its existence (if `force` is FALSE).
`absolute`	TRUE if `dfs.x` is an absolute HDF path and has to be preserved as-is. Use FALSE (default mode) to treat `dfs.x` as a reative path and append to the current working directory (see hdfs.cd and hdfs.pwd for more details).
`force`	Do not perform existence check when TRUE. Default is FALSE.

Details

This function is equivalent to hdfs.attach but will never do any metadata discovery and generation if HDFS directory has never been attached before. It also allows you to create identifier of non-existing HDFS object.

Value

ORCH HDFS object identifier which points to an HDFS object if everything is fine, otherwise will return NULL if `dfs.x` does not contain a valid HDFS path or the path does not exist (except when `force` == TRUE).

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.attach hdfs.exists

Examples

```
## Not run:
    hdfs.id("/tmp/bad_path") # returns NULL and error
    hdfs.id("/tmp/bad_path", force=T) # returns HDFS object

## End(Not run)
```

`hdfs.keysep` *Gets or sets default key field separator.*

Description

This function can be used to either return the currently configured system wide default key separator **or** can be used to set the system wide default key separator to a new value. The key separator is used in HDFS text based files to separate key field from value fields. Examples of input data that use the key separator are:

- key<key.sep>value1,value2...- With key data type.
- <key.sep>value1,value2...- Empty key data types.
- value1,value2...- Key-less data type.
- key- Key-only, no separator used.

Usage

```
hdfs.keysep(key.sep)
```

Arguments

`key.sep`	Optionally a new key separator value to set. Must be one character. If not specified then the function will only return the current value set.

Details

Keep in mind that the key separator can be specified explicitly at the time of writing data to HDFS for each specific object. The key separator is stored in HDFS object's metadata and default system-wide value is not used at the time of reading this object back from HDFS in ORCH. This system-wide default value is used only when user does not specify the key separator explicitly in the function call for the following operations:

- Writing a new dataset to HDFS.
- Attaching existing HDFS data which does not have any metadata.
- Attaching existing HDFS data with metadata missing key separator.

Value

Currently configured system-wide key separator. Upon ORCH startup it is set to a tabulation character "\t".

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.valuesep hdfs.delim

`hdfs.levels` *Reads or writes ORCH levels metadata for an HDFS object.*

Description

ORCH levels metadata contains definition of the distinct levels of each categorical/factor column in the HDFS object. The levels metadata is stored in a separate metadata file aside from the main metadata stored in __ORCHMETA__ file to lower its size and prevent potential bloating.

Usage

```
hdfs.levels(dfs.id, ..., overwrite = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions manipulating HDFS which represents a directory in HDFS. Alternatively user can pass a string with HDFS compliant directory path relative to the current working directory.
`...`	List of attributes and values to read or to write: • none: Get list of all levels available. • column_name=value[, ...]: Write levels for one or several columns. • "column_name"[,...]: Read levels for one or several columns.
`overwrite`	If a column already has levels written as a sidecar file in HDFS attempt to write it again will fail. Setting this parameter to TRUE will allow overwriting existing levels.

Details

This function allows user to read or write ORCH levels metadata for an HDFS object from a client R program or from within a running mapReduce R job. Each column of the HDFS object can have "levels" data attached to it. The levels identify unique values that are and can only be used within this column. At the same time column can contain factor indexes or original values. This allows uniform factorization of the column data in distributed mapReduce jobs that receive only part of an original dataset.

Value

List of levels if column levels to write are not specified in `...`, or only column names to read without values are specified in `...`. If only one column name to read is specified in `...` then returns only its value without wrapping it into a list. Otherwise return TRUE if all levels were written successfully, or FALSE if any level write has failed for some reason. See examples.

Note

If column(s) levels are specified as "name=levels" in [...] parameter then writes given levels into HDFS object alongside the main data as a sidecar file. If no column levels to write are given in [...] or only column names without actual level values are specified in [...] then reads them from HDFS and returns a list of attached levels.

`...` parameter can be specified using a vector or CSV string. In all cases it will mean to get one or several column levels. All styles can be mixed and interchanged as needed:

- c("column_name"[,"column_name"[,...]])
- "column_name[,column_name[,...]]"

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.meta hdfs.describe hdfs.attach

Examples

```
## Not run:
    hdfs.levels(x) # return list of all levels.
    hdfs.levels(x, "speed") # return levels of "speed" column.
    hdfs.levels(x, "speed", "dist") # return levels for two columns.
    hdfs.levels(x, speed=c(1,2,3)) # writes levels for one column.
    hdfs.levels(x, speed=c(1,2,3), dist=c(4,5,6)) # writes levels for two columns.

## End(Not run)
```

`hdfs.ls` *Lists files and directories.*

Description

Returns a vector with names of all HDFS directories and files located at the current working directory. The user can specify the HDFS path to a directory to list if needed. The function will list data and system files without any differentiation. This function is equivalent to "hadoop fs -ls" shell command.

Usage

```
hdfs.ls(dfs.path = ".", pattern = NULL)
```

Arguments

`dfs.path`	Optional relative to the current working path or absolute HDFS-compliant path. If not specified then list of all objects at the current working path will be returned. See hdfs.cd for more details about HDFS path specification.
`pattern`	Optional regular expression for filtering of returned file names. For example pattern="^[^_]" will filter out all "_*" files.

Value

R character vector of all (or filtered by `pattern`) HDFS file and directory names located at the current working directory or at the HDFS path specified by `dfs.path` argument. NULL will be returned in case of an invalid HDFS path or any other error.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.cd hdfs.pwd hdfs.root

Examples

```
cat("Running hdfs.ls() example.\n")

# Copy "cars" dataset into HDFS directory.
dfsCars <- hdfs.put(cars, dfs.name="cars_example")

# List all objects in the current working directory.
hdfs.ls()
# List all files in the HDFS directory with "cars" data.
hdfs.ls(dfsCars)
# List only data files in the HDFS directory with "cars" data.
hdfs.ls(dfsCars, pattern="[^._].*")

# Remove "cars" dataset from HDFS.
hdfs.rm(dfsCars, force=T, notrash=T)
```

`hdfs.meta` *Retrieves or updates ORCH metadata for an HDFS object.*

Description

Retrieves or updates ORCH metadata for an HDFS object. ORCH metadata describes the content of HDFS data files and allows ORCH to correctly read and parse HDFS raw part-files into R structured objects like data.frame or matrix.

Usage

```
hdfs.meta(dfs.id, ..., force = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively user can pass a string with HDFS compliant directory path relative to the current working directory.
`...`	List of attributes and values to updated or to retrieve: • none: Get list of all attributes • attr_name=value[, ...]: set one or several attributes • "attr_name"[,...]: get one or several attributes
`force`	Don't check for invalid or unknown attributes. This allows to set or retrieve custom attributes that do belong to ORCH.

Details

ORCH metadata keeps the following attributes:

- kvs: Reserved for ORCH.
- types: Vector of type names for each column.
- names: Vector of column names.
- class: R class corresponding to HDFS data.
- keyi: Index of a column containing keys.
- rownamei: Index of a column containing row names.
- key.sep: Symbol used as a separator between key and values.
- value.sep: Symbol used as a separator between values.
- origin: Description of HDFS object origin.
- dim: Number of rows (or -1 if unknown) and columns.
- pristine: TRUE if data is known to be valid and not have NA values.
- quote: Quoting symbol used for parsing data.
- categorized: TRUE if "factor" columns are stored as indexes.
- trim: TRUE if number of columns in data is less than "dim".
- rdata: TRUE the HDFS is stored as binary RData.
- split: number of records in one binary chunk.
- na.strings: strings that should be treated as NA values.

Value

List of attributes if user attributes to set are not specified in `...`, or only names of attributes to retrieve without values are specified in `...`. If only one attribute to retrieve is specified in `...` then returns only its unlisted value. Otherwise return TRUE if all attributes were set, or FALSE if any attribute was not set for some reason. See examples.

Note

If no attributes to update are given in `...` then returns a list of stored meta attributes. If any attributes are specified as "name=value" in `...` parameter then updates given attributes in HDFS object metadata.

`...` parameter can be specified using a vector or CSV string. In all cases it will mean to get one or several attributes. All styles can be mixed and interchanged as needed:

- c("attr_name"[,"attr_name"[,...]])
- "attr_name[,attr_name[,...]]"

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.describe hdfs.attach hdfs.levels

Examples

```
## Not run:
    # Examples of hdfs.meta invokations.
    hdfs.meta(x) # return list of attributes.
    hdfs.meta(x, "keyi") # return one attribute value.
    hdfs.meta(x, "key.sep", "value.sep") # return 2 attribute values.
    hdfs.meta(x, pristine=TRUE) # sets "orch.pristine" to TRUE in HDFS.
    hdfs.meta(x, bad_attr=TRUE) # error, unknown attribute.
    hdfs.meta(x, custorm_attr=TRUE, force=TRUE) # ok, attribute allowed.

## End(Not run)
```

`hdfs.mkdir` *Creates a new HDFS directory.*

Description

Creates a new HDFS sub-directory in the current working directory or if `dfs.name` includes path then relative to the current working directory. Newly created directory will be empty. This function is equivalent to "hadoop fs -mkdir" shell command.

Usage

```
hdfs.mkdir(dfs.name, overwrite = FALSE, cd = FALSE)
```

Arguments

`dfs.name`	Name of the new directory to create. The name can include an HDFS path relative to the current working HDFS directory.
`overwrite`	If TRUE then will delete all the data in existing directory with the same name. Default value is FALSE.
`cd`	If TRUE then automatically sets the newly created directory as the current working directory. Default value is FALSE. See hdfs.cd for more information.

Value

New HDFS directory absolute path as string or NULL if the new directory was not created for some reason.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.rmdir hdfs.cd

`hdfs.mv` *Moves HDFS directories and files.*

Description

Moves an existing HDFS file or directory located at `dfs.src` path relative to the current working directory to the specified by `dfs.dst` destination HDFS directory. If the destination directory already exists then the source object will be moved there preserving its original name. If the destination directory does not exist then the source file or directory will be renamed and optionally moved there. This function is equivalent to "hadoop fs -mv" shell command.

Usage

```
hdfs.mv(dfs.src, dfs.dst, overwrite = FALSE,
  force = FALSE)
```

Arguments

`dfs.src`	HDFS source file or directory name in the current working directory, or its relative path, or an absolute HDFS-compliant path. See hdfs.cd for more details about HDFS path specification.
`dfs.dst`	HDFS destination file or directory name in the current working directory, or its relative path, or an absolute HDFS-compliant path.
`overwrite`	Enable replacing of HDFS directory and/or file if already exist. By default overwriting is disabled.
`force`	Set this argument to TRUE to disable confirmation of '*' moving, don't HDFS I/O check errors, and do not return result.

Value

TRUE if file was moved successfully, FALSE if there was an error. In case of failure HDFS state may not be consistent, the destination data may be partially deleted and only a portion of the source data may be moved. If `force` is set to TRUE then the function returns the result invisibly.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.cp hdfs.rmdir hdfs.mkdir

`hdfs.ncol` *Returns number of columns of an HDFS object.*

Description

See hdfs.dim for detailed description of its functionality and parameters. This function is a shortcut for hdfs.dim()[2].

Usage

```
hdfs.ncol(dfs.id, force = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier to inspect. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory.
`force`	Do not ask confirmation for running a mapReduce job. This parameter must be set to TRUE if a script is intended to be run in a batch mode, e.g. unattended execution. `force` implicitly enables silent execution.

Value

Number of columns as an integer value. If the value is unknown and can not be computed for any reason then it will return NA.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.dim hdfs.nrow hdfs.meta

`hdfs.nrow` *Returns number of rows of an HDFS object.*

Description

See hdfs.dim for detailed description of its functionality and parameters. This function is a shortcut for hdfs.dim()[1].

Usage

```
hdfs.nrow(dfs.id, force = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier to inspect. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory.
`force`	Do not ask confirmation for running a mapReduce job. This parameter must be set to TRUE if a script is intended to be run in a batch mode, e.g. unattended execution. `force` implicitly enables silent execution.

Value

Number of rows as an integer value. If the value is unknown and can not be computed for any reason then it will return NA.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.dim hdfs.ncol hdfs.meta

`hdfs.parts` *Counts the number of data files in HDFS object.*

Description

Lists and returns number of data files the HDFS object `dfs.id` is divided into. Normally data files are named as "part-12345" but any file name can be used. Files with names starting with "_" or "." are excluded unless `all` argument is TRUE as they normally hold system information and ORCH metadata.

Usage

```
hdfs.parts(dfs.id, all = FALSE, nonzero = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier to inspect. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory.
`all`	Count all files of the specified HDFS object including system and ORCH metadata files. Default is FALSE.
`nonzero`	Count in only non-empty data files. Default is FALSE.

Details

If HDFS object has no data files then 0 will be returned indicating that the object exists in HDFS file system but its directory is empty. Non-existing HDFS objects will result in returning NULL to indicate that the object is invalid. Note that the object may contain a number of empty data files and while it has no data (e.g., it's empty) number of data files returned will still be > 0.

Value

Number of data files the HDFS object is divided into (normally they are named as "part-12345"). If HDFS object has no data files then 0 will be returned. If HDFS object does not exist then the function returns NULL.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

hdfs.size hdfs.ls

`hdfs.pull` *Copies data from HDFS to RDBMS.*

Description

Input object is a HDFS object identifier and the function returns the name of a new table containing loaded data from HDFS. Name of the table is the same as the name of the HDFS object's directory unless redefined by `db.name` argument. Pulling of data is done by one of the underlying drivers (see details about `driver` argument) and will start a number of mapReduce jobs that will read data from HDFS in parallel and push data to the database table.

Usage

```
hdfs.pull(dfs.id, db.name = NULL, overwrite = FALSE,
  sep = .orch.env$val.sep, driver = NULL)
```

Arguments

`dfs.id`	HDFS object identifier. You can also specify an absolute or relative (to the current work directory in HDFS) HDFS path to a directory to be exported to the database.
`db.name`	Optional database table name. If not specified then HDFS object name (e.g. its HDFS directory name) will be used as a target database table name
`overwrite`	If TRUE will remove an existing database table, otherwise export will fail with an error if the table already exists. Default is FALSE.
`sep`	Optional HDFS value fields separator. This argument will be used only when exporting an HDFS directory that was never attached and does not have attached ORCH metadata.
`driver`	Choose RDBMS to HDFS data transfer driver, the default one is selected when RDBMS connection is established via orch.connect. This allows to use a different driver for the data transfer. Available drivers are: "sqoop", "olh".

Details

If orch.connect was invoked in secure mode, then, this API will prompt the user to enter database password. The password is held encrypted in memory and transferred to an on-disk configuration file for use by Sqoop or other data transfer driver. This is the way Sqoop/OLH is invoked in general in the batch mode as well. If orch.connect was invoked in non-secure mode (i.e. `secure` = FALSE), then the password entered earlier would have been kept encrypted in memory and transferred to the Sqoop/OLH configuration file on disk. The configuration file gets destroyed automatically once Sqoop/OLH has read it. This is made possible by the fact that we create the configuration file as a temporary-unlinked file on Linux.

Value

Exported database table name that can be used in ore.sync to attach the table to Oracle R Enterprise framework. Otherwise will return NULL in case of any error.

Attention

Due to Sqoop/OLH limitations HDFS files without a key or with the key delimiter equal to the value delimiter can be imported from HDFS to RDBMS. Otherwise hdfs.pull will error out preventing any attempts on import.

Attention

There have been several bugs identified in Sqoop 1.4.1 that can cause this interface to fail as it relies on Sqoop functionality internally. Bugs have been filed against Sqoop. Depending on the version of the Sqoop installed in your environment the function may fail.

Note

Data transfer is executed synchronously and large datasets can appear to "hang" your R console for a while. A number of information messages will be reported to the user while the import procedure is running.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.connect hdfs.push hdfs.put hdfs.get

`hdfs.push` *Copies data from RDBMS to HDFS.*

Description

The input object `x` can be of "ore.frame" type, or the Database table name (optionally including schema), or a full SQL query and returns an HDFS object identifier which can be used in further HDFS/Hadoop function calls. Pushing of data is done by one of the underlying drivers (see details about `driver` argument) and will start a number of mapReduce jobs that will pull data out of the database in parallel and store it into a set of files in HDFS directory identified by `dfs.name`.

Usage

```
hdfs.push(x, key = NULL, dfs.name = NULL, sep = ",",
  overwrite = FALSE, split.by = NULL, driver = NULL)
```

Arguments

`x`	An object of type "ore.frame" representing a table or a SQL query and managed by Oracle R Enterprise (for more information see ore.frame), or a character object of length 1 that contains a table name (optionally with schema name) or a full SQL statement.
`key`	Optionally specifies the key column. It may be specified as column name or as column numeric index. If `key` == 0 then empty-key HDFS data will be generated meaning that the key column will contain "" strings.
`dfs.name`	Optionally a custom name to assign the imported HDFS object. If not specified then a temporary HDFS object will be generated which will be deleted at the end of R session.
`overwrite`	Allows overwriting of the target HDFS object with the same name. Only applies when `dfs.name` is specified.
`split.by`	Optionally specifies the column to use for data partitioning. This can greatly improve performance of data import from RDBMS if partitions are uniformly distributed.
`driver`	Choose RDBMS to HDFS data transfer driver, the default one is selected when RDBMS connection is established via orch.connect. This allows to use a different driver for the data transfer. Available drivers are: "sqoop", "olh".

Details

If orch.connect() was invoked in secure mode, then, this API will prompt the user to enter database password. The password is held encrypted in memory and transferred to an on-disk configuration file for use by Sqoop or other transport driver. This is the way Sqoop is invoked in general in a batch mode as well. If orch.connect() was invoked in secure=F mode, then the password entered earlier would have been kept encrypted in memory and transferred to Sqoop configuration file on disk. The configuration file gets destroyed automatically once Sqoop has read it. This is made possible by the fact that we create the configuration file as a temporary-unlinked file on Linux.

Value

HDFS object identifier if data was successfully exported or NULL if transfer error has occurred.

Attention

There have been several bugs identified in Sqoop 1.4.1 that can cause this interface to fail as it relies on Sqoop functionality internally. Bugs have been filed against Sqoop. Depending on the version of the Sqoop installed in your environment the function may fail.

Note

Data transfer is executed synchronously and large datasets can "hang" R environment for a while. A number of information messages will be reported to the user while the import procedure is running.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.connect hdfs.pull hdfs.put hdfs.get

`hdfs.put` *Copies data from R in-memory object into HDFS.*

Description

Copies data from R in-memory object (data.frame, matrix, vector or list) into HDFS file system. All data attributes like column names, data types, etc. get stored as ORCH metadata along side the data itself in HDFS.

Usage

```
hdfs.put(data, key = NULL, dfs.name = NULL,
  overwrite = FALSE, rownames = FALSE,
  categorize = FALSE, key.sep = .orch.env$key.sep,
  value.sep = .orch.env$val.sep,
  digits = .orch.env$digits,
  scientific = .orch.env$scientific)
```

Arguments

`x`	A data.frame (or other supported data type) to export into HDFS.
`key`	Name or index of the column which represent key value. NULL value (or -1) indicates key-less data (e.g. rows will contain only values "val1,val2"), "" value (or 0) indicated empty-key data (e.g. rows will contain values and empty key "\tval1,val2").
`dfs.name`	Custom name to assign the HDFS object (optional). If not specified then a unique temporary name will be generated for HDFS object. Temporary HDFS files are removed at the end of R session.
`overwrite`	Allows overwriting of HDFS objects with the same name. By default overwrite is disabled for safety of data.
`rownames`	Enables storing of row names as a data column in HDFS alongside with the data itself if TRUE. Row names are stored as a special last data column and transparently restored by ORCH framework when reading data back from HDFS.
`categorize`	Store "factor" columns as indexes. This also triggers a mechanism of storing "levels" as a meta sidecar file alongside with the data itself. For more details see hdfs.levels.
`key.sep`	Key field separator character, "\t" default. For uniform separators set it to the same value as `value.sep`. Key separator is stored in ORCH metadata.

`value.sep`	Value fields separator character, "," default. For uniform separators set it to the same value as `key.sep`. Key separator is stored in ORCH metadata.
`digits`	How many significant digits are to be used for numeric and complex data. The default, uses orch.options("digits"). Enough decimal places will be used so that the smallest (in magnitude) number has this many significant digits.
`scientific`	Logical specifying whether elements of a real or complex vector should be encoded in scientific format. By default uses orch.options("scientific").

Value

HDFS object identifier if data was successfully transferred into HDFS file system, otherwise NULL if any transfer error occurred.

Note

You can use hdfs.put instead of hdfs.push to copy data from ore.frame objects, such as database tables, to HDFS. The table must be small enough to fit in R memory; otherwise, the function fails. The hdfs.put function first reads all table data into local R memory and then transfers it to HDFS. For a small table, this function can be faster than hdfs.push because it does not use Sqoop and thus does not have the overhead incurred by hdfs.push.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.get hdfs.download hdfs.upload hdfs.meta hdfs.levels hdfs.describe

Examples

```
x <- hdfs.put(cars)
y <- hdfs.get(x)
all(y == cars)
all(names(y) == names(cars))
```

`hdfs.pwd` *Returns present working HDFS absolute path.*

Description

ORCH supports a notion of current working directory in HDFS. Every HDFS path when used with an ORCH function is considered to be relative to the current working directory. Upon ORCH startup the current working directory is automatically set to the user's home in HDFS, which is "<root>/user/<user>". HDFS user name is the same as clients OS user name. HDFS root is normally "/" but can be changed via hdfs.setroot.

Usage

```
hdfs.pwd()
```

Value

Present working HDFS absolute path including the HDFS root (see hdfs.root) or NULL if HDFS is not functional or not connected.

Note

Hadoop has no notion of "current working directory". This concept is entirely implemented and supported by ORCH only. ORCH closely follows Unix shell cd and pwd commands design to make navigation and access to HDFS resources easier for an R user.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.cwd hdfs.cd hdfs.root hdfs.ls

`hdfs.rmdir` *Removes an HDFS directory.*

Description

Deletes an existing directory and all its files and sub-directories in HDFS relative to the current working directory. All data and metadata objects stored in or associated with this directory will be deleted. As a result, all associated HDFS object identifiers will also be invalidated. Any ORCH operations using these invalid identifiers will result in failure.

Usage

```
hdfs.rmdir(dfs.name, force = FALSE, notrash = FALSE)
```

Arguments

`dfs.name`	HDFS-compliant directory path relative to the current working directory. Alternatively it can be HDFS object identifier to be deleted.
`force`	Set this argument to TRUE to disable confirmation of '*' deletion, don't HDFS I/O check errors, and do not return result.
`notrash`	HDFS has a feature to move deleted data to a trash bin. In order to disable this feature and permanently delete an HDFS object set this argument to TRUE. It is required to set it to TRUE if an object gets deleted from a mapReduce object due to Hadoop job restrictions.

Value

TRUE if data was successfully deleted or FALSE if any error was detected. In case of failure HDFS state may not be consistent, the data may not be deleted, or only a portion of the data may be deleted. If `force` is set to TRUE then the function will return the result invisibly.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.rm hdfs.mkdir hdfs.exists hdfs.ls

`hdfs.rm` *Removes an HDFS object including all its data.*

Description

Removes all data associated with the specified HDFS object identifier from HDFS including ORCH metadata. This will invalidate all HDFS object identifiers pointing to this HDFS data folder and any ORCH operations using these invalid identifier will result in failures. This function is equivalent to "hadoop fs -rmr" shell command.

Usage

```
hdfs.rm(dfs.id, force = FALSE, notrash = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier of the data to be deleted. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS-compliant directory path relative to the current working directory.
`force`	Set this argument to TRUE to disable confirmation of '*' deletion, don't HDFS I/O check errors, and do not return result.
`notrash`	HDFS has a feature to move deleted data to a trash bin. In order to disable this feature and permanently delete an HDFS object set this argument to TRUE. It is required to set it to TRUE if an object gets deleted from a mapReduce object due to Hadoop job restrictions.

Value

TRUE if data was successfully deleted or FALSE if any error was detected. In case of failure HDFS state is not consistent, the data may not be deleted, or only a portion of the data may be deleted. If `force` is set to TRUE then the function returns the result invisibly.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.rmdir hdfs.exists hdfs.ls

`hdfs.root` *Gets (or sets) ORCH HDFS root directory.*

Description

ORCH allows users to set a custom HDFS root directory which is different from default "/". This allows the creation of an isolated working space for an ORCH user or to map HDFS into one of the local folder if Hadoop is running in a standalone mode which is normally used to setup a test environment.

Usage

```
hdfs.root(dfs.path)
```

Arguments

`dfs.path`	Optional new HDFS root absolute path. If specified then the function will set the new root before returning its values.

Details

Any absolute HDFS path in ORCH is always relative to the current HDFS root, e.g. ORCH path "/a/b" when HDFS root in ORCH is set to "/tmp/hdfs" will actually result is accessing "/tmp/hdfs/a/b" absolute Hadoop's HDFS path. User is not allowed to access anything above the HDFS root path and ORCH will error out if such attempt is detected.

Upon startup ORCH will set HDFS root to "/" if Hadoop is running in distributed or pseudo-distributed mode, or will set HDFS root to "/tmp/hdfs" if Hadoop is running in standalone mode.

Value

HDFS root directory currently configured in ORCH or NULL if HDFS is not connected or not functional.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

hdfs.setroot hdfs.pwd hdfs.cd

---

hdfs.sample *Samples data in HDFS and returns the sample as an R in-memory object.*

Description

Copies the specified number of **arbitrary** records (or lines) from an HDFS directory into an R in-memory object of the type identified by ORCH metadata for this HDFS object. All original R data attributes like column names, data types, etc., will be restored if they are specified in the ORCH metadata. Otherwise, the generic, automatically generated attributes produced by hdfs.attach will be assigned. For example, attributes may have column names like "val1", "val2", or as defined by the user perhaps via hdfs.meta.

Usage

```
hdfs.sample(dfs.id, n = -1000L, level = 5)
```

Arguments

`dfs.id`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory.
`n`	Number of records (or lines) to sample, default is 1000. It is not guaranteed that the result will contain exactly this number of lines though. Specifying `n`=0 will return 0 records and should be used to retrieve data structure attributes like columns names of a data.frame. Specifying a negative value means "at least", e.g. `n`=-1000L will try to retrive 1000 records or more.
`level`	Number of HDFS part files to sample, higher numbers assures better and close to normal distribution but linearly slows down the response time of the function.

Details

The function is similar to hdfs.get but obtains only a subset of rows (or lines) from an HDFS directory. Although named "sample", this function does not obtain a truly random sample, where all rows are equally likely to be selected. hdfs.sample allows a user to obtain a data subset that can be loaded into R's memory for viewing or manipulation. Usage of this function instead of hdfs.get is advised when the HDFS files are too large to fit in R memory.

Key/value separator is not required for this function as it is stored alongside with the data itself and will be retrieved automatically from its ORCH metadata. This also means that HDFS directory must be attached at least once (via hdfs.attach) before using this function so the metadata will be generated if needed.

Value

A data.frame object in memory in the local R environment containing the sampled data set, or NULL if the operation has failed for some reason.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.get hdfs.download hdfs.pull

`hdfs.setroot` *Sets new HDFS root directory in ORCH.*

Description

Sets a new HDFS root directory. This feature is specific to ORCH only and does not change Hadoop's HDFS behavior in any way. All HDFS paths and operations within ORCH infrastructure are relative to the current HDFS root and user cannot change current working directory above its root. For more details see hdfs.root.

Usage

```
hdfs.setroot(dfs.path)
```

Arguments

`dfs.path`	An absolute path in HDFS file system to be set as current HDFS root. If this argument is not given then the user's HDFS home directory will be used as the root (also set by default at ORCH startup).

Value

Current HDFS root path or NULL if there was an error and root was not set to the new value. This can happen if `dfs.path` path is invalid or does not exist in HDFS file system.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.root hdfs.pwd hdfs.cd

`hdfs.size` *Returns total size of an HDFS object in bytes.*

Description

Inspects the specified HDFS object and returns total size of all its data files in bytes or in human-readable form if `units` argument is specified. Non-existing HDFS objects will report size NULL without any error.

Usage

```
hdfs.size(dfs.id, units = NULL)
```

Arguments

`dfs.id`	HDFS object identifier to inspect. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory.
`units`	If specified then output value is converted into a human-readable form. `units` can be one of the following values: "KB", "MB", "GB", "TB", or "PB".

Value

Total size of the HDFS object in "unit" bytes, or 0 if object does not have any data but anyway exists in HDFS, or NULL if object does not exist in HDFS.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.parts hdfs.ls

`hdfs.sync` *Synchronizes ORCH HDFS cache with Hadoop HDFS file system.*

Description

ORCH maintains its own cached mini-snapshot of HDFS to minimize requests to HDFS APIs and improve response of ORCH functions. In case ORCH cache is out of sync with current HDFS state this function can be used to reset ORCH cache and force re-caching of HDFS mini-snapshot.

Usage

```
hdfs.sync(dfs.id)
```

Arguments

`dfs.id`	HDFS object identifier with which cache must be synchronized, if it's not specified then whole HDFS cache will be reset. This is a special ORCH object representing a directory in HDFS, or it can be a string with HDFS-compliant path relative to the current working directory.

Details

At this moment only ORCH metadata stored alongside with an HDFS object is cached. This improves response time of most of the HDFS access API functions.

Value

None.

Attention

This function must be used when an external change of the HDFS object by another user or 3rd party process is expected to modify its content.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.cache

Examples

```
# Metadata is cached on write:
x <- hdfs.put(cars)
# ~0s, metadata is read from the cache:
system.time(hdfs.meta(x))
# Delete cache for this object only:
hdfs.sync(x)
# ~2.5s, metadata is read from HDFS and cached:
system.time(hdfs.meta(x))
# ~0s, metadata is read from the cache:
system.time(hdfs.meta(x))
```

`hdfs.tail` *Reads unformatted tail of an HDFS object.*

Description

Reads the last `n` lines of the specified HDFS object and returns without applying parsing or formatting. Due to HDFS design restrictions tail is concatenated from tails of each part file of the HDFS object, not the real `n` last lines of an HDFS file. This function is equivalent to "hadoop fs -tail" shell command.

Usage

```
hdfs.tail(dfs.id, n = 0L)
```

Arguments

`dfs.id`	HDFS object identifier to get tailing data for. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS-compliant directory path relative to the current working directory.
`n`	Number of tailing lines to return. Must be >= 0. If 0 is specified (default value) then will return a default tail portion of one last part-file which will give the fastest possible execution time. The default size is defined by Hadoop's default "-tail" command size and normally equals to 1KB of total "raw" data.

Details

User should understand that HDFS is a streaming file system and designed and optimized for streaming data from beginning of a file to its end. Getting a tail portion of an HDFS file is not usual operation and in certain conditions cannot be performed. If such cases ORCH may fall back to reading tail portions of several part-files in the same HDFS directory or to reading of head portions of part-files in order to satisfy [n] condition.

The function performance will degrade based on two factors - number of part files in the input HDFS directory (e.g. HDFS object) and size of each part file. Performance approximately linearly degrade with increase of number of HDFS data files and with size increase of each data file but with a cutoff at approximately 100KB after which file size increase will not significantly change its runtime anymore.

Value

Character vector of the specified length `n`. The length can be less than `n` if the specified number of lines cannot be retrieved for some reason. If the HDFS directory has no non-empty data files then a 0-size character vector will be returned. NULL is returned if the HDFS object's directory does not exist or an error has occurred.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.head hdfs.sample hdfs.get hdfs.download

`hdfs.toHive` *Converts an ORCH's HDFS object identifier to a HIVE table represented by ORE's ore.frame object.*

Description

This function is used to convert an HDFS object identifier in ORCH to a HIVE table that is represented by an ORE frame object. The returned ore.frame object can be used with ORE transparency layer in ORCH.

Usage

```
hdfs.toHive(dfs.id, table = NULL)
```

Arguments

`dfs.id`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions which represents a directory in HDFS. Or it can be a string with HDFS-compliant path relative to the current working direcotry.
`table`	A character string representing the target HIVE table name. If `table` is NULL (default), a table with temporary name is created which is dropped at the end of the R session or when the ore.frame associated with the table is garbage collected. If the table needs to be preserved accross sessions, a non-NULL `table` argument must be passed in.

Value

Returns the ore.frame object representing the HIVE table.

Attention

ORE HIVE supports factor types within R but, in HIVE, the factor columns, are of "string" type. If the input has one or more "factor" columns, they will be automatically changed to "character" type without changing any values. The user needs to de-factorize the input data first converting integer values to strings before calling hdfs.toHive in order to preserve original values. Refer to the ORCH manual for supported ORE HIVE types.

Attention

HDFS datasets that use different delimiter for the key column and value columns cannot be converted into HIVE tables because HIVE tables can use uniform delimiters only. User will need to convert it into uniform delimited representation before passing it to hdfs.toHive.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.fromHive

Examples

```
# Upload "cars" dataframe to HDFS.
x <- hdfs.put(cars, key=NA)

# Create a HIVE table corresponding to x.
y <- hdfs.toHive(x)

# Create a HIVE table named cars_temp.
z <- hdfs.toHive(x, "cars_temp")

# Print the values.
print(y)
print(z)

# Remove created Hive tables.
ore.drop(table="cars_temp")
hdfs.rmdir(x)
```

`hdfs.toRData` *Converts an HDFS text object into HDFS binary object.*

Description

This function allows the user to convert a text HDFS data object into ORCH's proprietary binary format based on R's RData binary format. It will execute a mapReduce job that reads an HDFS directory attached to ORCH as HDFS object and containing text files (see hdfs.attach) and outputs the same data but in a special ORCH binary format.

Usage

```
hdfs.toRData(dfs.id, out.name = NULL, overwrite = FALSE,
  parts = NULL, split = NULL, silent = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier of the input data to be converted. This is a special ORCH object returned by hdfs.attach and other functions which represents a directory in HDFS. Or it can be a string with HDFS-compliant path relative to the current working directory.
`out.name`	Output HDFS directory name or an HDFS object identifier of the output converted binary data. Note that the output directory must not exist otherwise the function will fail. See `overwrite` for more details. If the output directory is not specified a temporary one will be created in HDFS "/tmp".
`overwrite`	Allows overwriting of the output HDFS directory if already exists with the same name. By default overwrite is disabled for safety of data manipulations.
`parts`	Number of desired output "part" files. This option directly controls the size of each "part" file which will approximately equal to the total output size / number of "part" files. The function will try to satisfy the specified requirement but does not guarantee it due to the Hadoop jobs execution restrictions. If not specified it will rely on Hadoop's default behavior and will generate a part file per each input part file or HDFS split whichever size is less.
`split`	Maximum number of records per each RData payload. If number of records in the input text "part"-file is larger than the specified `split` size then the output binary "part"-file will have multiple data.frame structures with `split` records or less in each data.frame stored in RData format. This allows ORCH to read the data.frame by chunks limiting memory usage and improving overall performance. Refer to `map.split` and `reduce.split` configuration options of mapred.config.
`silent`	Do not print information messages to console. Do not print final attach summary at the end of the run. Do not ask to rebuild the binary data if user tries to change the splitting or other binary data characteristic.

Details

The binary format is readable by ORCH mapReduce R jobs only and gives the advantage of fastest achievable data read and write throughput in the jobs. Data can be loaded directly into R memory in the mapper or reducer without any parsing or conversion of text into R objects.

Binary R data can be partitioned into the specified number of "part" HDFS files and each "part" file can be split internally into several binary RData chunks of requested size (see `split` argument). When running a mapReduce job with the binary RData input the data is loaded by the chunks and splitting "part" HDFS files allows to limit memory usage and improve performance if the mapper or reducer function does not need to read the whole input data at once.

Value

HDFS object identifier if data was successfully converted to binary, otherwise NULL if any conversion error has occurred.

Note

Output of the function is always pristine. If input HDFS data is not pristine the function will remove all not clean and invalid rows from the data set and output clean filtered data only.

See Also

hdfs.fromRData

`hdfs.toRDD` *Converts an HDFS object into Spark's RDD object.*

Description

The function consumes a standard ORCH HDFS object and returns a compatible HDFS object that points to the same data set in HDFS but also contains a reference to an external Spark RDD object that was created out of this HDFS object. The returned RDD object can be used in all ORCH functions the same way as any non-RDD attached HDFS objects. but in addition it can be used in Spark-enabled analytics and Spark-specific APIs.

Usage

```
hdfs.toRDD(dfs.id, cache = FALSE)
```

Arguments

`dfs.id`	A non-attached to Spark HDFS object identifier. If the object was attached it will be reused.
`cache`	Forces caching of the HDFS data into Spark's memory.

Value

HDFS object identifier with attached Spark RDD object.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

spark.connect

`hdfs.upload` *Uploads a local file or directory into HDFS.*

Description

This is the simplest and fastest possible way to transfer data to HDFS from local storage. It just copies a local file or replicates a local directory into HDFS directory. By default if `dfs.id` and `dfs.file` are not specified the target HDFS directory get a unique ID and the HDFS file(s) are named as "part-12345". If any of the uploaded local files are larger than `split.size` argument (in bytes) the file automatically gets split into several smaller "part" files.

Usage

```
hdfs.upload(filename, dfs.id = NULL, dfs.file = NULL,
  overwrite = FALSE, header = FALSE,
  split.size = .orch.env$split.size, attach = TRUE, ...)
```

Arguments

`filename`	Local file names or directory names as a vector to put to HDFS. If directory is specified then all files in this directory will be uploaded into HDFS. You can mix file and directory names.
`dfs.id`	Name the target HDFS directory, or HDFS path relative to the current working directory, or HDFS object identifier. If the directory does not exist in HDFS it will be created. If it exists `overwrite` parameter must be considered.
`dfs.file`	Vector of strings that specifies desired names of files uploaded to HDFS. Its length must be the same as number of file to be uploaded into HDFS or 1 in which case it will be used as a prefix for every HDFS file name. If not specified or NULL then HDFS file will be named as "part-12345".
`header`	TRUE if local files have a header in the first line which should be removed before uploading to HDFS. You can specify number of rows to remove by assigning a numeric value to this argument too.
`overwrite`	Enable replacing of HDFS directory and/or file if already exist. By default replacing is disabled.
`split.size`	Maximum size in bytes of each HDFS "part" file or 0 to disable splitting. By default it is set to 10MB.
`attach`	Automatically attach the uploaded file as an HDFS object. See hdfs.attach for more details.

`...`	Parameters passed to hdfs.attach. Used only if `attach` == TRUE. See hdfs.attach for more details. • key.sep Key field separator character, ORCH system "\t" by default. • value.sep Value field separator character, ORCH system "," by default. • trim TRUE to ignore trailing empty fields. If HDFS data is suspected to have empty trailing columns like ",,,," this option allows to detect and exclude such redundant columns for the data description in metadata and its structure. • data.frame If TRUE enforces the class of the attached HDFS data to be "data.frame". Otherwise the class can be automatically recognized as "vector", or "matrix", or "data.frame". • silent Do not print information messages to console. Do not print final attach summary at the end of the run.

Details

Delimiters `key.sep` and `value.sep` are specified only as a "hint". ORCH will copy the local files as-is anyway and will automatically create its metadata with the specified delimiters. The content of the file copied into HDFS will not change. If you specify incorrect set of delimiters then attach of the copied data will fail. If you do not specify the delimiters then current ORCH defaults will be used.

Value

HDFS object identifier of the loaded data if it was attached. HDFS absolute path to the uploaded data if it wasn't attached. NULL if error occurred.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.download hdfs.put hdfs.get

`hdfs.valuesep` *Gets or sets default value fields separator.*

Description

Returns the currently configured or sets a new value of the system wide default value separator. The value separator is used in HDFS text based files to separate individual value fields from each other. Examples of input data that use the value separator are:

- key\tvalue1<values_separator>value2... Two values
- key\tvalue Key and one value, no value separator.
- value One value only, no separators at all.

Usage

```
hdfs.valuesep(value.sep)
```

Arguments

`value.sep`	Optionally a new value separator value to set. Must be one character. If not specified then the function will only return the current value set.

Details

Keep in mind that the value separator can be altered at the time of writing data to HDFS for each specific object. The value separator is stored in HDFS object's metadata and default system-wide value is not used at the time of reading this object back from HDFS in ORCH. This default value is used only when user does not specify the value separator explicitly in the function call for the following operations:

- Writing a new dataset to HDFS.
- Attaching existing HDFS data which does not have any metadata.
- Attaching existing HDFS data with metadata missing value separator.

Value

Currently configured system-wide value separator. Upon ORCH startup it is set to a comma character ",".

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.keysep hdfs.delim

`is.hdfs.id` *Tests if an R object is interpretable as HDFS object identifier.*

Description

Verifies if the R object specified by `x` contains ORCH type HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Returns TRUE if `x` contains an HDFS object identifier, otherwise false.

Usage

```
is.hdfs.id(x)
```

Arguments

`x`	An R object of length 1, not NULL.

Value

TRUE if x is "dfs.id" type object, FALSE otherwise

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.id hdfs.attach

`is.rdd.id` *Tests if an R object is interpretable as an HDFS object identifier and is attached to Spark containing corresponding Spark's RDD object.*

Description

Verifies if the R object specified by `x` contains ORCH type HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS.

Usage

```
is.rdd.id(x)
```

Arguments

x	An R object of length 1, not NULL.

Details

In addition tests that the HDFS object was attached to Spark session and contains corresponding Spark's RDD object. This is a special ORCH object returned by hdfs.toRDD function which can be used after an HDFS object is attached with hdfs.attach.

Returns TRUE if `x` contains an HDFS object identifier which is attached to Spark's session, otherwise FALSE.

Value

TRUE if `x` is of HDFS object identifier type and attached to the current Spark session, otherwise FALSE.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.attach hdfs.toRDD

`mapred.config` *Hadoop's mapReduce job configuration class.*

Description

This class contains a number of advanced configuration options one can use to adjust and fine tune a mapReduce job launched with hadoop.run or hadoop.exec ORCH functions in case the out-of-the-box ORCH job setup is not satisfactory.

Slots

`job.name`**:** Name of the mapReduce job. If the name is not specified then Hadoop's default job_ID will be used. It's a good idea to assign a name as it enables you to locate your job in the Hadoop execution logs if required.

`map.tasks`**:** Number of map tasks to run in the job. This option directly sets Hadoop's property "mapred.map.tasks". This is more a hint for Hadoop and the real number of mappers run may be less or more if Hadoop decides.

`reduce.tasks`**:** Number of reduce tasks to run in the job. This option directly sets Hadoop's property "mapred.reduce.tasks". This is more a hint for Hadoop and the real number of reducers run may be less or more if Hadoop decides.

`min.split.size`: Changes HDFS split size which is by default equal to HDFS block size (typically 64MB). Split size indirectly controls number of mappers and reducers launched by Hadoop as it defines the minimum size of data given to a map or reduce task. This option sets Hadoop's property "mapred.min.split.size".

`task.timeout`: Maximum time in seconds a map or reduce task is allowed to run before it gets force killed by Hadoop. Default value is 600 seconds. This option sets Hadoop's property "mapred.task.timeout".

`skip.na.recs`: This option enables a cleanup procedure in ORCH driver. Any input record containing NA in any of its fields will be removed and mapper and reducer user's function will receive a clean dataset. This is useful when user's R code does not handle NA correctly.

`map.valkey`: Include keys as part of values for mapper. When a data set is copied into HDFS one of its data columns can be used as a key. At this case values given to a mapper function will not have key values. If user's code expects original data structure with key column present in values this option should be enabled.

`map.filter`: Works in conjuction with `map.valkey` and indicates to the ORCH driver that mapper output should be exactly the same structure as given to it as input and the only operation it performs is filtering of some of the records. If `map.valkey` was enabled and key was inserted in values then it will automatically be removed from the mapper output.

`reduce.valkey`: Include keys as a part of values for reducer. When a data set is copied into HDFS one of its data columns can be used as a key. In this case values given to a reducer function will not have key values. If user's code expect original data structure with key column present in values this option should be enabled.

`reduce.filter`: Works in conjuction with `reduce.valkey` and indicates to the ORCH driver that reducer output should be exactly the same structure as given to it as input and the only operation it performs is filtering of some of the records. If `reduce.valkey` was enabled and key was inserted in values then it will automatically be removed from the reducer output.

`map.input`: R data type name expected by user's map function as one of the following values:

- "data.frame": Native ORCH data type. Input data can have different types of columns.
- "matrix": Input data is converted into matrix, if any column is "character" then all values in the matrix will be converted into "character" data type. Otherwise, the usual coercion hierarchy (logical < integer < double < complex) will be used, e.g., all-logical data frames will be coerced to a logical matrix, mixed logical-integer will give an integer matrix, etc.
- "vector": Input data is converted into a vector on row-by-row basis. E.g. c(row1-col1, row1-col2, ..., row2-col1, ...). The same data type conversion rules as for "matrix" input types are applied.
- "list": Input data is converted into a list on row-by-row basis. E.g. list(list(row1-col1, row1-col2, ...), list(row2-col1, ...), ...). All data types are preserved. This input mode should be used only for backward compatibility of pre ORCH-2.1 scripts or for unstructured data where field number and types are different from row to row.

`map.output`: Definition of the mapper output format in the form of data.frame. User's mapper function can output via orch.keyvals or orch.keyval an arbitrary data structure. For ORCH to correctly configure Hadoop job and its data stream parser for running the reduce job (or in case of map-only job to store ORCH metadata alongside with output HDFS dataset) it needs to know its structure upfront. If `map.output` is not specified then ORCH will assume that mapper output has the same structure as input data. Template must be provided in the following form:

- template := data.frame([<columns>])
- columns := [key,]<value>[,<columns>]

- key := key=<key_type>
- key_type := NA | "none" | R scalar object
- value := <value_name>=<value_type>
- value_type := "character" | "factor" | R scalar object

Specification data.frame(key=NA) is a special case and only tells the framework that mapper output will be key-less but does not specify the format. For example, if a mapper writes a data.frame with no key column, an integer column "a", a numeric column "b" and a character column "c", in that order, then `map.output=data.frame(key=NA, a=1L, b=1.0, c="a")`

`map.split`: Number of records to supply at once to a mapper. In order to limit memory usage and prevent R running out of memory user can set an upper limit to the number of rows ORCH driver can supply to a user's mapper function at once. The last invocation may have fewer rows due to split boundary. Values accepted:

- >0: Upper limit. An in-memory buffer will be used to accumulate the required number of records and released each time a chunk of data is given to the mapper.
- -1: No limit, give all data to the mapper. All input data will be accumulated in memory, converted into the target data type and then given to the mapper.
- 0: Give the same data size as ORCH read buffer. This is a pass-through mode assuring the lowest memory usage but in this mode there are no guarantees about size of data given to the mapper as it can range from 1 to all input rows.

`map.eos`: Send End Of Stream signal to a user's mapper function once all data was streaming in and given to the function. The very last invocation of the mapper will be with NULL key and NULL values indicating EOS condition. This is useful if the mapper has to perform special actions or output specific data at the very end.

`reduce.input`: R data type name expected by user's reduce function. Can have one of the following values: "data.frame", "matrix", "vector", "list". For more detail see `map.input` definition.

`reduce.output`: Definition of the reducer output format in a form of data.frame. User's reducer function can output via orch.keyvals or orch.keyval an arbitrary data structure and for ORCH to correctly configure Hadoop job and to store ORCH metadata alongside with output HDFS dataset it needs to know its structure upfront. If `reduce.output` is not specified then ORCH will sample the output data and automatically attach it, which results in generating ORCH metadata. For more detail see `map.output` definition. Specification data.frame(key=NA) is a special case and only tells the framework that reducer output will be key-less but does not specify the format.

`reduce.split`: Number of records to supply at once to a reducer. In order to limit memory usage and prevent from R running out of memory a user can set an upper limit to the number of rows ORCH driver can supply to the user's reducer function at once. The last invocation may have fewer rows due to split boundary limitation. Note that if there are more values with the same key than `reduce.split` limit then key block will be split into parts and the reducer function must correctly handle duplicated key blocks. For more details see `map.split` definition.

`reduce.eos`: Send End Of Stream signal to a user's reducer function once all data was streaming in and given to the function. The very last invocation of the reducer will be with NULL key and NULL values indicating EOS condition. The purpose of this configuration setting is to handle the scenario where the number of rows per key is too large to fit in the reducer memory. This setting allows reduce code to deal with chunks of rows at a time with a EOS flagging end of input.

`verbose`: Produce verbose Hadoop execution log. This option directly sets Hadoop's command line argument "-verbose".

`hdfs.access`**:** Enables ORCH usage of all "hdfs." commands inside of mapReduce job. This allows users to read/write and do any other HDFS file system manipulation normally available to a user in ORCH client but inside of server-side mapper and reducer user's function. By default current working HDFS directory in every mapper and reducer will be set to the same path as in ORCH client at the moment the Hadoop job is launched.

`output.quoted`**:** Tells that output of the mapReduce job uses quoted notation and specifies quoting character at the same time. For instance in `output.quoted` = "'" this means that data can contain records like "a,'b,c',d", where 'b,c' is one field. If quote is not set correctly then output data may not be parsed when reading it back in ORCH or another mapReduce job. The value is stored in ORCH metadata alongside with the main dataset in HDFS. If this is a map-only job then the user's mapper function output is considered as quoted, otherwise user's reducer output.

`output.pristine`**:** Tells ORCH that output of the mapReduce job is expected to be "pristine". Definition of "pristine" data is that every character value in each field stored in HDFS when converted to its column data type as specified in ORCH metadata would not produce `NA` result except special values "NA" and "". For Example, if a column type is "numeric" and if there is any empty or non-convertible to numeric value in its data then whole data set can't be considered as "pristine". Having data set in "pristine" mode greatly improves data read and parse performance in ORCH mapReduce job. But specifying a non-conforming data as "pristine" will cause Hadoop job execution failure. If this is a map-only job then the user's mapper function output is considered as "pristine", otherwise user's reducer output is marked as pristine.

`output.key.sep`**:** Output key field separator character, "\t" default. For uniform separators set it to the same value as `output.value.sep`. The key separator is stored in ORCH metadata. If this is a map-only job then this setting is applied to the user's mapper function output, otherwise user's reducer output.

`output.value.sep`**:** Output value fields separator character, "," default. For uniform separators set it to the same value as `output.key.sep`. The value separator is stored in ORCH metadata. If this is a map-only job then this setting is applied to the user's mapper function output, otherwise user's reducer output.

`mapred.quoted`**:** Tells that output of the user's mapper uses quoted notation and specifies quoting character at the same time, e.g. `mapred.quoted` = "'". This values is used only to parse data correctly in following reduce jobs and not stored in ORCH metadata. If this is a map-only job then it's ignored. For more detail see `output.quoted` definition.

`mapred.pristine`**:** Tells ORCH that output of the user's mapper is expected to be "pristine". This value is used only to parse data correctly in the following reduce jobs and not stored in ORCH metadata. If this is a map-only job then it is ignored. For more detail see `output.pristine` definition.

`mapred.key.sep`**:** Key field separator character between map and reduce jobs, "\t" default. For uniform separators set it to the same value as `mapred.value.sep`. This value is used only to parse data correctly in following reduce jobs and not stored in ORCH metadata. If this is a map-only job then it's ignored. For more detail see `output.key.sep` definition.

`mapred.value.sep`**:** Value field separator character between map and reduce jobs, "," default. For uniform separators set it to the same value as `mapred.key.sep`. This value is used only to parse data correctly in following reduce jobs and not stored in ORCH metadata. If this is a map-only job then it's ignored. For more detail see `output.value.sep` definition.

`direct.call`**:** This option works only if input data is in RData binary, otherwise the option will be ignored by the ORCH driver. With this option set to TRUE, ORCH driver will bypass any input caching, data splitting and type conversion and will directly pass the data as it's stored in RData to the user's mapper or reducer. The user looses the control of data size and type input to the mapReduce callbacks but gains the fastest throughput.

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

hadoop.run hadoop.exec

Examples

```
## Not run:
    hadoop.run(
        data = dfsRes,
        mapper = function(k,v) {orch.keyvals(NULL,v+1)}
        reducer = function(k,v) {orch.keyvals(NULL,v+1)}
        config = new("mapred.config",
            job.name = "greatest job ever!",
            map.tasks = 10,
            reduce.tasks = 10
            # more config options
        ))

## End(Not run)
```

---

orch.connected *Returns TRUE if ORCH is connected to Oracle Database.*

Description

Returns TRUE if ORCH is connected to Oracle Database.

Usage

```
orch.connected()
```

Value

TRUE if ORCH is connected to Oracle Database, otherwise FALSE.

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.connect orch.dbinfo

`orch.connect` *Establishes connection to Oracle Database.*

Description

This function connects ORCH with an instance of Oracle Database. All following database import and export operation will be performed using this connection. After the connection is established it is validated by reading USER variable from the Database. Displays connection attributes and error message if there was a problem detected. If the user password is not supplied it will be asked in a prompt at connection time and each time a connection with database is required, (i.e., when invoking hdfs.push and hdfs.pull).

Usage

```
orch.connect(user, sid, host, password = NULL,
  port = 1521, secure = TRUE, driver = "sqoop",
  silent = FALSE, dbcon = NULL)
```

Arguments

`user`	The database user name.
`sid`	Oracle system ID (SID) that is used to uniquely identify a particular database on a system.
`host`	The host name or IP address of the database server to connect to.
`password`	The database password (Optional). If not specified then the user will be prompted to enter the password.
`port`	The Database server connection port, default is 1521.
`secure`	Chooses ORCH to Database connection mode. In secure mode ORCH does not store the password and user will be prompted to enter a password on every attempt to access Database. Default setting is TRUE.
`driver`	Choose Database to HDFS data transfer driver, "sqoop" is the default one. Available drivers are: "sqoop", "olh".
`silent`	If true then does not print connection information to the R console. Otherwise will display will print out user, host, port and sid of the established connection. Default setting is TRUE.
`dbcon`	Provide an alternative way to specify all connection parameters as one orch.dbcon object. See orch.dbcon for more details.

Details

By default, `secure` is set to TRUE which means the user is always prompted for the password. In the secure mode, the password is always requested for every attempt made by the user to connect to a database. The `secure` = FALSE mode is envisioned to be used for testing purposes. In this mode, the password is kept encrypted in memory and subsequent APIs that require this password will **not** prompt the user to enter password each time.

In case of connection failure or any other error the connection gets rolled back to the one which was established prior calling this function.

Value

TRUE if the connection was successfully established and validated, FALSE if connection failed.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.disconnect orch.reconnect orch.dbcon

`orch.create.parttab` *Creates a partition HIVE table from hdfs id or a HIVE ore.frame*

Description

This function is used to create a partitioned table from an ORCH-HDFS file or ORE-HIVE `ore.frame` based on named partitioned columns provided as input.

Usage

```
orch.create.parttab(input, partcols, parttab = NULL)
```

Arguments

`input`	This can be one of the following: 1. the ORCH HDFS identifier representing the input HDFS file 2. ore.frame object representing a HIVE table
`partcols`	Vector of column names in the `input` to be used as partitioned columns. `partcols` cannot be NULL or missing.
`parttab`	Optional argument for the partitioned HIVE table name. If this argument is skipped, then a partitioned HIVE table is created with temporary name, which is dropped at the end of the session.

Details

The goal of this functions is to partition `input` based on the partition columns using HIVE. The partitioned directories returned can be used for further ORCH analytical processing (e.g., model building etc.).

Value

This function returns a list of dfs identifiers corresponding to all the partition directory locations in the partitioned HIVE table. Each of the list elements can be used as an input to `hdfs.attach` for further ORCH processing.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

Examples

```
# Create a HIVE table
library(MASS)
ore.create(cement, table="cmnt")

# do filtering and projection on the input
filtered_x <- cmnt[cmnt$x1 > 7 & cmnt$x4 < 45, ]
filtered_x <- filtered_x[, c('x1', 'x2', 'x3')]

#  two column partitioning
part_dirs <- orch.create.parttab(filtered_x,
                      partcols = c("x3","x1"), "cmnt_parttab")

# print the list of partitioned directories
print(part_dirs)

# print the named partitioned table
print(cmnt_parttab)

# put iris data set into HDFS
iris.dfs <- hdfs.put(iris, key=NA)

# partition the above iris data set
part_dirs <- orch.create.parttab(iris.dfs, partcols=c("Species", "Petal.Width"))

# print the list of partitioned directories
print(part_dirs)

# print the data in the first partition
hdfs.get(part_dirs[[1]])
```

`orch.datagen` *ORCH's data generator.*

Description

This function is used to generate an HDFS data set with specific data characteristics for testing of the ORCH functionality and any user-defined mapReduce code. Generates a dataset of approximate `data.size` size GB with `numeric.col.count` numeric (floating point) columns, `integer.col.count` integer columns, `factor.col.count` categorical columns, and `character.col.coun` string columns.

Usage

```
orch.datagen(data.size = 1L * GB, numeric.col.count = 0L,
  integer.col.count = 0L, factor.col.count = 0L,
  character.col.count = 0L, numeric.mean = 0,
  numeric.sd = 1,
  integer.sample.size = .Machine$integer.max,
  integer.sample.zero = 0L, factor.levels = 5L,
  character.length = 80L, character.length.range = 10L,
  part.size = 0L,
  parts = if (part.size == 0L) max(10L, data.size/(10 * G.)) else 0L,
  row.pattern = NULL, percent.na = 0, keys = 0L,
  key.sep = NULL, value.sep = NULL, out.name = NULL,
  overwrite = FALSE, task.timeout = -1L)
```

Arguments

`data.size`	Size of the data (in bytes) to be generated. Note, this number is used to approximate the number of rows in the data set using the other parameters. The output data set is close to the value of `data.size`. Default value is 1GB.
`numeric.col.count`	Number of numeric (floating point) columns in the dataset.
`integer.col.count`	Number of integer columns in the dataset.
`factor.col.count`	Number of categorical (factor) columns in the dataset.
`character.col.count`	Number of string (character) columns in the dataset.
`numeric.mean`	Generated numeric values mean, by default 0.
`numeric.sd`	Generated numeric values standard deviation, by default 1.
`integer.sample.size`	Generated integer values sample size, by default max integer value.
`integer.sample.zero`	Generated integer values 0-value, by default 0.
`factor.levels`	Number of level of the generated factor values.
`character.length`	Generated string values length, by default 80.
`character.length.range`	Generated string values range of length, by default 10.
`part.size`	Required size of each "part"-file in the output dataset. Setting this parameter will configure number of mappers to be run by the ORCH datagen mapReduce job. Note, this parameter is used as a hint to the Hadoop framework. The actual number of mapper tasks launched might be different.
`parts`	Required number of "part"-files in the output dataset. Setting this parameter will configure number of mappers to be run by the ORCH datagen mapReduce job. Note, this parameter is used as a hint to the Hadoop framework. The actual number of mapper tasks launched might be different.

`row.pattern`	This argument can be used in conjunction with arguments `numeric.col.count`, `integer.col.count`, `factor.col.count`, and `character.col.count` arguments to specify order of the columns, or it can be used by itself which will automatically set number of columns of each type. This a string or a vector of characters where each character denotes a columns type: • n: numeric • i: integer • f: factor • c: character
`percent.na`	Percent of values (cells) in the generated data that needs to be missing (`NA`). The generated data will have approximately `percent.na` of the total values as `NA`.
`keys`	If not 0 then a key column will be generated with number of distinct integer values specified by this parameter.
`key.sep`	Key field separator character, default system-wide value is used if this argument is not specified (normally "\t").
`value.sep`	Value field separator character, default system-wide value is used if this argument is not specified (normally ",").
`out.name`	Output HDFS directory name or an HDFS object identifier of the output data. Note that the output directory must not exist when Hadoop job is submitted otherwise the job will fail. If the output directory is not specified a temporary one will be created in HDFS "/tmp".
`overwrite`	Allows overwriting of HDFS objects with the same name. By default overwrite is disabled for safety of data. Default is FALSE.
`task.timeout`	Maximum time in seconds a map or reduce task is allowed to run before it gets force killed by Hadoop. Default value is 600 seconds.

Details

The number of records in the generated data set is calculated using the number of columns and approximate size of each column type when written in HDFS.

The value of numeric columns are generated using normal distribution generator function `rnorm` with `numeric.mean` and `numeric.sd` parameters in `R`. The categories of the factor columns are randomly selected from levels `1`to `factor.levels`. Integer values are generated using `sample` function with `integer.sample.size` parameter and then adjusted

When `percent.na` is non-zero, a set of size equal to number of rows is created for each column. This set has about `percent.na` percent values missing (`NA`). A random sample is then selected from this set.

As a result of the sampling techniques and datatype size approximations used for data generation, the generated data set approximates the input paramters `data.size` and `percent.na`.

Value

HDFS identifier pointing to the directory containing the generated data set. Or NULL if the mapReduce job has failed or any other error occurred.

`orch.dbcon` *Stored database connection object.*

Description

This object stores all RDBMS credentials needed to establish a connection to the database. The object provides a simple and compact way for the user to switch among several databases without entering your credentials each time you re-establish the connection. The current Database connection object can be retreived using the function `orch.dbcon`. This object can be used to connect to the database once again with orch.[re]connect() function. If the database is not connected then `orch.dbcon()` returns an empty dbcon object.

Slots

`ok`: TRUE if the connection is established and validated.

`host`: Hostname, URL, or IP address of the connected RDBMS server, or "" if not connected.

`port`: Server port number (default 1521).

`sid`: Oracle system ID (SID) that uniquely identifies a particular database on a system, or "" if not connected.

`user`: The database user name, or "" if not connected.

`passwd`: The database user password, or "" if either not connected or connected in secure mode.

`secure`: The ORCH to RDMBS connection mode. In secure mode, ORCH does not store the password and the user will prompted to enter a password on every attempt to access the RDBMS. Default setting is TRUE.

`drv`: Hadoop driver ("sqoop" or "olh") to be used for establishing the connection. The same driver will be used for data transfer when hdfs.push and hdfs.pull are invoked.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.connect orch.disconnect orch.reconnect orch.dbinfo

orch.dbg.lasterr

Returns the very last error message reported. Messages considered to be errors are those having severity level ERROR, CRITICAL, or FATAL.

Description

Returns the very last error message reported. Messages considered to be errors are those having severity level ERROR, CRITICAL, or FATAL.

Usage

```
orch.dbg.lasterr(clear = FALSE)
```

Arguments

`clear`	NULL-ify the last error message after returning.

Value

The last error message reported to the ORCH debug logging sub-system. If `clear` is TRUE then returns as invisible value.

orch.dbg.off

Globally disables debugging in ORCH framework.

Description

Globally disables debugging in ORCH framework. `severity` allows to turn off individual message severity logging only. Also disables assertions in ORCH code when debugging is completely disabled.

Usage

```
orch.dbg.off(severity = NULL, assert = NULL)
```

Arguments

`severity` Optional vector of message severity numeric IDs or string names. It can be specified in a format of comma-separated string as well. Accepted configuration values:

- NULL Suspends debugging and all log message until it's resumed with orch.dbg.on() function call. Turns off asserts too. For example: orch.dbg.off().
- vector List of severities to disable individually. Only those severities will not be logged any more. Asserts will be still enabled. For example: orch.dbg.off(c("info","trace")
- string Comma-separated list of severity names to disable individually. Only those severities will not be logged any more. Asserts will be still enabled. For example: orch.dbg.off("info,trace").

- "all" Will completely turn off debugging and reset all enabled and disabled severities. For example: orch.dbg.off("all")

`assert` Enable or disable asserts throughout the code:

- TRUE Keeps asserts enabled.
- FALSE Force disable asserts.
- NULL Default action. If individual severity(ies) is turned off or "all" the assert settings does not change. If debugging is disabled globally then asserts are disabled too.

Value

None.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.dbg.on orch.dbg.assert

Examples

```
## Not run:
    orch.dbg.off()                   # suspend debugging
    orch.dbg.off("all")              # turn off and reset debugging
    orch.dbg.off("warning")          # disable only warning messages
    orch.dbg.off("warning,info")     # disable warning and info messages
    orch.dbg.off(assert=TRUE)        # turn off log but keep asserts

## End(Not run)
```

`orch.dbg.on` *Globally enables debugging in ORCH framework.*

Description

Globally enables debugging in ORCH framework. `severity` allows to set new debug severity level or turn on individual message severity logging. Also enables assertions in ORCH code when debugging is completely enabled.

Usage

```
orch.dbg.on(severity = NULL, assert = NULL)
```

Arguments

`severity`	Optional vector of severity numeric ID or string name. It can be specified in a format of comma-separated string as well. Accepted configuration values: • NULL This means to just enable debugging without changing the the current debug settings. Severity will stay the same as prior turning off orch.dbg.off(). For example: orch.dbg.on(). • "all" will enable all debug output and reset individually enabled / disabled debug severities. For example: orch.dbg.on("all"). • vector A list that indicates a global severity level plus individual severities to enable only. For example: orch.dbg.on(c("error","trace")) • string Comma-separated list of severity names which will indicate a global severity level plus individual severities to enable only. For example: orch.dbg.on("error,trace") • "" If the first value is empty "" then the current severity will not be changed and only additional list severities will be enabled individually. For example: orch.dbg.on(",trace") • "~" If the first value is "only" or "~" then only listed severities will be enabled and global severity level will be set to FATAL. For example: orch.dbg.on("~,info")
`assert`	Enable or disable asserts throughout the code: • TRUE Force enable asserts. • FALSE Keeps asserts disabled. • NULL Default action. If individual severity(ies) is turned off or "all" the assert settings does not change. If debugging is enabled globally then asserts are enabled too.

Value

None.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.dbg.off orch.dbg.assert

Examples

```
## Not run:
    orch.dbg.on()                # resume debugging.
    orch.dbg.on("all")           # log all debug output.
    orch.dbg.on("warning")       # log warnings, errors, and up.
    orch.dbg.on("error,trace")   # log errors and up, plus TRACE only.
    orch.dbg.on(",trace")        # enable TRACE in addition.
    orch.dbg.on("~,info")        # log INFO messages only.
```

```
## End(Not run)
```

`orch.dbg.output` *Sets a new debug log output stream or a file name.*

Description

Allows to set a new ORCH debug log output stream or a file name. If the new output is not specified then sets to `stdout` by default. Upon ORCH startup debug log is set to "\tmp\orch-<user name>.log".

Usage

```
orch.dbg.output(con = "")
```

Arguments

`con`	R connection object to be used as the debug log output, see file for more deails. Also can be a file name as a string, or `stdout()`, or `stderr()`. "" will be considered as stdout().

Value

None.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

`orch.dbinfo` *Prints out current or stored database connection information.*

Description

Displays information about the current or stored Database connection (if `dbcon` argument is specified). This is just an information tool, no results are returned.

Usage

```
orch.dbinfo(dbcon)
```

Arguments

`dbcon`	Optional argument that allows the user to specify a stored database connection object orch.dbcon which he wants to describe. If not specified then the current established connection is used.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.connect orch.disconnect orch.reconnect orch.dbcon

`orch.debug` *Checks or sets mapReduce "debug" mode.*

Description

Checks or sets mapReduce "debug" mode. Debug mode allows the user to simulate mapReduce job run within the same R session from which the job was submitted and set debug breakpoints in their mapper, reducer, combiner or any function that was exported into ORCH mapReduce job environment via `export` argument of the hadoop.run function. When "debug" mode is enabled ORCH will prepare the mapReduce driver script as always but instead of submitting it to a Hadoop cluster it will load the scripts locally and run own local implementation of Hadoop pipeline invoking local user's functions.

Usage

```
orch.debug(onoff)
```

Arguments

`onoff`	TRUE to enable the "debug" mode, FALSE to disable the "debug" mode. If not speficied then the current setting for the "debug" mode will be returned only.

Details

This greatly improves debug-ability of mapReduce jobs allowing users to inspect input and output values of the mapper, reducer, or combiner, do step by step walk through their functions and identify errors and bugs. Users can use built-in R debug tools or any 3rd party debug library of their choice.

Value

Current "debug" mode. If `onoff` argument is not specified then returns the value visibly otherwise invisibly.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.dryrun hadoop.run hadoo.exec

`orch.disconnect` *Disconnects from Oracle Database.*

Description

Drops a connection to the database. After the disconnect the functions that access the database (i.e. hdfs.push, hdfs.pull) will not work and error out on attempt of use as the connection with the database will be broken.

Usage

```
orch.disconnect(silent = FALSE, dbcon = FALSE)
```

Arguments

`silent`	Do not print connection status messages to the R console. Default setting is FALSE.
`dbcon`	Return current Database connection object orch.dbcon after the disconnect is performed. This allows to re-establish the same connection again using this connection object.

Value

Can return two types depending on `dbcon` argument value:

- If `dbcon` argument is set to TRUE then will return the previous Database connection object of class orch.dbcon. The object can be used to connect to the database once again with orch.reconnect function. If the database was already disconnected then will return NULL.
- If `dbcon` argument is set to FALSE then will return TRUE if the connection was successfully dropped or FALSE if the database was already disconnected.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.connect orch.connected orch.reconnect

`orch.dryrun` *Checks or sets mapReduce "dryrun" mode.*

Description

Checks or sets mapReduce "dryrun" mode. Dry run mode allows the user to run mapReduce jobs as shell scripts outside of Hadoop and debug or benchmark the scripts. When "dry run" mode is enabled ORCH will put the input data into a local temprorary directory and generate csh shell compliant command line that will simulate execution of the scripts in Hadoop environmnet via ORCH.

Usage

```
orch.dryrun(onoff, direct.io)
```

Arguments

`onoff`	TRUE to enable the "dry run" mode, FALSE to disable the "dry run" mode. If not speficied then the current setting for the "dry run" mode will be returned only.
`direct.io`	Use direct local file system read/write IO in ORCH driver instead of streaming data in/out via OS stdin/stdout. This eliminates streaming overhead for benchmarking.

Details

The shell command line can be retrieved from ORCH debug log and used standalone outside of ORCH in order to repeat the run or/and debug the map and reduce scripts if there are any failures. For this the user must enable ORCH debug log via orch.dbg.on.

Value

Current "dryrun" mode with attribute "direct.io" indicating current "direct.io" mode. If `onoff` argument is not specified then returns it visibly otherwise invisibly.

Note

`direct.io` can be switched on or off only when "dryrun" mode is enabled. As soon as it's disabled "direct.io" will be turned off too.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.debug hadoop.run hadoop.exec

`orch.export` *Makes R objects from a user's local R session available in the Hadoop execution environment, so that they can be referenced in MapReduce jobs.*

Description

Local objects to Hadoop job export function. Constructs a list of object values and the same assigns object names to the list names. Example: export(a,b) is the same as list(a=a, b=b).

Usage

```
orch.export(..., MODE = NULL)
```

Arguments

`...`	One or more variables, data frames, or other R in-memory objects, by name or as an explicit named definition, in a comma-separated list. If unnamed value is given (e.g. orch.export(1)) which can not be exported then the function will remove this values from the export list and issue a user warning.
`MODE`	Alters export mode in this particular case. Can be "source", "rdata", or ".GlobalEnv". In case of "source" mode all exported R object will be embedded into mapReduce R script as source code. In case of "rdata" all exported R objects will be stored in a binary RData sidecar file, shared between all Hadoop nodes and loaded in mapReduce driver script. ".GlobalEnv" mode re-assigns exported objects to .GlobalEnv namespaces in order to prevent auto-loading of their corresponding packages in the ORCH driver during deserialization. ".GlobalEnv" can be used in conjuction with "rdata", e.g. `MODE=c("rdata",".GlobalEnv")`. The default mode is "rdata".

Value

List of named values that should be exported into mapReduce server-side job. Only variables and named values will be included. If there are no variables to export then NULL will be returned.

Note

You can use this function to prepare local variables for use in hadoop.exec and hadoop.run functions. The mapper, reducer, combiner, init, and final arguments can reference the exported variables.

Important

In ORCH debug mode (see orch.debug) this function may change the list of exported R objects in order to accomodate the debug facility of ORCH framework. For instance all global functions will be excluded from the export list because they are accessible as-is. Functions defined inside of scope of other functions (or other environments) will be exposed in the global R namespace for the same reason.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hadoop.run hadoop.exec

Examples

```
# This code fragment shows orch.export used in the
# export argument of the hadoop.run function:
b <- 2
x <- hadoop.run(seq(1,3),
    export = orch.export(a=1, b),
    mapper = function(k,v) {
        # a and b are accessible in the mapReduce job:
        v <- (v + a) * b
        orch.keyvals(k, v)
    }
)
print(x)
```

`ORCH_HAL_VERSION` *ORCH system control environment variable.*

Description

You can set this ORCH environment variable before starting R and loading the ORCH library. It enables you to override auto-detection of a Hadoop version and to specify the use of an exact version of the ORCH Hadoop Abstraction Layer.

Details

Supported versions are:

- 1: Apache/IDC/Hortonworks 1.*
- 2: Cloudera CDH3u*
- 3: Cloudera CDH4.* with MR1
- 4: Cloudera CDH4.[0-3] with MR2
- 4.1: Cloudera CDH4.4 with MR2
- 4.2: Cloudera CDH5.* with MR2

If ORCH auto-detection cannot identify the Hadoop version then an informational message indicating that `ORCH_HAL_VERSION` is used will be shown to the user upon loading of the ORCH library. If ORCH auto-detection can identify the Hadoop version and it is not consistent with the one specified by `ORCH_HAL_VERSION` version then a **warning** message will be issued upon loading of the ORCH library and the version specified by `ORCH_HAL_VERSION` will be used instead.

If `ORCH_HAL_VERSION` is not set (default), then ORCH uses Hadoop version auto-detection. If it cannot identify the Hadoop distribution or version, then ORCH issues an **error** message and remains in an error state (not initialized). This state prevents HDFS and mapReduce operations from functioning correctly. You must unload ORCH, set the correct value of `ORCH_HAL_VERSION`, and reload ORCH.

Note

If `ORCH_HAL_VERSION` is set to an invalid value, then an **error** message is issued when loading ORCH and the value is ignored. ORCH will continue to operate as if the variable was not set. You can unload ORCH, set the correct value of `ORCH_HAL_VERSION`, and reload ORCH in order to correct this.

You can **override** the HAL version when you are testing ORCH against a new Hadoop distribution. In this case, ORCH loads and initializes, but you may encounter failures when invoking ORCH API functions. ORCH does not provide any functional guarantees as this case.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

Examples

```
## Not run:
  csh: setenv ORCH_HAL_VERSION 0
  bash: export ORCH_HAL_VERSION=0

## End(Not run)
```

ORCH_HDFS_CHECK *ORCH system control environment variable.*

Description

ORCH performs a simple HDFS functional check when loading the library to ensure that HDFS is configured correctly and that a supported version of ORCH Hadoop Abstraction Layer is specified. You can disable this feature either to improve loading time or to proceed even after an error with HDFS interaction is detected.

- 1 | TRUE Performs the HDFS functional check (default).
- 0 | FALSE Skips the HDFS functional check.

Details

If ORCH_HDFS_CHECK is not set (default), then ORCH performs the HDFS checks. If ORCH_HDFS_CHECK is set to an invalid value, then an **error** message is issued upon loading ORCH and the value is ignored, resulting in the default action.

Note

You can **skip** the functional checks if you are testing ORCH against a new Hadoop distribution. If ORCH_HAL_VERSION is not configured correctly and ORCH fails to recognize the new Hadoop distribution, then ORCH remains in an uninitialized state even when ORCH_HDFS_CHECK is set to 0.

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

Examples

```
## Not run:
  csh: setenv ORCH_HDFS_CHECK 0
  bash: export ORCH_HDFS_CHECK=0

## End(Not run)
```

ORCH_JAR_BUILD_NAME *ORCH system control environment variable.*

Description

You can set this ORCH environment variable before starting R and loading the ORCH library. It allows you to override auto-detection of a Hadoop distribution provider and to specify the build name of the ORCH custom Hadoop JAR library. The build name is appended to the ORCH library file name to differentiate distribution-specific versions of the library.

Details

If ORCH can not auto-detect the Hadoop version and HAL then the build name will be set to "" and will default to the library compiled with Cloudera's Distribution of Hadoop.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

Examples

```
## Not run:
  # Force use of HortonWorks-specific library.
  csh: setenv ORCH_JAR_BUILD_NAME hdp
  bash: export ORCH_JAR_BUILD_NAME=hdp

## End(Not run)
```

ORCH_JAR_MR_VERSION *ORCH system control environment variable.*

Description

You can set this ORCH environment variable before starting R and loading the ORCH library. It allows you to override auto-detection of a Hadoop mapReduce API version and to specify the use of the appropriate version of the ORCH Hadoop JAR library.

Details

Supported versions are:

- 1: MRv1.
- 2: MRv2, or YARN.

If ORCH can not auto-detect the Hadoop version and HAL then mapReduce version will default to version 2.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

Examples

```
## Not run:
  # Force use of mapReduce version 1.
  csh: setenv ORCH_JAR_VERSION 1
  bash: export ORCH_JAR_VERSION=1

## End(Not run)
```

`orch.keyval` *Outputs one (key,value) pair from a mapReduce job.*

Description

Inserts one key and value (or a set of values) pair into the ORCH driver's output buffer. All keys and values will be streamed out into HDFS at the end of the job or in arbitrary time points when ORCH decides. Streaming format is based on job configuration and by default will be comma-separated text with keys separated by '\t'. Both `key` and value `...` are optional and may be missing.

Usage

```
orch.keyval(key = NULL, ...)
```

Arguments

`key`	The key. Must be one-value vector or factor only, no complex structures like list of data.frame can be used. NULL value indicates key-less output (like "val1,val2"). "" value indicates no-key output (like "\tval1,val2). See examples.
`...`	Key's value(s). If only one argument is specified then it can be a vector or a list of values. If multiple arguments specified then only primitive types like numeric, integer, etc. can be used, no complex structures like list or data.frame are accepted. All values given will be assigned to the same `key` and written out as one record.

Value

None.

Attention

If you erroneously use orch.keyvals when you have **one** key and values pair then instead of outputting 1 record the function will output N records containing repeating this key and for each values if input data is compatible (for instance one key and a vector of values is given). This will result in incorrect output data format.

Note

One can understand this is function as a "return" expression of a mapReduce user function which does not break function execution. User can invoke this function multiple times in any place of mapReduce R function or do not invoke this function at all which will result in no output at all. Every invocation will push key and values into internal ORCH driver's buffer which will continue to accumulate returned values till the function finishes.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.keyvals hadoop.run hadoop.exec

Examples

```
## Not run:
    # Different ways to invoke orch.keyval:
    orch.keyval(key=1, 1,2,3)      # will write "1\t1,2,3"
    orch.keyval(key=1, c(1,2,3))  # will write "1\t1,2,3"
    orch.keyval(key=NULL, 1,2,3)  # will write "1,2,3"
    orch.keyval(key="", 1,2,3)     # will write "\t1,2,3"
    orch.keyval(key=1)             # will write "1\t"
    orch.keyval()                  # will not write out anything

## End(Not run)
```

`orch.keyvals` *Outputs multiple (key,value) pairs from a mapReduce job.*

Description

Inserts multiple key and value (or a set of values) pairs into the ORCH driver's output buffer. All keys and values will be streamed out into HDFS at the end of the job or in arbitrary time points when ORCH decides. Streaming format is based on job configuration and by default will be comma-separated text with keys separated by '\t'. Both `key` and value `...` are optional and may be missing.

Usage

```
orch.keyvals(key = NULL, val = NULL)
```

Arguments

`key`	The key. Must be a vector or factor of the same length as value argument `...`, no complex structures like list of data.frame can be used. If only one key is specified and value argument `...` multiple records then this key will be replicated for each record. NULL key indicates key-less output (like "val1,val2"). "" key indicates no-key output (like "\tval1,val2). See examples.
`...`	Key's value(s). Can be a data.frame, a vector or factor, a matrix, or a list. If it's a data.frame or a matrix then each row will be treated as a separate (key,value) record. If it's a vector or a factor then each value will be treated as an individual record. If it's a list then each of its element must represent a set of values of one record.

Details

Length of `key` vector and number of rows in values `...` must be the same and combination of corresponding keys and values will form output records. The only exception is `key` may be one value only in which case the same key will be used with every value(s) when outputting pairs.

Value

None.

Attention

If you erroneously use orch.keyval when you have **multiple** key and value pairs then instead of outputting N records the function will output one record containing one key and all values if input data is compatible (for instance one key and a vector of values is given). This will result in incorrect output data format.

Note

This is function can be understand as a "return" expression of a mapReduce user function which does not break function execution. User can invoke this function multiple times in any place of mapReduce R function. Or do not invoke this function at all which will result in no output at all. Every invocation will push key and values into internal ORCH driver's buffer which will continue to accumulate returned values till the function finishes.

Depending on the values `...` data type this function will behave differently. The best option is to use data.frame as value type as it's native storage type in ORCH and will guarantee the best output performance. Next is vector slightly slower, then matrix and list is the slowest one to output.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.keyval hadoop.run hadoop.exec

Examples

```
## Not run:
    # Different ways to invoke orch.keyval:
    orch.keyvals(key=1, 1,2,3)              # will write "1\t1","1\t2","1\t3"
    orch.keyvals(key=c(1,2,3), c(1,2,3)) # will write "1\t1","2\t2","3\t3"
    orch.keyvals(key=NULL, 1,2,3)           # will write "1","2","3"
    orch.keyvals(val=c(1,2,3))              # will write "1","2","3"
    orch.keyvals(key="", 1,2,3)             # will write "\t1","\t2","\t3"
    orch.keyvals(key=c(1,2,3))              # will write "1\t","2\t","3\t"
    orch.keyvals()                          # will not write out anything

## End(Not run)
```

`ORCH_LOG_OUTPUT` *ORCH system control environment variable.*

Description

Controls the ORCH startup log output. If not specified then log will be written to "/tmp/orch-<user>.log" file. This environment variable allows to change the output stream to any other file or to redirect it to stdout which might be helpful for ORCH startup debugging. See orch.dbg.output for details.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.dbg.output

Examples

```
## Not run:
  csh: setenv ORCH_LOG_OUTPUT /tmp/orch.log
  bash: export ORCH_LOG_OUTPUT=/tmp/orch.log

## End(Not run)
```

ORCH_LOG_SEVERITY *ORCH system control environment variable.*

Description

Controls the ORCH startup log severity. If not specified only ERRORs will be logged. If ORCH fails to startup correctly this option may help to identify the issue via more detailed log. See orch.dbg.on for the list of available ORCH log severity levels.

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.dbg.on orch.dbg.off

Examples

```
## Not run:
  csh: setenv ORCH_LOG_SEVERITY all
  bash: export ORCH_LOG_SEVERITY=all

## End(Not run)
```

`ORCH_MAPRED_CHECK` *ORCH system control environment variable.*

Description

ORCH performs a simple mapReduce functional check when loading the ORCH library to ensure that mapReduce is configured correctly and that a supported version of the ORCH Hadoop Abstraction Layer is detected or specified. You can disable this feature either to improve loading time or to proceed even when an error with the mapReduce job submission is detected.

- 1 | TRUE Performs the mapReduce functional check (default).
- 0 | FALSE Skips the mapReduce functional check.

Details

If `ORCH_MAPRED_CHECK` is not set (default), then ORCH performs the mapReduce checks. If `ORCH_MAPRED_CHECK` is set to an invalid value, then an **error** message is issued when loading ORCH and the value is ignored, resulting in the default action.

Note

You can **skip** the functional checks if you are testing ORCH against a new Hadoop distribution. If ORCH_HAL_VERSION is not configured correctly and ORCH fails to recognize the new Hadoop distribution, then ORCH remains uninitialized even if `ORCH_MAPRED_CHECK` is set to 0.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

Examples

```
## Not run:
  csh: setenv ORCH_MAPRED_CHECK 0
  bash: export ORCH_MAPRED_CHECK=0

## End(Not run)
```

`orch.options` *Allow the user to set and examine a variety of global ORCH options that affect the way ORCH computes and displays its results.*

Description

Invoking orch.options() with no arguments returns a list with the current values of the options. Note that not all options listed below are set initially. To access the value of a single option, one should use 'orch.options("<option_name>")'. For example: orch.options("digits")

Usage

```
orch.options(...)
```

Arguments

`...`	List of options and values to updated or to retrieve: • none: Get list of all ORCH options. • options_name=value[, ...]: Set one or several options. • "options_name"[,...]: Get one or several options.

Details

List of supported ORCH options:

- digits: Controls the number of digits to write when writing numeric values into an HDFS file. It is a suggestion only. Valid values are 1...22 with default 15.
- scientific: Either a logical specifying whether elements of a real or complex vector should be encoded in scientific format when writing into an HDFS file, or an integer penalty (see options("scipen")). Missing values correspond to the current default penalty.

Value

List of all ORCH option values if the user options to set are not specified in `...`. If the option are being retrieved, e.g., `orch.options(c("digits","scientific"))` then returns only their values. Otherwise return TRUE if all options were set, or FALSE if any option was not set for some reason. See examples.

Note

`...` parameter can be specified using a vector or CSV string. In all cases it will mean to get one or several attributes. All styles can be mixed and interchanged as needed:

- c("options_name"[,"options_name"[,...]])
- "options_name[,options_name[,...]]"

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

Examples

```
## Not run:
    # Examples of orch.options invokations.
    orch.options()
    orch.options("digits")
    orch.options("digits", "scientific")
    orch.options("digits,scientific")
    orch.options(digits=7)
    orch.options(digits=7, scientific=TRUE)

## End(Not run)
```

`orch.pack` *Encodes any of R object(s) into a string stream friendly format.*

Description

Packs a set of R objects into a text stream-friendly format. The function can be used with any R object regardless of their complexity and structure. Its output is a string in a proprietary format based on base64 encoding which is guaranteed to do not contain any special symbols like "\10","\t","\n", etc. or separators like ",", ",", etc. This packed string is safe to be used as value in orch.keyval and orch.keyvals functions to write out (key,value) pair from a mapReduce R job.

Usage

```
orch.pack(..., COMPRESS = -1L, DEPARSE = FALSE)
```

Arguments

`...`	any R objects to pack.
`COMPRESS`	Allows compression of the input data before proprietary base64 encoding to lower its size. 3 values are accepted: • -1, default: let function decide, be default objects of size >1KB will be compressed; • TRUE: enforce compression always; • FALSE: disable compression entirely. • "auto": Enables base64 "auto" compression setting. Encode engine will compare and choose compressed vs uncompressed encoding based on the encoded object size. Encoding will take longer but guarantees that the output has the smallest size possible. • "smart": Enables base64 "smart" compression setting. It's the same as "auto" will not attempt any compression on small size objects of size <= 1KB.

`DEPARSE`	Controls how to encode complex R objects (like data.frame, list, etc.): • TRUE: The function will deparse the object into R source code in order to serialize the R object. This will produce smaller output size but may have a performance hit during orch.unpack. Also not all R object can be correctly deparsed/parsed (especially custom classes from 3rd party packages) which can case the orch.unpack to produce non-usable resulting objects. This mode is kept for backward compatibility and ability to produce smaller outputs only, otherwise it should not be used. • FALSE, default: The function will serialize the object into a raw byte array. This will produce larger output size but would allow to avoid the performance hit of R source code parsing during orch.unpack.

Details

Syntax is the same as list(...), e.g. you can use it like pack(a=1, b='x'). The main motivation is to provide an ability to output complex data sets from a mapper or reducer. For example orch.keyval(key, orch.pack(anything)).

Value

Custom base64-based encoded string (optionally compressed).

Note

All control parameters (i.e. `COMPRESS`, `DEPARSE`, etc.) are named in UPPER-CASE in order to better differentiate them from user arguments supplied in free-form "..." argument.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.unpack orch.keyval orch.keyvals

Examples

```
x <- orch.pack(10)
orch.unpack(x) == 10
```

`orch.reconnect` *Reconnects to Oracle Database with previous credentials.*

Description

After a user invokes `orch.disconnect(dbcon=TRUE)` to drop the database connection, ORCH returns an `orch.dbcon` object that can be used by `orch.reconnect` to restablish the connection.

Usage

```
orch.reconnect(dbcon, silent = FALSE)
```

Arguments

`dbcon`	The stored database connection object, can be returned by orch.disconnect. See orch.dbcon for more information.
`silent`	Set it to TRUE to not print connection status messages to the R console. Default setting is FALSE.

Details

Reconnect is faster than orch.connect as it does not perform expensive connectivity checks as the initial connect does and relies on the assumption that the database connection was verified once before. Only a quick connection test is performed.

Value

TRUE if connection was successfully established and validated or FALSE if connection has failed.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.connect orch.disconnect

`orch.revision` *Allows to check currently installed ORCH revision number (unique build).*

Description

Allows to check currently installed ORCH revision number (unique build).

Usage

```
orch.revision()
```

Value

Current ORCH revision as a number value.

`orch.sample` *Get a sample of an ORCH HDFS object or ORE HIVE table*

Description

This function is used to get a simple random sample of an ORCH HDFS object. It can also be used to obtain a sample of a HIVE table using HIVE block sampling.

Usage

```
orch.sample(input, percent = 1, output = NULL)
orch.sample(input, size, output = NULL)
```

Arguments

`input`	This can be one of the following: 1. the ORCH HDFS identifier representing the input HDFS file 2. ore.frame object representing a HIVE table
`percent`	Percent of input records desired in the sample. Default is `1`
`size`	Number of input records desired in the sample.
`output`	Character string specifying the location of the sample output

Details

This function is used to get a simple random sample of an ORCH HDFS object. See Simple random sample for details.

When the input is an HDFS identifier, the output is an HDFS identifier containing the sample of the data. The size of the sample desired can either be specified directly through the `size` parameter or can be specified indirectly as a percentage of the input size using the `percent` parameter.

When the desired sample size is specified as a percentage of the input size, a Java map-only hadoop job is used to generate the sample and the whole data is scanned row by row. Java's pseudo-random number generator is used to select or reject a row to be included in the sample. Thus, the size of the sample obtained will only be approximately equal to the desired size, specified through the `percent` argument (as opposed to being exactly equal).

When the desired sample size is specified directly through the `size` argument, a Java Map-Reduce implementation of the Reservoir Sampling algorithm is used. (See Reservoir Sampling for details on the algorithm).The size of the sample obtained will be exactly equal to the specified `size`. It should be noted that the entire sample will need to be held in the memory of the Reducer task. This has to be kept in mind while specifying the `size` argument.

This function can also be used to obtain a sample of a HIVE table using HIVE block sampling.

When the input is an `ore.frame` representing a source HIVE table, the output is an `ore.frame` object representing the sample HIVE table. HIVE uses block sampling, so the granularity of data returned would be at the HDFS block size level. See HIVE sampling for details. Since block sampling is inherently faster than scanning the whole dataset, HIVE sampling is considerably faster than HDFS sampling.

If the output location of the sample is not specified (`NULL`), the sample output is stored in a temporary HDFS location if the input is a HDFS identifier or a temporary HIVE table if the input is a HIVE table. For HIVE table input, the temporary HIVE table is dropped at the end of the R session or when the `ore.frame` object associated with it is garbage collected. For HDFS identifier input, the HDFS sample output is not removed or garbage collected.

If a non-default output parameter is specified, it is treated as the HDFS output location or HIVE table name depending on the input. For HIVE table case, this sample HIVE table is not dropped at the end of session or when the `ore.frame` object associated with it is garbage collected. So, if the sample HIVE table needs to be preserved accross sessions, a non-default output table name must be passed in.

Value

This can be one of the following:

1. the ORCH HDFS identifier representing the sample when the input is HDFS identifier
2. the ore.frame representing the sample HIVE table when the input is a HIVE table

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

See Also

`orch.toHive` `orch.fromHive`

Examples

```
# Create a HIVE table
ore.create(iris[1:4], table="iris1")

y10 <- orch.sample(iris1, percent = 10, output = "samp_out10")
# output sample table contents
print(y10)

# copy iris data set into HDFS
x <- hdfs.put(iris, dfs.name = "/tmp/iris_tmp")
z <- orch.sample(x, percent = 10, output = "/tmp/samp_hdfsout10")
```

```
# print the sample output
print(head(hdfs.get(z)))

# Sample using size
zz <- orch.sample(x, size = 10, output = "/tmp/samp_hdfsout_size10")
print(hdfs.get(zz))
```

`orch.scale` *Scale the columns of ORCH HDFS object or ORE HIVE table*

Description

This function is used to scale the columns of an ORCH HDFS object or a ORCH-HIVE frame

Usage

```
orch.scale(input, center = TRUE, scale = TRUE)
```

Arguments

`input`	This can be one of the following: 1. the ORCH HDFS identifier representing the input HDFS file 2. ore.frame object representing a HIVE table
`center`	It can be a logical value (default = TRUE) or a numeric vector of same length as number columns in `input`
`scale`	It can be a logical value (default = TRUE) or a numeric vector of same length as number of columns in `input`

Details

The value of `center` determines centering method. If `center` is a numeric vector, then corresponding value from `center` is subtracted from each column of `input`. If `center` is `TRUE` then centering is done by subtracting the column means of `input` from their corresponding column values, no centering is done if `center` is `FALSE`.

The value of `scale` determines column scaling method. If `scale` is a numeric vector, then each column of `input` is divided by the corresponding value from `scale`. No scaling is done if `scale` is `FALSE`. If `scale` is `TRUE` then scaling is done by dividing the columns by their standard deviations if `center` is `TRUE`, and the root mean square if `center` is `FALSE`.

If the input is an ORCH HDFS identifier, the scaling operation is performed in HIVE after converting the HDFS identifier to a HIVE table. After the completion of the HIVE computation, a temporary HIVE table storing the scale output is created and an HDFS identifier pointing to this table location is returned. This temporary HIVE table is dropped at the end of the session or during R garbage collection invocation for the HDFS id object.

Value

This can be one of the following:

1. the ORCH HDFS identifier representing the scaled values when input is HDFS identifier
2. the ore.frame representing the scaled values when input is a HIVE table

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

See Also

`scale`

Examples

```
# Create a HIVE table
library(MASS)
ore.create(cement, table="cmnt")

# do filtering and projection on the input
filtered_x <- cmnt[cmnt$x1 > 7 & cmnt$x4 < 45, ]
filtered_x <- filtered_x[, c('x1', 'x2', 'x3')]

# Perform centering but no scaling
x <- orch.scale(filtered_x, center=c(1,2,3),scale = FALSE)
# output scaled values
print(x)

# Create a dfs identifier
dfs.id <- hdfs.put(cement, key=NA)

# Perform  both centering and scaling
x <- orch.scale(dfs.id)
# output scaled values
print(hdfs.get(x))
```

`orch.tempPath` *Changes the path where temporary data is stored.*

Description

This function allows switching to a new temporary directory for security, disc quota or performance reasons. Temporary files are created and deleted when transferring data between R memory or local file system and HDFS.

Usage

```
orch.tempPath(path)
```

Arguments

`path`	The new temporary storage path. By default "/tmp" is used by ORCH. The function will verify the path exists and will abort if it does not.

Value

Current temp path if `path` is missing. If the new path was set its value will be returned invisibly.

`orch.unpack` *Decodes result or orch.pack back to original R object(s).*

Description

Unpacks a set of R objects from a proprietary ORCH string stream friendly format encoded with orch.pack. The main motivation is to provide an ability to output and input complex data types in a mapper or reducer via text-based Hadoop stream.

Usage

```
orch.unpack(vals, as.list = FALSE)
```

Arguments

`vals`	Result of orch.pack function, may be a vector.
`as.list`	Always return a list of unpacked objects even if it contains only one packed object. Otherwise unpacking of one packed object will result in unlisted value.

Value

List of original R object(s) or only its value if one R object was packed and `as.list`==FALSE.

Note

If `vals` contain only one packed object then it will unpack and return this object's value alone as-is (unless `as.list`==TRUE). If `vals` contains several packed objects then will unpack every one of them and return them as a list where the name of each list's element corresponds to the packed variable name.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.pack orch.keyval orch.keyvals

Examples

```
orch.unpack(orch.pack(a=1)) # == list(a=1).
orch.unpack(orch.pack(a=1), as.list=T) # == list(list(a=1))
orch.unpack(rep(orch.pack(a=1),2)) # == list(list(a=1), list(a=1))
```

`orch.version` *Allows to check currently installed ORCH version.*

Description

Allows to check currently installed ORCH version.

Usage

```
orch.version()
```

Value

Current ORCH version as a string value.

`rdd.isCached` *Tests if the given Spark RDD object was cached in memory.*

Description

When an HDFS object is attached to a Spark session it is possible to force Spark to cache its data in memory improving performance of consecutive Spark jobs on this object. Note that it is not guaranteed that Spark will retain cache data in memory and it may uncache it any moment when more free memory is required for current computations. This function tests if the data was "forcibly" cached (by means of `hdfs.toRDD(x,  cache=T)` call for instance) in memory but does not guarantee that the data is still cached.

Usage

```
rdd.isCached(rdd.id)
```

Arguments

`rdd.id`	HDFS object identifier which refers to an in-memory Spark's RDD object.

Value

TRUE if the object was cached in Spark's memory and consecutive Spark jobs on this object will not read any data from a disk, otherwise FALSE.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

hdfs.toRDD

`spark.connected` *Returns TRUE if there is an "active" Spark context. Otherwise returns FALSE.*

Description

Returns TRUE if there is an "active" Spark context. Otherwise returns FALSE.

Usage

```
spark.connected()
```

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

spark.connect spark.disconnect spark.session

`spark.connect` *Creates a new Spark session.*

Description

The function will create a new Spark execution context if `master` contains the URL of Spark master node. If `master` contains an existing Spark session object then the function will reuse its context instead. The created (or reused) session will be set as "active" session for ORCH globally.

Usage

```
spark.connect(master, name = NULL, memory = NULL,
  dfs.namenode = NULL, ...)
```

Arguments

`master`	This can be a Spark master node URL to connect to or an existing Spark context to set as a current execution context.
`name`	Name of a new Spark execution context. This parameter will be used only if `master` is Spark master URL and a new Spark context is created. Default name is "ORCH".
`memory`	Amount of memory to use per executor process, in the same format as JVM memory strings (e.g. 512m, 2g). Or in byte if specified as integer. Default is 2 GB.
`dfs.namenode`	Default HDFS Namenode to use when converting and HDFS object into RDD object. If not set, then the current local file system will be used.
`...`	Any of the additional spark properties can specified within the `spark.connect` call as property="value" pairs if needed. See `http://spark.apache.org/docs/latest/configuration.html#available-properties` for the complete list and description of all spark properties. Few spark properties are also described further below. 1. spark.executor.cores : The number of cores to use on each executor. For YARN and standalone mode only. In standalone mode, setting this parameter allows an application to run multiple executors on the same worker, provided that there are enough cores on that worker. Otherwise, only one executor per application will run on each worker. 2. spark.cores.max : When running on a standalone deploy cluster, the maximum amount of CPU cores to request for the application from across the cluster (not from each machine). If not set, the default will be spark.deploy.defaultCores on Spark's standalone cluster manager. 3. spark.akka.threads : Number of actor threads to use for communication. Can be useful to increase on large clusters when the driver has a lot of CPU cores.

Details

If there is an "active" Spark execution session (i.e., this function was already invoked once) then it will be "disconnected" (or made non-"active", see spark.disconnect), but it will not be immediately terminated, i.e. Spark resources will remain allocated. If there are no references to this context left anywhere in R the session, it will be deleted at some point by R and Java garbage collectors and all Spark resource will be released.

Value

Invisibly returns a new (or re-connected) Spark session object in case of success, otherwise NULL.

Attention

If `master` contains an existing Spark context object and any Spark parameters have non-default values, i.e., `name`, `memory` was specified, then the context will be created again using the same parameters as the existing one plus non-default user-specified parameters that overwrite the existing ones.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

spark.disconnect spark.connected spark.session

Examples

```
# To use a local Spark pseudo-cluster in your R session (assuming your HDFS
# namenode URI is <mynamenode.example.com:8020>) with 1 GB of memory per
# executor:
## Not run:
spark.connect(master="local[*]", memory="1g",
  dfs.namenode="mynamenode.example.com")

## End(Not run)
# To use Spark on YARN in your R session with 1 GB of memory per executor
# and 2 cores on each executor:
## Not run:
spark.connect(master="yarn-client", memory="1g",
  dfs.namenode="mynamenode.example.com", spark.executor.cores=2)

## End(Not run)
# To use a standalone spark cluster in your R session (assuming Spark master
# is <myspark.example.com:7077>) with 1 GB of memory per executor:
## Not run:
spark.connect(master="spark://myspark.example.com:7077", memory="1g",
  dfs.namenode="mynamenode.example.com")

## End(Not run)
# Sample connection to local spark cluster with local filesystem access:

spark.connect("local[*]")
# Check if connected.
spark.connected()
# Disconnect.
spark.disconnect()
```

`spark.disconnect` *Deletes the current Spark execution context.*

Description

The function does not actually delete the current context but rather makes it non-"active". If there are no references to this context left anywhere in R code it will be deleted at some point by R and Java garbage collectors.

Usage

```
spark.disconnect()
```

Value

Invisibly returns TRUE if there was an "active" Spark execution context and it was "disconnected", otherwise returns FALSE.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

spark.connect spark.connected spark.session

`spark.property` *Returns the value of the Spark property of the "active" Spark execution session object if there is one (i.e., spark.connect function was invoked), otherwise returns NULL.*

Description

Returns the value of the Spark property of the "active" Spark execution session object if there is one (i.e., spark.connect function was invoked), otherwise returns NULL.

Usage

```
spark.property(property)
```

Arguments

`property`	Character string specifying the Spark property value to be queried. See `http://spark.apache.org/docs/latest/configuration.html#available-propert` for the complete list and description of all spark properties.

Value

Value of the `property` from the current Spark Context.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

spark.connect spark.disconnect spark.connected spark.session

`spark.session` *Returns the "active" Spark execution session object if there is one (i.e., spark.connect function was invoked), otherwise returns NULL.*

Description

ORCH Spark session is represented by R class SparkSession and contains a reference to ORCH Java Spark session object. This object includes native Spark's context and a number of ORCH constructs and functions consolidated together in order to accommodate ORCH and Spark integration.

Usage

```
spark.session()
```

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

spark.connect spark.disconnect spark.connected

`hdfs.write` *Writes a distributed model matrix as a CSV file.*

Description

This function is used to write the model matrix created using `orch.model.matrix` as a Comma Separated Values (CSV) file to HDFS, a local file system, or any other Hadoop-compliant abstract file system, which is enabled in the Hadoop configuration. It is also used to write the predictions created using Spark analytics in ORAAH, which include:

- `orch.lm2`
- `orch.glm2`
- `orch.ml.logistic`
- `orch.ml.linear`
- `orch.ml.lasso`
- `orch.ml.ridge`

- orch.ml.svm
- orch.ml.gmm
- orch.ml.kmeans
- orch.ml.dt
- orch.ml.random.forest

Written data in HDFS will preserve all metadata required for retreiving it later from without ORAAH. In order to access written data in an R session you can use hdfs.get.

Usage

```
hdfs.write(data, outPath, overwrite = FALSE)
```

Arguments

data	Distributed model matrix object.
outPath	Destination directory relative to the currently set user's HDFS root path. See link{hdfs.root} function for more information.
overwrite	Whether to overwrite the destination directory if it exists. Default is FALSE.

Value

HDFS identifier object which points to data in HDFS, if data was written.

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

hdfs.attach hdfs.get hdfs.head

Examples

```
library(rpart)
data <- hdfs.put(kyphosis)
modelMatrix <- orch.model.matrix(Kyphosis ~ Number, data = data)
dfs.id <- hdfs.write(modelMatrix, outPath = "destination", overwrite = TRUE)
head(hdfs.get(dfs.id))
hdfs.rm(dfs.id)
```

`orch.evaluate` *Evaluate a fit*

Description

This is a S4 generic method to evaluate a fit. For example, this could be used to help in tuning model parameters by evaluating the fit on a held out cross validation data set.

Usage

```
    orch.evaluate(object, ...)

    ## S4 method for signature 'orch.lmf.jellyfish'
orch.evaluate(object, input, dfs.output = NULL)

    ## S4 method for signature 'orch.mahout.lmf.als'
orch.evaluate(object, input, dfs.output = NULL)
```

Arguments

`object`	An instance of a model
`input`	A CSV ratings file containing entries of the form (user, item, rating). This can be one of the following 1. the HDFS directory containing the input file 2. R data.frame object 3. ore.frame object 4. name of a file in the local file system
`dfs.output`	The output HDFS directory where the error metrics result file should be created. If not specified, this method will internally create a directory and use that as the output directory.

Methods

`signature(object = "orch.lmf.jellyfish")` This function computes the error metrics (SSE, RMSE) on an input ratings file for the input model instance of class `orch.lmf.jellyfish`

Returns a list with the following components -

1. SSE - Sum of Squared Errors
2. RMSE - Root Mean Squared Error
3. inputDir - The HDFS directory containing the input
4. outputDir - The HDFS directory containing the error metrics output

`signature(object = "orch.mahout.lmf.als")` This function computes the RMSE error metric on an input ratings file for the input model instance of class `orch.mahout.lmf.als`

Returns a list with the following components -

1. RMSE - Root Mean Squared Error
2. inputDir - The HDFS directory containing the input
3. outputDir - The HDFS directory containing the error metrics output

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

Examples

```
## Setup the input (user, item, rating) entries
u <- sample(1:100, 300, replace=TRUE)
i <- sample(1:10, 300, replace=TRUE)
ui <- unique(cbind(u,i))
r <- sample(1:5, nrow(ui), replace=TRUE)
input <- cbind(ui,r)

# Fit an "orch.lmf.jellyfish" model
fit1 <- orch.lmf(input, latin=2, iterations=10, rank=3)
print(fit1)

# Evaluate this "orch.lmf.jellyfish" model
se.fit1 <- orch.evaluate(fit1, fit1$inputDir)
se.fit1

# For "mahout-als", set up an input file
inputFile <- tempfile(tmpdir='/tmp')
write.table(input, file=inputFile, sep=",", col.names=FALSE, row.names=FALSE)

# Fit using "mahout-als"
fit2 <- orch.lmf(inputFile, method="mahout-als", rank=3, iterations=5)
print(fit2)

# Evaluate this "mahout-als" model
se.fit2 <- orch.evaluate(fit2, fit2$inputDir)
se.fit2
```

`orch.export.fit` *Export a fit to HDFS*

Description

This is a S4 generic method to export a fit to HDFS

Usage

```
    orch.export.fit(object, ...)

    ## S4 method for signature 'orch.lmf.jellyfish'
orch.export.fit(object, dfs.output = NULL,
    type = c("data.frame", "ore.frame", "hdfs"), leftTableName,
    rightTableName, overwrite  = FALSE)

    ## S4 method for signature 'orch.nmf.jellyfish'
```

```
orch.export.fit(object, dfs.output = NULL,
    type = c("data.frame", "ore.frame", "hdfs"), leftTableName,
    rightTableName, overwrite  = FALSE)
```

Arguments

`object` An instance of a model

`dfs.output` The output HDFS directory where the factor matrices should be created in CSV format. If not specified, this method will internally create a directory and use that as the output directory.

`type` One of:

1. "hdfs" - the factor matrices are exported as CSV format HDFS files
2. "data.frame" - the factor matrices are exported as CSV format HDFS files. Further, the factor matrices are exported as R data.frame objects
3. "ore.frame" - the factor matrices are exported as CSV format HDFS files. Further, the data is transferred to the connected Oracle database and ore.frame objects are returned

`leftTableName` The name of the Oracle database table where the left factor matrix is to be stored. This argument is considered only when "type" is picked as ore.frame.

`rightTableName` The name of the Oracle database table where the right factor matrix is to be stored. This argument is considered only when "type" is picked as ore.frame.

`overwrite` Controls whether the database tables should be overwritten. This argument is considered only when "type" is picked as ore.frame.

Methods

`signature(object = "orch.lmf.jellyfish")` This function exports an `orch.lmf.jellyfish` model. This is done by exporting the L and R factor matrices into CSV format HDFS files and then additionally exporting them either as R data.frame objects or as ore.frame objects based on the user's input.

Returns a list with the following components -

1. Ldir - HDFS directory containing the left factor matrix in CSV format
2. Rdir - HDFS directory containing the right factor matrix in CSV format
3. L - the left latent factor matrix. First column of L is userid. Remaining columns are user features (as many as "rank" used while fitting the model). L will either be a data.frame or ore.frame depending on user's choice
4. R - the right latent factor matrix. First column of R is itemid. Remaining columns are item features (as many as "rank" used while fitting the model). R will either be a data.frame or ore.frame depending on user's choice

`signature(object = "orch.nmf.jellyfish")` This function exports a `orch.nmf.jellyfish` model. This is done by exporting the W and H factor matrices into CSV format HDFS files and then additionally exporting them either as R data.frame objects or as ore.frame objects based on the user's input.

Returns a list with the following components -

1. Wdir - HDFS directory containing the left factor matrix in CSV format
2. Hdir - HDFS directory containing the right factor matrix in CSV format

3. W - the left latent factor matrix. First column of W is row-id. Remaining columns are the basis vectors (as many as "rank" used while fitting the model). W will either be a data.frame or ore.frame depending on user's choice
4. H - the right latent factor matrix. First column of H is column-id. Remaining columns are the coefficients (as many as "rank" used while fitting the model). H will either be a data.frame or ore.frame depending on user's choice

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

Examples

```
## Setup the input (user, item, rating) entries
u <- sample(1:100, 300, replace=TRUE)
i <- sample(1:10, 300, replace=TRUE)
ui <- unique(cbind(u,i))
r <- sample(1:5, nrow(ui), replace=TRUE)
input <- cbind(ui,r)

# Fit an "orch.lmf.jellyfish" model
fit <- orch.lmf(input, latin=2, iterations=5, rank=3)
print(fit)

# Export the model into R data frames
lr <- orch.export.fit(fit)
dim(lr$L)
dim(lr$R)
```

`orch.getFactorLevels` *Factor Levels*

Description

Creates a list of factor levels.

Usage

```
orch.getFactorLevels(formula, dfs.dat, keepSpace = TRUE)
```

Arguments

`formula`	An `orch.formula` object.
`dfs.dat`	An `hdfs.id` object.
`keepSpace`	Whether to keep or remove any leading and trailing whitespace for factor levels.

Details

Creates a list of factor levels. Note: the function supports only the simplest formulae; for instance, interactions and `I()` function are not allowed. Function `F(x)` can be used to ensure `x` will be treated as a factor variable.

Value

A named list containing the factor levels for the categorical variables. The order in the factor levels for each categorical variable is undefined.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

See Also

`getXlevels` `orch.formula`

Examples

```
# Load libraries for examples
library(ORCHstats)
library(rpart)

dfs.dat <- hdfs.put(kyphosis)
levels <- orch.getFactorLevels(Kyphosis ~ Age + F(Number) + Start, dfs.dat = dfs.dat)
```

`orch.getXlevels` *Factor Levels for a Model Matrix*

Description

Creates a list of factor levels that can be used in the `xlev` argument of a `model.matrix` call.

Usage

```
orch.getXlevels(Terms, dfs.dat)
```

Arguments

`Terms`	A `terms` or `formula` object.
`dfs.dat`	An `hdfs.id` object.

Details

This function is the ORCH equivalent to the `getXlevels` function in the **stats** package.

Value

A named list containing the factor levels for the categorical variables derived in the `Terms` argument.

The order of the components of the named list is undefined. The order in the factor levels for each categorical variable is also undefined.

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

See Also

getXlevels

Examples

```
X <- hdfs.put(data.frame(V1 = -1:2,
                         V2 = 1:4,
                         V3 = rep(c("a", "b"), 2),
                         V4 = rep(c("A", "B"), c(2, 2))))
trms <- terms(V1 ~ log(V2) * V3 * V4)
orch.getXlevels(trms, X)
```

---

orch.glm2 *ORAAH Fitting Generalized Linear Models (GLM).*

Description

High performance logistic regression, based on a parallel distributed Iteratively Reweighted Least Squares (IRLS) algorithm. GLM is used to fit logistic regression models.

Usage

```
  orch.glm2(formula, data, relObjDiff = 1e-08,
    relVarDiff = 1e-08, maxIterations = 20L,
    verbose = TRUE, maxBlockRows = 20000L)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`relObjDiff`	Relative difference between objective function values at the current and the previous iterations. By default, `1E-8` is used as a stopping criterion.
`relVarDiff`	Relative difference between solution vectors at the current and the previous iterations. By default, `1E-8` is used as the stopping criterion.
`maxIterations`	Maximum number of IRLS iterations. By default, 20 iterations is used as the stopping criterion.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.
`maxBlockRows`	Maximum number of rows in a partition. Smaller number or rows will create smaller partitions and more paritions. More data partitions ensures higher parallelization degree across Spark cluster but at the same time small paritions will cause higher communication and resource management overhead.

Details

Assuming each training observation comprises an observed class `y[i]` (0, 1), and a vector of features `x[i]`, the logistic regression seeks to maximize the log-likelihood `l(beta0, beta) = sum(-log(1 + exp(beta0 + x[i] * beta))) + sum(y[i] * (beta0 + x[i] * beta))`.

The implementation is based on a parallel distributed Iteratively Reweighted Least Squares (IRLS) algorithm. To carry out IRLS iterations ORAAH GLM utilizes parallel distributed linear algebra algorithms, including parallel supernodal Cholesky factorization,

ORAAH GLM can efficiently handle both numeric and high cardinality factor variables. ORAAH GLM automatically switches to the out-of-core mode, if the input data does not fit into the distributed memory.

Value

GLM2 fit object, `orch.glm2`.

ORAAH Formula

Everywhere below `A` can be either an ID (column name), it can also denote any generated column, for instance `sin(A / 10)`, or any subset of columns for instance `(A1 + A2 + A3)`.

Numerical engines, such as linear regression, cannot consume raw data; there must be a way to specify response and explanatory variables, nonlinear transformations, and interactions. Formula is such an engine, a recipe which specifies which columns (terms) to include to the model, and how to transform them if desired.

- `.` dot-character is a shortcut for all variables (all data columns), except the response.
- `+A` plus operator means to include this variable into the model. Plus operator here is used in set-theoretic sense. There is no arithmetic summation here of any kind. We add a term (column) to an ordered set of statistical terms (model).
- `-A` minus operator means to remove this variable from the model. Example `- (A + B)` removes both A and B variables from the model. Example `. - (A + B)` includes all variables, except A, B, and the response.
- `A : B` include the interaction between A and B variables.
- `A * B` include these variables and the interactions between them. This is equivalent to `A + B + A : B`. `Example A * ( . - B ) * Z`
- `(A1 + A2 + ... + Ak)^n` include these variables and all interactions up to n-way. For instance, `(A + B)^2` is equivalent to `A + B + A : B`. The exponentiation (the power operator) can lead to much more compact model specification. For instance `(. - A)^3` will include all variables, excluding the A and the response, and will include the corresponding interactions. To reiterate, A can be either an ID (variable name) or any complex term. For instance, `(log(A) + B : Z)^2` is equivalent to `log(A) + B : Z + log(A) : B : Z`
- `I()` Identity function. Its argument will be treated in arithmetic sense (as versus set-theoretic sense). For instance: `I(log(A) + B)` will include a new column, whose elements are `log(A[k]) + B[k]`. Here, the plus operator (and all other operators) will be treated in their traditional arithmetic sense.
- 24 arithmetic functions. The argument will be treated in arithmetic sense. Example `log(A / 10 + B * Z)`.

abs	acos	asin
atan	cbrt	ceil
cos	cosh	exp

```
expm1   floor      log
log10   log1p      rint
round   signum     sin
sinh    sqrt       tan
tanh    toDegrees  toRadians
```

- Relational operators, currently supported for numerical terms only. Example Y ~ X + (A > B).

```
A >= B   A <= B
 A > B   A < B
 A == B  A != B
 A && B  A || B
 A & B   A | B
```

- `+1` Add the intercept.
- `-0` Add the intercept (equivalent to +1).
- `-1` Delete the intercept.
- `+0` Delete the intercept (equivalent to -1).

It is very important to keep in mind, that all factor variables (including factor-factor and factor-numeric interactions), are unrolled following one-hot scheme, meaning internally they will be substituted by `k-1` dummy variables.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

`orch.formula` `predict.orch.glm2` `oracle.model.matrix`

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.glm2(formula = Kyphosis ~ log(Age) + Number, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

orch.glm *Generalized Linear Models for HDFS objects*

Description

Functions for fitting and using generalized linear models on HDFS data.

Usage

```
### Fitting function
orch.glm(formula, data, family = gaussian(), start = NULL,
        control = list(...), contrasts = NULL, xlev = NULL,
        ylev = NULL, yprob = NULL, sparse = FALSE,
        nMappers = -1, nReducers = 1, mapSplit = 0,
        reducer.serial.limit = 8L, task.timeout = -1L, ...)

### Fit control function
orch.glm.control(devlre = 8, maxit = 25, trace = FALSE, linesearch = TRUE, ...)

### Specific methods for orch.glm objects
## S3 method for class 'orch.glm'
predict(object, newdata = NULL, type = c("link", "response"),
        se.fit = FALSE, dispersion = NULL, na.action = na.pass,
        skip.vals = FALSE, mapSplit = 0, nMappers = -1, ...)

## S3 method for class 'orch.glm'
deviance(object)

## S3 method for class 'orch.glm'
extractAIC(fit, scale = 0, k = 2, ...)

## S3 method for class 'orch.glm'
vcov(object, ...)

### Inherited methods for orch.glm objects
#coef(object, ...)
#coefficients(object, ...)
#family(object, ...)
#formula(x, ...)
#logLik(object, ...)
#nobs(object, ...)
```

Arguments

`formula`	A `formula` object representing the model to be fit.
`data`	An HDFS object specifying the data for the model.
`family`	A `family` object specifying the generalized linear model family details. This is the same type of object used for the `glm` function in the **stats** package.
`start`	An optional `numeric` vector specifying the initial coefficient estimates in the linear predictor.

`control`	An optional `list` object containing a list of fit control parameters to be interpreted by the `orch.glm.control` function.
`contrasts`	An optional named `list` to be supplied to the `contrasts.arg` argument of `model.matrix`.
`xlev`	An optional named `list` of `character` vectors specifying the `levels` for each factor variable.
`ylev`	An optional `character` vector to specify the response variable levels in `binomial` generalized linear models.
`yprob`	An optional numeric value between 0 and 1 specifying the overall probability of `y != ylev[1]` in `binomial` generalized linear models.
`sparse`	A logical value indicating whether a sparse matrix solver should be used from the **Matrix** package.
`nMappers`	Hint for number of mappers to be used for the Hadoop jobs. Hadoop defaults are used.
`nReducers`	Hint for number of reducers to be used for the Hadoop jobs. Default is 1.
`mapSplit`	Number of records to supply at once to a mapper. See `map.split` in `mapred.config`
`reducer.serial.limit`	Maximum number of records later phase reducers should process serially
`task.timeout`	Maximum time in seconds a map or reduce task is allowed to run before it gets force killed by Hadoop. Hadoop defaults are used.
`devlre`	A positive number specifying the minimum log relative error of the residual deviance convergence criterion, $-log10(\|dev - dev_{old}\|/\|dev\|) \geq devlre$.
`maxit`	A positive integer specifying the maximum number of Fisher scoring iterations.
`trace`	The control parameter that controls the output produced at each Fisher scoring iteration; a value of `FALSE` or `0` indicating no output, a value of `TRUE` or `1` indicating the printing of the residual deviance for each iteration, or a value of `2` indicating the printing of the residual deviance and runtime breakdown for each iteration.
`object`	An `orch.glm` object.
`newdata`	An HDFS or Hive object.
`skip.vals`	If FALSE, then input value columns are included in the output, else they are not included in the output. Default is FALSE.
`type`	A character string specifying the type of predictions or residuals to produce.
`se.fit`	A logical value indicating whether to return the standard errors for the predictions.
`na.action`	The manner in which `NA` values are handled, either `na.omit` or `na.pass`.
`...`	Additional arguments.

Details

The `orch.glm` function fits generalized linear models using a Fisher scoring iteratively re-weighted least squares algorithm. Instead of the traditional step halving to prevent the selection of less optimal coefficient estimates, a line search is used to select new coefficient estimates at each iteration starting from the current coefficient estimates and moving through the Fisher scoring suggested estimates using the formula $(1 - \alpha) * old + \alpha * suggested$ where α in $[0, 2]$.

Each iteration uses map/reduce operations to calculate the necessary sufficient statistics for generating new coefficient estimates. For more parallelism during reducer computation, a tree of reducers can be used.

To ensure stability, collinear terms are removed from the re-weighted least squares equations prior to solving for new coefficient estimates. After the algorithm has either converged or reached the maximum number of iterations, a final embedded map/reduce operation is used to generate the complete set of model-level statistics. For more parallelism during reducer computation, a tree of reducers can be used.

Value

For `orch.glm`, returns an `orch.glm` object.

For `summary.orch.glm`, returns a `summary.orch.glm` object.

For `predict.orch.glm`, returns an `hdfs.id` object. This corresponds to the output HDFS file.

The output file contains a key column in addition if and only if `newdata` had a key. The value of the key column can be used to associate a record in `newdata` with its corresponding record in the output file.

The format of the output file is as follows - If key column is present it will appear first. This will be followed by the remaining columns in newdata if and only if `skip.vals==FALSE`. The ordering amongst these columns is preserved. Finally, the columns corresponding to the prediction results follow.

See Also

`orch.lm`, `glm`, `family`

Examples

```
# Load libraries for examples
library(ORCHstats)
library(rpart)

# Logistic regression
KYPHOSIS <- hdfs.put(kyphosis)
kyphFit1 <- orch.glm(Kyphosis ~ ., data = KYPHOSIS, family = binomial())
kyphFit2 <- glm(Kyphosis ~ ., data = kyphosis, family = binomial())
summary(kyphFit1)
summary(kyphFit2)

# Predict (note, we leave the result on HDFS)
pred <- predict(kyphFit1, newdata = KYPHOSIS)

# Poisson regression
SOLDER <- hdfs.put(solder)
solFit1 <- orch.glm(skips ~ ., data = SOLDER, family = poisson())
solFit2 <- glm(skips ~ ., data = solder, family = poisson())
summary(solFit1)
summary(solFit2)
```

orch.kmeans	*K-Means Clustering for HDFS objects*

Description

Perform k-means clustering on a data matrix stored as an HDFS file.

Usage

```
orch.kmeans(x, centers, iter.max = 10, nstart = 1, nstart.per.batch=nstart,
            num.mappers=-1, num.reducers=1, reducer.serial.limit=8,
            task.timeout=-1, job.name="ORCH k-means")
```

Arguments

`x`	An `hdfs.id` object containing numeric columns. This input matrix is in dense matrix representation. The key column, if present, is ignored. Only the value columns are considered.
`centers`	either the number of clusters, say k, or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in x is chosen as the initial centres.
`iter.max`	the maximum number of iterations allowed.
`nstart`	if `centers` is a number, the number of random sets that should be chosen
`nstart.per.batch`	The maximum number of random starts to be run in a single batch. See details for more information.
`num.mappers`	Hint for number of mappers to be used for the Hadoop jobs. Hadoop defaults are used.
`num.reducers`	Hint for number of reducers to be used for the Hadoop jobs. Default is 1.
`reducer.serial.limit`	Maximum number of records later phase reducers should process serially
`task.timeout`	Maximum time in seconds a map or reduce task is allowed to run before it gets force killed by Hadoop. Hadoop defaults are used.
`job.name`	Prefix to be used for the Hadoop job names

Details

The data in the HDFS file `x` is clustered by the k-means method, which aims to partition the points into `k` groups such that the sum of squares from points to the assigned cluster centres is minimized. At the minimum, all cluster centres are at the mean of their Voronoi sets (the set of data points which are nearest to the cluster centre).

The algorithm of Lloyd(1957) is implemented using MapReduce. In each iteration tasks work in parallel on disjoint sets of rows of the input matrix. The reducer then puts all these together by performing a weighted mean computation to compute the cluster centers. The cluster centers provided by each mapper are weighted by the respective cluster sizes provided by the mapper and the weighted mean is computed. For more parallelism during reducer computation, a tree of reducers are used.

The assumption is that the matrix of cluster centers will fit in R memory.

k clusters may not always be returned because it is possible that no point will be closest to one or more centres.

Trying several random starts (nstart> 1) is often recommended. Given the cost of data scans, the implementation attempts to batch these random starts in such a way as to minimize the number of scans required. Thus, it is best to use the default value which results in all the random starts being part of a single batch. The only reason to override the default and choose smaller batch sizes is due to considerations on the memory consumption of a batch. In general, this only applies when the number of centers is large.

Value

`orch.kmeans` returns an object of class "`orch.kmeans`" which has a `print` method. It is a list with components:

`centers`	An in memory R matrix of the final cluster centres.
`prev.centers`	An in memory R matrix of cluster centers used at the beginning of the final iteration. It is these centers that are used to determine the cluster allocation of the input points.
`totss`	The total sum of squares.
`withinss`	An in memory R vector of within-cluster sum of squares, one component per cluster.
`tot.withinss`	Total within-cluster sum of squares, i.e., `sum(withinss)`.
`betweenss`	The between-cluster sum of squares, i.e. `totss-tot.withinss`.
`size`	An in memory R vector of the number of points in each cluster, one component per cluster.
`iter`	Number of iterations performed.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

See Also

`kmeans`

Examples

```
require(graphics)

# a 2-dimensional example
x <- data.frame(rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
           matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2)))
colnames(x) <- c("x", "y")
xdir <- hdfs.put(x)

kcl <- kmeans(x, iter.max=2, centers=x[c(1,51),], algorithm="Lloyd")
ocl <- orch.kmeans(xdir, iter.max=2, centers=x[c(1,51),])

stopifnot(all.equal(kcl$centers, ocl$centers),
          all.equal(kcl$withinss, ocl$withinss, check.attributes=FALSE),
          all.equal(kcl$tot.withinss, ocl$tot.withinss),
          all.equal(kcl$totss, ocl$totss),
          all.equal(kcl$betweenss, ocl$betweenss),
          all.equal(kcl$size, ocl$size)
         )

plot(x, col=kcl$cluster)
points(ocl$centers, col = 1:2, pch = 8, cex = 2)

# Prediction
pred <- orch.predict(ocl, xdir)
head(hdfs.get(pred))
```

`orch.lm2` *ORAAH Fitting Linear Models (LM)*

Description

High performance linear regression, based on parallel distributed normal equations and Cholesky factorization.

Usage

```
orch.lm2(formula, data, verbose = TRUE,
  maxBlockRows = 20000L)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.
`maxBlockRows`	Maximum number of rows in a partition. Smaller number or rows will create smaller partitions and more paritions. More data partitions ensures higher parallelization degree across Spark cluster but at the same time small paritions will cause higher communication and resource management overhead.

Details

ORAAH LM is used to carry out linear regression `y = X * beta + e`, where `y` is the response, `X` is the design matrix, `beta` is a vector of regression coefficients, and `e` is the error.

The implementation is based on parallel distributed normal equations `X^T * X * beta = X^T y`, and parallel supernodal Cholesky factorization.

ORAAH LM can efficiently handle both numeric and high cardinality factor variables. ORAAH LM automatically switches to the out-of-core mode, if the input data does not fit into the distributed memory.

Value

LM2 fit object, `orch.lm2`.

ORAAH Formula

Everywhere below `A` can be either an ID (column name), it can also denote any generated column, for instance `sin(A / 10)`, or any subset of columns for instance `(A1 + A2 + A3)`.

Numerical engines, such as linear regression, cannot consume raw data; there must be a way to specify response and explanatory variables, nonlinear transformations, and interactions. Formula is such an engine, a recipe which specifies which columns (terms) to include to the model, and how to transform them if desired.

- `.` dot-character is a shortcut for all variables (all data columns), except the response.
- `+A` plus operator means to include this variable into the model. Plus operator here is used in set-theoretic sense. There is no arithmetic summation here of any kind. We add a term (column) to an ordered set of statistical terms (model).
- `-A` minus operator means to remove this variable from the model. Example `- (A + B)` removes both A and B variables from the model. Example `. - (A + B)` includes all variables, except A, B, and the response.
- `A : B` include the interaction between A and B variables.
- `A * B` include these variables and the interactions between them. This is equivalent to `A + B + A : B`. Example `A * ( . - B ) * Z`
- `(A1 + A2 + ... + Ak)^n` include these variables and all interactions up to n-way. For instance, `(A + B)^2` is equivalent to `A + B + A : B`. The exponentiation (the power operator) can lead to much more compact model specification. For instance `(. - A)^3` will include all variables, excluding the A and the response, and will include the corresponding interactions. To reiterate, A can be either an ID (variable name) or any complex term. For instance, `(log(A) + B : Z)^2` is equivalent to `log(A) + B : Z + log(A) : B : Z`
- `I()` Identity function. Its argument will be treated in arithmetic sense (as versus set-theoretic sense). For instance: `I(log(A) + B)` will include a new column, whose elements are `log(A[k]) + B[k]`. Here, the plus operator (and all other operators) will be treated in their traditional arithmetic sense.
- 24 arithmetic functions. The argument will be treated in arithmetic sense. Example `log(A / 10 + B * Z)`.

abs	acos	asin
atan	cbrt	ceil
cos	cosh	exp
expm1	floor	log
log10	log1p	rint
round	signum	sin
sinh	sqrt	tan
tanh	toDegrees	toRadians

- Relational operators, currently supported for numerical terms only. Example Y ~ X + (A > B).

A >= B	A <= B
A > B	A < B
A == B	A != B

```
A && B   A || B
A & B    A | B
```

- `+1` Add the intercept.
- `-0` Add the intercept (equivalent to +1).
- `-1` Delete the intercept.
- `+0` Delete the intercept (equivalent to -1).

It is very important to keep in mind, that all factor variables (including factor-factor and factor-numeric interactions), are unrolled following one-hot scheme, meaning internally they will be substituted by `k-1` dummy variables.

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.formula predict.orch.lm2 oracle.model.matrix

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.lm2(formula = Age ~ log(Number) + Kyphosis, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

---

orch.lmf *Fit a Low Rank Matrix Factorization Model*

Description

This function is used to fit a Low Rank Matrix Factorization model

Usage

```
orch.lmf(input, method =c("jellyfish", "mahout-als"), dfs.output = NULL, ...)
```

Arguments

`input`	A CSV ratings file containing entries of the form (user, item, rating). This can be one of the following 1. the HDFS directory containing the input file 2. R data.frame object 3. ore.frame object 4. name of a file in the local file system
`method`	The method to be used. Default is `jellyfish`
`dfs.output`	The output HDFS directory where the model should be created. If not specified, this method will internally create a directory and use that as the output directory.
`...`	Optional method specific arguments The arguments specific to the "jellyfish" method are:
`latin`	Latin Square dimension for Map Reduce parallelism. This is an optional argument. The default value is computed based on the memory per mapper.
`rank`	The rank of the latent factor matrices. This is an optional argument with default value of 50.
`iterations`	Number of iterations of Incremental Gradient Descent (IGD) to be performed. This is an optional argument with default value 10.
`step`	Learning Rate / Step size to be used in IGD. This is an optional argument with default value 0.05.
`decay`	Decay parameter for step size to be used in IGD. This is an optional argument with default value 0.8.
`regularizer`	Regularization parameter to be used in IGD. This is an optional argument with default value 2.3.
`init`	Values for initalization of factors will be uniformly chosen from (0 .. init). This is an optional argument with default value 1.
`seed`	Seed value for random number generation. This is an optional argument.
`mapmem`	Amount of memory available per mapper in MB. This is an optional argument with default value 200. The arguments specific to the "mahout-als" method are:
`rank`	The rank of the latent factor matrices. This is an optional argument with default value of 50.
`iterations`	Number of iterations to be performed. This is an optional argument with default value 10.
`regularizer`	Regularization parameter to be used in ALS. This is an optional argument with default value 0.065.

Details

The `jellyfish` algorithm implements a projected incremental gradient descent method. Massive parallelization of the gradient computations are achieved by partitioning the matrix into chunks.

Value

Returns the fitted model, an object of an `orch.lmf` subclass.

In case of "jellyfish", this is a list with the following components

`results`	A data frame with the error metrics (RMSE, SSE) after each iteration of IGD.
`nrows`	Number of rows in training input matrix.
`ncols`	Number of columns in training input matrix.
`nratings`	Number of input entries.
`inputDir`	The HDFS directory containing the input.
`modelDir`	The HDFS directory containing the model.

In case of "mahout-als", this is a list with the following components

`inputDir`	The HDFS directory containing the input.
`modelDir`	The HDFS directory containing the model.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

Examples

```
## Setup the input (user, item, rating) entries
u <- sample(1:100, 300, replace=TRUE)
i <- sample(1:10, 300, replace=TRUE)
ui <- unique(cbind(u,i))
r <- sample(1:5, nrow(ui), replace=TRUE)
input <- cbind(ui,r)

# Fit an "orch.lmf.jellyfish" model
fit1 <- orch.lmf(input, latin=2, iterations=5, rank=3)
print(fit1)

# For "mahout-als", set up an input file
inputFile <- tempfile(tmpdir='/tmp')
write.table(input, file=inputFile, sep=",", col.names=FALSE, row.names=FALSE)

# Fit using "mahout-als"
fit2 <- orch.lmf(inputFile, method="mahout-als", rank=3)
print(fit2)
```

`orch.lm` *Linear Regression for HDFS Objects*

Description

Functions for fitting and using linear regression models on HDFS data.

Usage

```
### Fitting functions
orch.lm(formula, dfs.dat, nMappers = -1L, nReducers = 1L, mapSplit = 0,
        contrasts = NULL, xlev = NULL, sparse = FALSE,
        reducer.serial.limit = 8L, task.timeout = -1L, ...)

### Specific methods for ore.lm objects
## S3 method for class 'orch.lm'
summary(object, correlation = FALSE, symbolic.cor = FALSE, ...)

## S3 method for class 'orch.lm'
vcov(object, ...)

## S3 method for class 'orch.lm'
anova(object, ...)

## S3 method for class 'orch.lm'
deviance(object)

## S3 method for class 'orch.lm'
nobs(object)

## S3 method for class 'orch.lm'
predict(object, newdata, se.fit = FALSE, scale = NULL, df = Inf,
        interval = c("none", "confidence", "prediction"),
        level = 0.95, na.action = na.pass, pred.var = NULL,
        skip.vals =FALSE, mapSplit = 0, nMappers = -1, ...)


### Inherited methods for ore.lm objects
#coef(object, ...)
#coefficients(object, ...)
#confint(object, parm, level = 0.95, ...)
#formula(x, ...)
```

Arguments

`formula`	A `formula` object representing the model (`orch.lm`) or initial model (`ore.stepwise`) to be fit.
`dfs.dat`	An HDFS object specifying the data for the model.
`contrasts`	An optional named `list` to be supplied to the `contrasts.arg` argument of `model.matrix`.

`xlev`	An optional named `list` of `character` vectors specifying the `levels` for each factor variable.
`sparse`	A logical value indicating whether a sparse matrix solver should be used from the **Matrix** package.
`nMappers`	Hint for number of mappers to be used for the Hadoop jobs. Hadoop defaults are used.
`nReducers`	Hint for number of reducers to be used for the Hadoop jobs. Default is 1.
`reducer.serial.limit`	Maximum number of records later phase reducers should process serially
`task.timeout`	Maximum time in seconds a map or reduce task is allowed to run before it gets force killed by Hadoop. Hadoop defaults are used.
`object, model, newdata`	`orch.lm` object.
`correlation, symbolic.cor`	Argument not implemented.
`REML`	Argument not implemented.
`se.fit`	A logical value indicating whether to return the standard errors for the predictions.
`scale`	The scale parameter for standard error of the predictions.
`df`	The degrees of freedom for the predictions when argument `scale` is not `NULL`.
`interval`	The type of interval to return, either `"none"`, `"confidence"`, or `"prediction"`.
`level`	The level for argument `interval`.
`na.action`	The manner in which `NA` values are handled, either `na.omit` or `na.pass`.
`pred.var`	When argument `interval` is `"prediction"`, the variance for a single observation.
`...`	Additional arguments.

Details

The `orch.lm` function performs least squares regression on HDFS data.

A model fit is generated using map/reduce operations where the map operation creates QR decompositions of the `model.matrix`, or `sparse.model.matrix` if argument `sparse = TRUE`, and the reduce operation block updates those QR decompositions. For more parallelism during reducer computation, a tree of reducers can be used.

Once the coefficients for the model have been estimated another pass of the data is made to estimate the model-level statistics.

If there are collinear terms in the model, `orch.lm` will not estimate the coefficient values for a collinear set of terms.

Value

For `orch.lm`, returns an `orch.lm` object.

For `summary.orch.lm`, returns a `summary.orch.lm` object.

For `predict.orch.lm`, returns an `hdfs.id` object. This corresponds to the output HDFS file.

The output file contains a key column in addition if and only if `newdata` had a key. The value of the key column can be used to associate a record in `newdata` with its corresponding record in the output file.

The format of the output file is as follows - If key column is present it will appear first. This will be followed by the remaining columns in newdata if and only if `skip.vals==FALSE`. The ordering amongst these columns is preserved. Finally, the columns corresponding to the prediction results follow.

See Also

`orch.glm`, `lm`,

Examples

```
# Prepare the model and the data
# Note, the number of mappers is defined by the ORCH platform.
dat <- hdfs.put(iris)
frm <- Petal.Width ~ Sepal.Length + (Sepal.Width + Petal.Length)^2
fit <- orch.lm(frm, dat)

# Print summary
summary(fit)

# Predict (note, we leave the result on HDFS)
pred <- predict(fit, newdata = dat)
```

`orch.load.model` *Load MLlib Models from HDFS.*

Description

This function loads a model created using Spark MLlib in ORAAH from hdfs for scoring/prediction. It also enables loading of models created by other users if access to the hdfs path where models are saved is provided.

Usage

```
orch.load.model(dfs.name)
```

Arguments

`dfs.name`	A string specifying absolute or relative HDFS path of the saved model.

Value

MLlib fit object of type among `orch.ml.logistic`, `orch.ml.linear`, `orch.ml.lasso`, `orch.ml.ridge`, `orch.ml.svm`, `orch.ml.gmm`, `orch.ml.kmeans`, `orch.ml.dt` or `orch.ml.random.forest` present at the location provided by `dfs.name`.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.save.model

Examples

```
library(rpart)
data   <- hdfs.put(kyphosis)
model  <- orch.ml.ridge(formula = Number ~ Age, data = data)
orch.save.model(model, "ridgeKypSave", overwrite=TRUE)
model.load <- orch.load.model("ridgeKypSave")
pred   <- predict(model.load, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite=TRUE)
```

`orch.ml.dt` *MLlib Decision Tree.*

Description

MLlib Decision Tree.

Usage

```
orch.ml.dt(formula, data, type = NULL, nClasses = NULL,
  impurity = NULL, maxDepth = 4L, maxBins = 100L,
  verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`type`	Can be "classification" or "regression". The default value is `NULL`, in which case it will be determined automatically based on the input data set and formula.
`nClasses`	Number of classes. The default value is `NULL`, in which case it will be determined automatically based on the input data set and formula.
`impurity`	Criterion used for information gain calculation. Values "gini" and "entropy" are supported for classification, and "variance" for regression. The default value is `NULL`, in which case it will be determined automatically based on the input data set and formula.

`maxDepth`	Maximum depth of the decision tree, default value `4`.
`maxBins`	Maximum number of bins used for splitting features, default value `100`.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

Decision trees are recursive algorithms consisting of binary nodes. Each node is characterized by a decision boundary over one of the predictor variables.

Decision boundaries.

Each binary node identifies a decision w.r.t. one predictor variable x_i by splitting domain region $\mathcal{R}$ of that predictor into two disjoint domain regions $\mathcal{R}_1$, $\mathcal{R}_2$: $\mathcal{R}_1 \cup \mathcal{R}_2 = \mathcal{R}$, $\mathcal{R}_1 \cap \mathcal{R}_2 = \emptyset$. If predictor is continuous, then the decision is sought as a split boundary θ between the enclosing regions $\mathcal{R}_1$ and $\mathcal{R}_2$ so that both new regions are continuous. If the predictor is categorical, then the decision boundary θ is defined by two disjoint category subsets $\mathcal{R}_1$ and $\mathcal{R}_2$ directly: $\theta = (\mathcal{R}_1, \mathcal{R}_2)$.

Each decision node therefore is characterized by a heuristic referred to as the *information gain*:

$$IG(\mathcal{R}, \mathcal{R}_1, \mathcal{R}_2) = I(\mathcal{R}) - \frac{N_1}{N} I(\mathcal{R}_1) - \frac{N_2}{N}(\mathcal{R}_2)$$

.

Here, N, N_1, N_2 are cardinalities of subsets of the training set such that $x_i \in \mathcal{R}$, $x_i \in \mathcal{R}_1$, and $x_i \in \mathcal{R}_2$, respectively. Also, the quantities $I(\cdot)$ are measures of target variable impurity in the specified predictor regions.

At prediction time, if the input's predictor $x_i \in \mathcal{R}_1$, then prediction algorithm recursively walks down the left subtree, otherwise the algorithm walks down the right subtree. The domain range of x_i is assigned $\mathcal{R} \leftarrow \mathcal{R}_1$ for the left subtree, and $\mathcal{R} \leftarrow \mathcal{R}_2$ for the right subtree.

The tree leaves have a prediction quantity $\hat{y}$ associated with them. The prediction algorithm stops when the leaf is reached, at which point prediction result is taken as the prediction quantity of the leaf reached.

Impurity heuristics.

There are several choices of the impurity heuristic to be used during tree fitting.

For categorical targets y, i.e., a classification problem, the choices are:

- Gini impurity: $I^{\text{Gini}}(\mathcal{R}) = \sum_{i=1}^{C} f_i (1 - f_i)$;
- Entropy: $I^{\text{Entropy}}(\mathcal{R}) = -\sum_{i=1}^{C} f_i \log f_i$.

Here, C is the cardinality of target category set, and f_i is frequency of the i-th category in the training subset subject to $x_i \in \mathcal{R}$.

For a continuous target y, i.e., a regression problem, impurity choice is the variance of the target variable:

$$I^{\text{Variance}}(\mathcal{R}) = \text{VAR}(y), \text{ subject to } x_i \in \mathcal{R}.$$

Fitting.

The fitting of decision tree is therefore driven by assigning model parameters to each node: a choice of predictor variable x_i to use, and the split boundary θ. For exact strategies for finding (i, θ), please refer to the MLlib manual.

Value

Decision Tree fit object, `orch.ml.dt`.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

`orch.formula` `predict.orch.ml.dt` `oracle.model.matrix`

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.dt(formula = Kyphosis ~ Number + Age, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

`orch.ml.gmm` *MLlib Gaussian Mixture Model*

Description

MLlib implementation of Gaussian Mixture Model fitting.

Usage

```
orch.ml.gmm(formula, data, nGaussians = 2L,
  maxIterations = 20L, convergenceTol = 1e-04,
  seed = as.integer(1e+08 * runif(1)), verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`nGaussians`	Number of gaussian centers.

`maxIterations`	Maximum number of iterations. By default, 20 iterations is used as the stopping criterion.
`convergenceTol`	Convergence tolerance. By default, `1E-4` is used as the stopping criterion.
`seed`	Pseudo-random number generator seed, for cluster initialization.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

Gaussian Mixture Models (GMM) are often used for data clustering.

Gaussian Mixture Models express probability density of any particular input point as a weighted mixture of individual multivariate normal distributions:

$$p\left(\mathbf{x}_i|\boldsymbol{\theta}\right) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right).$$

K denotes the number of the normally distributed components in the summation.

Fitting the model means finding parameters of this distribution $\boldsymbol{\theta} = \left\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k : k = 1, 2, ...K\right\}$. Probabilistic approach seeks to maximize the posterior (MAP) of the parameters $\boldsymbol{\theta}$ given observed input $\mathbf{X}$, K, and the hyperparameters of prior distributions of $\boldsymbol{\theta}$.

Once the GMM model parameters are estimated, either the training or some new input $\mathbf{X}$ can be then assigned to clusters $k : k = 1, 2, \ldots K$. These assignments can be expressed as responsibility quantities r_{ik} representing probabilities of the point $\mathbf{x}_i$ being generated by the k-th normal component of the distribution:

$$r_{ik} = p\left(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}\right).$$

The process of assigning quantities r_{ik} to the input points $\mathbf{x}_i$ is called *soft clustering*.

The process of *hard clustering*, on the other hand, associates each input point $\mathbf{x}_i$ with exactly one normal component in the distribution. Hard clustering is usually derived based on responsibility estimates of the soft clustering, for example:

$$z_i^* = \arg\max_k r_{ik}.$$

MLlib itself is capable of finding both soft and hard cluster assignments.

ORAAH ‘predict‘ implementation performs hard cluster assignment.

Value

GMM fit object, `orch.ml.gmm`.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

`orch.formula` `predict.orch.ml.gmm` `oracle.model.matrix`

Examples

```
library(rpart)
data   <- hdfs.put(kyphosis)
model  <- orch.ml.gmm(formula = ~ Number + Age, data = data)
pred   <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

`orch.ml.kmeans` *MLlib K-means.*

Description

MLlib K-means.

Usage

```
orch.ml.kmeans(formula, data, nClusters = 2L,
  maxIterations = 20L, nParallelRuns = 1L,
  initializationMode = "k-means||",
  seed = as.integer(1e+08 * runif(1)), verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`nClusters`	Number of clusters. By default `2` clusters will be formed.
`maxIterations`	Maximum number of iterations. By default, `20` iterations is used as the stopping criterion.
`nParallelRuns`	Number of parallel runs, defaults to `1`. The best model is returned.
`initializationMode`	Initialization model, either "random" or "k-means||". Default is "k-means||".
`seed`	Seed value for cluster initialization. If not specified a pseudo-random generated number will be used.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

K-means is a simple unsupervised learning technique performing data partitioning into k clusters. Each cluster is assigned a centroid point, and every point in the dataset is assigned to the closest centroid, thus producing a Voronoi tessellation. The training produces a model c onsisting of k centroid points.

Let the training input be $\mathcal{D} = \{\mathbf{x}_i : i = 1, 2, \dots m\}$. Let cluster centroid points be $\{\mu_j : j = 1, 2, \dots k\}$, and the paritioning of the input points into clusters based on nearest centroid criteria at any moment $\mathcal{S} = \{\mathcal{S}_j : j = 1, 2, \dots k\}$. The fitting seeks a solution (centroid model $\boldsymbol{\mu}$) as:

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} \sum_{j=1}^{k} \sum_{i}^{\mathbf{x}_i \in \mathcal{S}_j} \|\mathbf{x}_i - \mu_j\|_2 \,.$$

The exact solution is NP-hard and is usually intractable; various modifications seek a local minimum of the objective instead. MLlib employs a variety of the algorithm called "k-means ||". This algorithm replaces classic Forgy initialization with a probabilistic approximation of density proxies, so that Lloyd iterations have a better chance of achieving a better local minimum solution due to a better initial guess.

Value

K-means fit object, `orch.ml.kmeans`.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.formula predict.orch.ml.kmeans oracle.model.matrix

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.kmeans(formula = ~ Number + Age, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

orch.ml.lasso — *MLlib Lasso (Least Absolute Shrinkage and Selection Operator) with Stochastic Gradient Descent.*

Description

Linear regression family of methods seeks to minimize a loss function employing 1-norm penalty over the fitted parameters β.

Usage

```
orch.ml.lasso(formula, data, convergenceTol = 0.001,
  miniBatchFraction = 1, featureScaling = FALSE,
  maxIterations = 100L, regParam = 0.01, stepSize = 1,
  validateData = TRUE, verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`convergenceTol`	Convergence tolerance. By default, `1E-3` is used as the stopping criterion.
`miniBatchFraction`	Fraction of the data to compute a gradient. By default, entire data is used for gradient evaluation.
`featureScaling`	Whether to carry out feature scaling before fitting the model. Default behavior being `FALSE`.
`maxIterations`	Maximum number of iterations. By default, `100` iterations is used as the stopping criterion.
`regParam`	Regularization parameter, default value `0.01`.
`stepSize`	Step size (learning rate), default value is `1.0`.
`validateData`	Whether to validate the input data. Default behavior being `TRUE`.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

Generalized linear models family of methods seeks to minimize a loss function employing 1-norm penalty over the fitted parameters β.

Suppose we have the training dataset of predictors $\mathcal{D} = \{\mathbf{x}_i : i = 1, 2, \ldots N\}$ and their corresponding target variables $\{y_i : i = 1, 2, \ldots N\}$. Linear methods seek to minimize the loss function:

$$L(\boldsymbol{\beta}) = \frac{1}{2N} \sum_i \left(\boldsymbol{\beta}^\top \mathbf{x}_i - y_i\right)^2 + \lambda R(\boldsymbol{\beta}),$$

where λ is the regularization rate (parameter `regParam`), and $R(\boldsymbol{\beta})$ is the regularization penalty function.

Value

Lasso fit object, `orch.ml.lasso`.

Fitting

This MLlib version seeks solution using SGD (Stochastic Gradient Descent) over several training epochs. The maximum amount of epochs is controlled by the `maxIterations` parameter of the training procedure. During each epoch j, a fraction of the input is sampled into a minibatch $\mathcal{S}_j$, and then a partial loss gradient is computed and solution is updated according to:

$$\boldsymbol{\beta}^{(j+1)} = \boldsymbol{\beta}^{(j)} - \alpha^{(j)} \nabla_{\boldsymbol{\beta}} L\left(\boldsymbol{\beta}^{(j)}\right),$$

where $\alpha^{(j)}$ is the SGD learning rate in j-th epoch. In MLlib, the epoch learning rate $\alpha^{(j)}$ is subject to annealing schedule:

$$\alpha^{(j)} = \alpha\sqrt{j},$$

where α is the initial learning rate as supplied by the `stepSize` parameter.

Fitting

Lasso regression uses 1-norm regularization:

$$R(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1.$$

The Lasso update is:

$$\boldsymbol{\beta}^{(j+1)} = \operatorname{prox}_{\lambda\alpha^{(j)}\|\cdot\|_1}\left(\boldsymbol{\beta}^{(j)} + \frac{\alpha^{(j)}}{|\mathcal{S}_j|} \sum_i^{\boldsymbol{x}_i \in \mathcal{S}_j} r_i^{(j)} \boldsymbol{x}_i\right),$$

where $\operatorname{prox}_{\lambda\alpha^{(j)}\|\cdot\|_1}(\cdot)$ is element-wise application of the proximal operator of the function $\lambda\alpha^{(j)}\|\cdot\|_1$, and $r_i^{(j)} = y_i - \boldsymbol{\beta}^{(j)\top}\mathbf{x}_i$ is the previous epoch's residual at point i.

The proximal operator for 1-norm, and any real $\gamma > 0$ as:

$$\operatorname{prox}_{\gamma\|\cdot\|_1}(f) = \begin{cases} f - \gamma, & f > \gamma; \\ 0, & -\gamma \le f \le \gamma; \\ f + \gamma, & f < -\gamma. \end{cases}$$

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

https://en.wikipedia.org/wiki/Lasso_(statistics)

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.formula predict.orch.ml.lasso oracle.model.matrix

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.lasso(formula = Kyphosis ~ Number + Age, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

---

orch.ml.linear *MLlib Linear Regression with Stochastic Gradient Descent.*

Description

MLlib Linear Regression with Stochastic Gradient Descent.

Usage

```
  orch.ml.linear(formula, data, convergenceTol = 1e-04,
    miniBatchFraction = 1, featureScaling = FALSE,
    maxIterations = 100L, stepSize = 1,
    validateData = TRUE, verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`convergenceTol`	Convergence tolerance. By default, `1E-4` is used as the stopping criterion.
`miniBatchFraction`	Fraction of the data to compute a gradient. By default, entire data is used for gradient evaluation.
`featureScaling`	Whether to carry out feature scaling before fitting the model. Default behavior being `FALSE`.

`maxIterations`	Maximum number of iterations. By default, `100` iterations is used as the stopping criterion.
`stepSize`	Step size (learning rate), default value is `1.0`.
`validateData`	Whether to validate the input data. Default behavior being `TRUE`.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

Generalized linear models family of methods seeks to minimize a loss function employing 1-norm penalty over the fitted parameters $\boldsymbol{\beta}$.

Suppose we have the training dataset of predictors $\mathcal{D} = \{\mathbf{x}_i : i = 1, 2, \ldots N\}$ and their corresponding target variables $\{y_i : i = 1, 2, \ldots N\}$. Linear methods seek to minimize the loss function:

$$L(\boldsymbol{\beta}) = \frac{1}{2N} \sum_i \left(\boldsymbol{\beta}^\top \mathbf{x}_i - y_i\right)^2 + \lambda R(\boldsymbol{\beta}),$$

where λ is the regularization rate (parameter `regParam`), and $R(\boldsymbol{\beta})$ is the regularization penalty function.

Value

Linear regression fit object, `orch.ml.linear`.

Fitting

This MLlib version seeks solution using SGD (Stochastic Gradient Descent) over several training epochs. The maximum amount of epochs is controlled by the `maxIterations` parameter of the training procedure. During each epoch j, a fraction of the input is sampled into a minibatch $\mathcal{S}_j$, and then a partial loss gradient is computed and solution is updated according to:

$$\boldsymbol{\beta}^{(j+1)} = \boldsymbol{\beta}^{(j)} - \alpha^{(j)} \nabla_{\boldsymbol{\beta}} L\left(\boldsymbol{\beta}^{(j)}\right),$$

where $\alpha^{(j)}$ is the SGD learning rate in j-th epoch. In MLlib, the epoch learning rate $\alpha^{(j)}$ is subject to annealing schedule:

$$\alpha^{(j)} = \alpha\sqrt{j},$$

where α is the initial learning rate as supplied by the `stepSize` parameter.

Fitting

The OLS update in MLlib is:

$$\boldsymbol{\beta}^{(j+1)} = \boldsymbol{\beta}^{(j)} + \frac{\alpha^{(j)}}{|\mathcal{S}_j|} \sum_i^{x_i \in \mathcal{S}_j} r_i^{(j)} \mathbf{x}_i,$$

where $r_i^{(j)} = y_i - \boldsymbol{\beta}^{(j)\top} \mathbf{x}_i$ is the previous epoch's residual at point i.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.formula predict.orch.ml.linear oracle.model.matrix

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.linear(formula = Number ~ Age, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

---

orch.ml.logistic *MLlib Logistic Regression with L-BFGS.*

Description

Logistic regression multinomial logistic regression.

Usage

```
orch.ml.logistic(formula, data, nClasses = 2L,
  convergenceTol = 1e-04, featureScaling = TRUE,
  maxIterations = 100L, regParam = 0.01,
  validateData = TRUE, verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`nClasses`	Number of classes, use `2` for logistic regression. By default, `2` classes is used.
`convergenceTol`	Convergence tolerance. By default, `1E-4` is used as the stopping criterion.
`featureScaling`	Whether to carry out feature scaling before fitting the model. Default behavior being `TRUE`.
`maxIterations`	Maximum number of iterations. By default, `100` iterations is used as the stopping criterion.

`regParam`	Regularization parameter, default value `0.01`.
`validateData`	Whether to validate the input data. Default behavior being `TRUE`.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

Suppose we have a training dataset consisting of predictors $\mathcal{D} = \{\mathbf{x}_i : i = 1, 2, \ldots N\}$, and their corresponding target variables $\{y_i : i = 1, 2, \ldots N\}$.

Logistic regression seeks to minimize a loss function of the form:

$$L(\boldsymbol{\beta}) = \frac{1}{N}\sum_{i=1}^{N} \log\left(1 + \exp\left(-y_i \boldsymbol{\beta}^\top \mathbf{x}_i\right)\right) + \lambda R(\boldsymbol{\beta}),$$

where λ is the regularization rate (parameter `regParam`), and $R(\boldsymbol{\beta})$ is the regularization penalty function.

This method uses L2 normalization:

$$R(\boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{\beta}\|_2^2.$$

The prediction score estimator is evaluated by applying the logistic function over linear combination of predictors:

$$\hat{y}(\mathbf{x}) = \frac{1}{1 + \exp\left(-\boldsymbol{\beta}^\top \mathbf{x}\right)}.$$

For binomial targets the outcome is predicted as positive if $\hat{y}(\mathbf{x}) > 0.5$, and as negative otherwise. The interpretation of the score estimator is probabilistic. When regularization is used ($\lambda > 0$), the score estimates maximum aposteriori (MAP) of the positive outcome. Otherwise, the score the probability of positive outcome per maximum likelihood estimate (MLE).

In MLlib the logistic regression procedure also is extended to support multi-class predictions. In this case, if K is the number of classes (parameter `nClasses`), then $K - 1$ logistic regression models are trained. At prediction time, the class $i + 1$ is selected if the i-th model has highest score that is greater than 0.5; otherwise, class 1 is selected.

ORAAH adds formula functionality in addition to MLlib functionality. Within ORCH formula parameter, the target should be a factor in order to trigger multiclass target transformation for MLlib. If the target is continuous, it should be following the MLlib conventions of specifying multiclass targets as one of 0, 1, .. (K-1), where K is the number of classes.

Value

Logistic regression fit object, `orch.ml.logistic`.

Fitting

This method maps to MLlib implementation that uses the full batch LBFGS optimizer to converge on the solution.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

`orch.formula` `predict.orch.ml.logistic` `oracle.model.matrix`

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.logistic(formula = Kyphosis ~ Number, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

`orch.ml.pca` *MLlib Principal components analysis.*

Description

MLlib Principal components analysis.

Usage

```
orch.ml.pca(formula, data, nPrincipalComponents,
  verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`nPrincipalComponents`	Number of top principal components to compute.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

Principal component analysis (PCA) is widely used in dimensionality reduction . It is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated columns into a set of linearly uncorrelated columns called principal components. The number of principal components is less than or equal to the number of original columns. This transformation is defined in such a way that the first principal component has the largest possible variance (i.e., accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible.

Value

PCA fit object, `orch.ml.pca`.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

`orch.formula` `oracle.model.matrix`

Examples

```
library(rpart)
data <- hdfs.put(kyphosis)
projected <- orch.ml.pca(formula = ~ Number + Age, data = data, nPrincipalComponents = 1L
hdfs.write(projected$pca, outPath = "pca_row_matrix", overwrite = TRUE)
```

`orch.ml.random.forest` *MLlib Random Forest.*

Description

MLlib Random Forest.

Usage

```
orch.ml.random.forest(formula, data, nTrees = 1L,
  type = NULL, nClasses = NULL, impurity = NULL,
  featureSubsetStrategy = "auto", maxDepth = 4L,
  maxBins = 100L,
  seed = as.integer(runif(n = 1L, min = 0, max = 1e+06)),
  verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.

`type`	Can be set to "classification" or "regression". Default value is `NULL`, in which case it will be determined automatically based on the input data set and formula.
`nClasses`	Number of classes. Default value is `NULL`, in which case it will be determined automatically based on the input data set and formula.
`impurity`	Criterion used for information gain calculation. Values "gini" and "entropy" are supported for classification, and 'variance' for regression. Default value is `NULL`, in which case it will be determined automatically based on the input data set and formula.
`featureSubsetStrategy`	Feature subset strategy. Number of features to consider for splits at each node. Supported values are "auto", "all", "sqrt", "log2", "onethird". If "auto" is set, this parameter is set based on `nTrees` as follows: • If `nTrees == 1`, set to "all"; • if `nTrees > 1` (forest) set to "sqrt" for classification and to "onethird" for regression.
`maxDepth`	Maximum depth of the decision trees, default value is `4`.
`maxBins`	Maximum number of bins used for splitting features, default value is `100`.
`seed`	Seed value for cluster initialization. If not specified a pseudo-random generated number will be used.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

MLlib Random forest trains several decision trees at the same time. Input for every decision tree learning is bootstrapped. Bootstrapping means sampling individual tree's input from the total input without replacement.

Aside from the sampling of the input, another way the training randomizes the process is random selection of the attribute subsets to consider for individual tree node boundaries.

As the result, the model produces an ensemble of experts (each being a decision tree) that vary in goodness of fit in various areas of the input domain.

The prediction is produced using expert majority vote for classification targets, and averaging of expert scores for regression problems.

Value

Random Forest fit object, `orch.ml.random.forest`.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

`orch.formula` `predict.orch.ml.random.forest` `oracle.model.matrix`

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.random.forest(formula = Kyphosis ~ Number + Age, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

`orch.ml.ridge` *MLlib Ridge Regression with Stochastic Gradient Descent.*

Description

MLlib Ridge Regression with Stochastic Gradient Descent.

Usage

```
  orch.ml.ridge(formula, data, convergenceTol = 0.001,
    miniBatchFraction = 1, featureScaling = TRUE,
    maxIterations = 100L, regParam = 0.01, stepSize = 1,
    validateData = TRUE, verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`convergenceTol`	Convergence tolerance. By default, `1E-3` is used as the stopping criterion.
`miniBatchFraction`	Fraction of the data to compute a gradient. By default, entire data is used for gradient evaluation.
`featureScaling`	Whether to carry out feature scaling before fitting the model. Default behavior being `TRUE`.
`maxIterations`	Maximum number of iterations. By default, `100` iterations is used as the stopping criterion.
`regParam`	Regularization parameter, default value `0.01`.
`stepSize`	Step size (learning rate), default value is `1.0`.
`validateData`	Whether to validate the input data. Default behavior being `TRUE`.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

Generalized linear models family of methods seeks to minimize a loss function employing 1-norm penalty over the fitted parameters $\boldsymbol{\beta}$.

Suppose we have the training dataset of predictors $\mathcal{D} = \{\mathbf{x}_i : i = 1, 2, \ldots N\}$ and their corresponding target variables $\{y_i : i = 1, 2, \ldots N\}$. Linear methods seek to minimize the loss function:

$$L(\boldsymbol{\beta}) = \frac{1}{2N} \sum_i \left(\boldsymbol{\beta}^\top \mathbf{x}_i - y_i\right)^2 + \lambda R(\boldsymbol{\beta}),$$

where λ is the regularization rate (parameter `regParam`), and $R(\boldsymbol{\beta})$ is the regularization penalty function.

Value

Ridge regression fit object, `orch.ml.ridge`.

Fitting

This MLlib version seeks solution using SGD (Stochastic Gradient Descent) over several training epochs. The maximum amount of epochs is controlled by the `maxIterations` parameter of the training procedure. During each epoch j, a fraction of the input is sampled into a minibatch $\mathcal{S}_j$, and then a partial loss gradient is computed and solution is updated according to:

$$\boldsymbol{\beta}^{(j+1)} = \boldsymbol{\beta}^{(j)} - \alpha^{(j)} \nabla_{\boldsymbol{\beta}} L\left(\boldsymbol{\beta}^{(j)}\right),$$

where $\alpha^{(j)}$ is the SGD learning rate in j-th epoch. In MLlib, the epoch learning rate $\alpha^{(j)}$ is subject to annealing schedule:

$$\alpha^{(j)} = \alpha\sqrt{j},$$

where α is the initial learning rate as supplied by the `stepSize` parameter.

Fitting

The ridge regression update is:

$$\boldsymbol{\beta}^{(j+1)} = \text{prox}_{0.5\lambda\alpha^{(j)}\,\|\cdot\|_2^2} \left(\boldsymbol{\beta}^{(j)} + \frac{\alpha^{(j)}}{|\mathcal{S}_j|} \sum_i^{\boldsymbol{x}_i \in \mathcal{S}_j} r_i^{(j)} \boldsymbol{x}_i\right),$$

where $\text{prox}_{\lambda\alpha^{(j)}\|\cdot\|_2^2}(\cdot)$ is element-wise application of the proximal operator of the function $0.5\lambda\alpha\,\|\cdot\|_2^2$, and $r_i^{(j)} = y_i - \boldsymbol{\beta}^{(j)\top}\mathbf{x}_i$ is the previous epoch's residual at point i.

The proximal operator for 2-norm, and any real $\gamma > 0$ is defined as:

$$\text{prox}_{0.5\gamma\|\cdot\|_2^2}(f) = (1 - \gamma) f.$$

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.formula predict.orch.ml.ridge oracle.model.matrix

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.ridge(formula = Number ~ Age, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

---

orch.ml.svd — *MLlib Singular Value Decomposition*

Description

MLlib reduced rank SVD approximation.

Usage

```
orch.ml.svd(formula, data, nSingularValues,
  computeU = FALSE, rCond = 1e-09, verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`nSingularValues`	Number of leading singular values to keep.
`computeU`	Whether to compute U, by default it's not computed.
`rCond`	The reciprocal condition number. All singular values smaller than `rCond * sigma` are treated as zero, where sigma is the largest singular value. Default values is `1E-9`.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

Singular value decomposition is defined as:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top,$$

where $\mathbf{A} \in \mathcal{R}^{m\times n}$, $\mathbf{U} \in \mathcal{R}^{m\times r}$, and $\mathbf{V} \in \mathcal{R}^{n\times r}$ are orthonormal; $\mathbf{\Sigma} \in \mathcal{R}^{r\times r}$ is diagonal; $r = rank\left(\mathbf{A}\right)$.

The diagonal elements of $\mathbf{\Sigma}$ are called *singular values* and are denoted as $\sigma_1, \sigma_2 \ldots \sigma_r$; the columns of $\mathbf{U}$ and $\mathbf{V}$ are called *left* and *right singular vectors*, respectively.

Solving SVD means finding components of the right hand side of the equation, given some known left-hand-side input $\mathbf{A}$.

In general case there is more than one SVD solution. The scope of solutions is artificially reduced by a convention. According to the convention, the singular values are positive; and the singular vectors and their corresponding singular vectors are arranged in the decreasing order such that $\sigma_{i+1} < \sigma_i$.

A *reduced-rank* SVD is the approximation of the full rank SVD such that

$$\mathbf{A} \approx \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top,$$

where only $k : k < r$ largest singular values and their corresponding singular vectors are taken to comprise right hand side matrices. Thus, $\mathbf{U}_k \in \mathcal{R}^{m\times k}$, $\mathbf{\Sigma}_k \in \mathcal{R}^{k\times k}$, and $\mathbf{V}_k \in \mathcal{R}^{n\times k}$.

MLlib SVD seeks an approximation of the reduced rank singular value decomposition, taking k as a user-specified parameter.

Value

SVD fit object, `orch.ml.svd`.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

`orch.formula` `oracle.model.matrix`

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.svd(formula = ~ Number + Age, data = data, nSingularValues = 1L)
```

`orch.ml.svm` *MLlib Support Vector Machine (SVM) with Stochastic Gradient Descent*

Description

Compute fit for linear SVM using MLlib.

Usage

```
orch.ml.svm(formula, data, convergenceTol = 1e-04,
  miniBatchFraction = 1, featureScaling = FALSE,
  maxIterations = 100L, regParam = 0.01, stepSize = 1,
  validateData = TRUE, verbose = TRUE)
```

Arguments

`formula`	An object of class `orch.formula` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under Details.
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory. Also it can accept an Oracle Model Matrix object prepared using `orch.model.matrix` function.
`convergenceTol`	Convergence tolerance. By default, `1E-4` is used as the stopping criterion.
`miniBatchFraction`	Fraction of the data to compute a gradient. By default, entire data is used for gradient evaluation.
`featureScaling`	Whether to carry out feature scaling before fitting the model. Default behavior being `FALSE`.
`maxIterations`	Maximum number of iterations. By default, `100` iterations is used as the stopping criterion.
`regParam`	Regularization parameter, default value `0.01`.
`stepSize`	Step size (learning rate), default value is `1.0`.
`validateData`	Whether to validate the input data. Default value is `TRUE`.
`verbose`	Whether to report progress and performance statistics. Default value is `TRUE`.

Details

Generalized linear models family of methods seeks to minimize a loss function employing 1-norm penalty over the fitted parameters $\boldsymbol{\beta}$.

Suppose we have the training dataset of predictors $\mathcal{D} = \{\mathbf{x}_i : i = 1, 2, \dots N\}$ and their corresponding target variables $\{y_i : i = 1, 2, \dots N\}$. Linear methods seek to minimize the loss function:

$$L(\boldsymbol{\beta}) = \frac{1}{2N} \sum_i \left(\boldsymbol{\beta}^\top \mathbf{x}_i - y_i\right)^2 + \lambda R(\boldsymbol{\beta}),$$

where λ is the regularization rate (parameter `regParam`), and $R(\boldsymbol{\beta})$ is the regularization penalty function.

Value

SVM fit object, `orch.ml.svm`.

Fitting

This MLlib version seeks solution using SGD (Stochastic Gradient Descent) over several training epochs. The maximum amount of epochs is controlled by the `maxIterations` parameter of the training procedure. During each epoch j, a fraction of the input is sampled into a minibatch $\mathcal{S}_j$, and then a partial loss gradient is computed and solution is updated according to:

$$\boldsymbol{\beta}^{(j+1)} = \boldsymbol{\beta}^{(j)} - \alpha^{(j)} \nabla_{\boldsymbol{\beta}} L\left(\boldsymbol{\beta}^{(j)}\right),$$

where $\alpha^{(j)}$ is the SGD learning rate in j-th epoch. In MLlib, the epoch learning rate $\alpha^{(j)}$ is subject to annealing schedule:

$$\alpha^{(j)} = \alpha\sqrt{j},$$

where α is the initial learning rate as supplied by the `stepSize` parameter.

Fitting

Linear SVM uses the hinge loss function along with L2 regularization:

$$L_{\text{hinge}}(\boldsymbol{\beta}) = \max\left(0, 1 - y\boldsymbol{\beta}^\top \mathbf{x}\right);$$

$$R(\boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{\beta}\|_2^2;$$

$$L(\boldsymbol{\beta}) = L_{\text{hinge}}(\boldsymbol{\beta}) + \lambda R(\boldsymbol{\beta}).$$

Although hinge loss is designed for use with labels $\{-1, 1\}$, MLlib gradient update implementation is adjusted for labels $\{0, 1\}$. Our formula performs all necessary adjustments automatically if a factor target variable is used; however, if class label is specified as a continuous target variable, that variable must be in $\{0, 1\}$.

Gradient updates within MLlib are performed using Stochastic Gradient Descent (SGD).

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

orch.formula predict.orch.ml.svm oracle.model.matrix

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.svm(formula = Kyphosis ~ Number + Age, data = data)
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

---

orch.model.matrix *Creates a distributed model matrix.*

Description

Machine learning and statistical algorithms require a Distributed Model Matrix (DMM) for their training phase. For supervised learning algorithms DMM captures a target variable and explanatory terms; for unsupervised learning DMM captures explanatory terms only. Internally Distributed Model Matrices are stored as Spark RDDs (Resilient Distributed Datasets).

Usage

```
orch.model.matrix(formula, data, type = "labeledPoint",
  factorMode = "oneHot", maxBlockRows = 20000L,
  verbose = TRUE)
```

Arguments

`formula`	A `formula` representing the model to be fit (see "details" section below for more information.)
`data`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS which represents a directory in HDFS. Alternatively it can be a string with HDFS compliant directory path relative to the current working directory.
`type`	"dmm" distributed model matrix type; "labeledPoint" (supervised learning), and "vector" (unsupervised learning) are supported.
`factorMode`	Factor mode. "oneHot" and "none" are supported.
`maxBlockRows`	Maximum number of rows in a partition. Smaller number or rows will create smaller partitions and more paritions. More data partitions ensures higher parallelization degree across Spark cluster but at the same time small paritions will cause higher communication and resource management overhead.
`verbose`	Whether to report progress and performance statistics. Default is `TRUE`.

Details

The following section describes the `formula` argument format and specification in details. For more information and examples you can also refer to the base R specification of `formula`.

Value

Distributed model matrix object.

ORAAH Formula

Everywhere below `A` can be either an ID (column name), it can also denote any generated column, for instance `sin(A / 10)`, or any subset of columns for instance `(A1 + A2 + A3)`.

Numerical engines, such as linear regression, cannot consume raw data; there must be a way to specify response and explanatory variables, nonlinear transformations, and interactions. Formula is such an engine, a recipe which specifies which columns (terms) to include to the model, and how to transform them if desired.

- `.` dot-character is a shortcut for all variables (all data columns), except the response.
- `+A` plus operator means to include this variable into the model. Plus operator here is used in set-theoretic sense. There is no arithmetic summation here of any kind. We add a term (column) to an ordered set of statistical terms (model).
- `-A` minus operator means to remove this variable from the model. Example `- (A + B)` removes both A and B variables from the model. Example `. - (A + B)` includes all variables, except A, B, and the response.
- `A : B` include the interaction between A and B variables.
- `A * B` include these variables and the interactions between them. This is equivalent to `A + B + A : B`. `Example A * ( . - B ) * Z`
- `(A1 + A2 + ... + Ak)^n` include these variables and all interactions up to n-way. For instance, `(A + B)^2` is equivalent to `A + B + A : B`. The exponentiation (the power operator) can lead to much more compact model specification. For instance `(. - A)^3` will include all variables, excluding the A and the response, and will include the corresponding interactions. To reiterate, A can be either an ID (variable name) or any complex term. For instance, `(log(A) + B : Z)^2` is equivalent to `log(A) + B : Z + log(A) : B : Z`
- `I()` Identity function. Its argument will be treated in arithmetic sense (as versus set-theoretic sense). For instance: `I(log(A) + B)` will include a new column, whose elements are `log(A[k]) + B[k]`. Here, the plus operator (and all other operators) will be treated in their traditional arithmetic sense.
- 24 arithmetic functions. The argument will be treated in arithmetic sense. Example `log(A / 10 + B * Z)`.

abs	acos	asin
atan	cbrt	ceil
cos	cosh	exp
expm1	floor	log
log10	log1p	rint
round	signum	sin
sinh	sqrt	tan
tanh	toDegrees	toRadians

- Relational operators, currently supported for numerical terms only. Example Y ~ X + (A > B).

A >= B	A <= B
A > B	A < B
A == B	A != B

```
A && B   A || B
A & B    A | B
```

- `+1` Add the intercept.
- `-0` Add the intercept (equivalent to +1).
- `-1` Delete the intercept.
- `+0` Delete the intercept (equivalent to -1).

It is very important to keep in mind, that all factor variables (including factor-factor and factor-numeric interactions), are unrolled following one-hot scheme, meaning internally they will be substituted by `k-1` dummy variables.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

`hdfs.attach` `hdfs.write` `hdfs.get` `hdfs.sample`

Examples

```
library(rpart)
data <- hdfs.put(kyphosis)
modelMatrix <- orch.model.matrix(Kyphosis ~ Number, data = data)
hdfs.rm(data)
```

`orch.neural2` *High performance multilayer feed-forward neural network on Spark with L-BFGS algorithm*

Description

The `orch.neural2` function solves multilayer feed-forward neural network models. It supports an arbitrary number of hidden layers and an arbitrary number of neurons per layer. Each layer could be assigned a different activation function. The L-BFGS algorithm is used to solve the underlying unconstrained nonlinear optimization problem.

Usage

```
orch.neural2(formula, data, weight = NULL,
  hiddenSizes = NULL, activations = NULL,
  gradTolerance = 1e-09, maxIterations = 200L,
  objMinProgress = 1e-06, lowerBound = -0.7,
  upperBound = 0.7, seed = as.integer(1e+08 * runif(1)),
  nUpdates = 20L, scaleHessian = TRUE,
  maxBlockRows = 20000L,
  verbose = getOption("orch.trace", FALSE))
```

Arguments

`formula`	A formula object.
`data`	An HDFS object specifying the data for training the model.
`weight`	A vector of initial weights. If not specified, the initial weights will be randomly generated.
`hiddenSizes`	An integer vector, whose elements store the number of neurons in each hidden layer. `orch.neural2` supports an arbitrary number of hidden layers. The length of `hiddenSizes` indicates the number of hidden layers in the model, and `hiddenSizes[k]` stores the number of neurons in the `k`-th hidden layer. If not specified, the input units will be directly connected to the output neurons (no hidden structure). If any element of `hiddenSizes` is zero, then all hidden neurons will be dropped, which is equivalent to `hiddenSizes=NULL`. Example: `hiddenSizes=c(10, 4)` specifies a neural network with two hidden layers (`length(hiddenSizes)` is 2); the first hidden layer will have 10 neurons, and the second one will have 4.
`activations`	A vector of activation functions for the hidden and the output neural network layers. The `orch.neural2` function supports a single activation function per layer. Neurons are grouped into layers, and each layer (a subset of neurons) can be assigned its own activation function. Note: the target variable range needs to correspond to the range of the output activation function. For instance, logistic sigmoid can be used to model targets in the range of zero to one (range of the sigmoid function). The `orch.neural2` function does not preprocess the input data; appropriate data normalization and scaling are strongly recommended. If not specified, the activation function for each hidden layer is bipolar sigmoid and for the output it is linear. If `activations` is specified, its size must be `length(hiddenSizes) + 1`, where the last element corresponds to the output layer. Possible values:

`"atan"`	arctangent	$f(x) = \arctan x$
`"bSigmoid"`	bipolar sigmoid	$f(x) = \frac{1-e^{-x}}{1+e^{-x}}$
`"linear"`	linear	$f(x) = x$
`"sigmoid"`	logistic sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
`"tanh"`	hyperbolic tangent	$f(x) = \tanh x$
`"entropy"`	entropy (output only)	$f(x) = \log(1 + \exp(x)) - yx$

Example: `activations=c("sigmoid", "tanh", "linear")` corresponds to a neural network with two hidden layers. The first hidden layer is assigned the `sigmoid` activation function, the second hidden layer is as-

	signed the `tanh` activation function, and the output (target) layer is assigned the `linear`.
`gradTolerance`	Numerical optimization stopping criterion: desired gradient norm.
`maxIterations`	Numerical optimization stopping criterion: maximum number of iterations.
`objMinProgress`	Numerical optimization stopping criterion: minimal relative change in the objective function value.
`lowerBound`	Lower bound for the weight initialization (not used if `weight` is specified).
`upperBound`	Upper bound for the weight initialization (not used if `weight` is specified).
`seed`	pseudo-random number generator seed, for weight initialization.
`nUpdates`	Number of L-BFGS update pairs.
`scaleHessian`	A logical value that indicates whether to scale the inverse of the Hessian matrix in L-BFGS updates.
`maxBlockRows`	maximum number of rows in a model matrix partition.
`verbose`	A logical value that indicates whether to print out execution information.

Value

A neural network model object, `orch.neural2`.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

Examples

```
# regression with iris dataset
    IRIS <- hdfs.put(iris)
    mod<- orch.neural2(formula        = Sepal.Length ~.,
                       data           = IRIS,
                       hiddenSizes    = c(10, 10),
                       activations    = c("sigmoid", "tanh", "linear"),
                       seed           = 0,
                       objMinProgress = 1e-5,
                       maxIterations  = 400,
                       verbose        = TRUE)
    summary(mod)
    p <- predict(mod, IRIS, supplemental=c("Species", "Sepal.Length"))
    IrPred.dfs <- hdfs.write(p, "IrPred", overwrite=TRUE)
    IrPred <- hdfs.get(IrPred.dfs)

    # binary classification with kyphosis dataset
    library(rpart)
    KYPHOSIS <- hdfs.put(kyphosis)
```

```
mod<- orch.neural2(formula        = Kyphosis ~.,
                   data           = KYPHOSIS,
                   hiddenSizes    = c(20, 20),
                   activations    = c("sigmoid", "sigmoid", "entropy"),
                   seed           = 0,
                   verbose        = TRUE)
p <- predict(mod, KYPHOSIS, supplemental=c("Age", "Kyphosis"))
KyPred.dfs <- hdfs.write(p, "KyPred", overwrite=TRUE)
KyPred <- hdfs.get(KyPred.dfs)
```

`orch.neural` *Multilayer Feed-Forward Neural Network for Oracle R Connector for Hadoop*

Description

Multilayer feed-forward neural network on HDFS data.

Usage

```
orch.neural(
    formula,
    dfs.dat,
    weight          = NULL,
    xlev            = NULL,
    hiddenSizes     = NULL,
    activations     = NULL,
    gradTolerance   = 1E-1,
    maxIterations   = 200L,
    objMinProgress = 1E-6,
    lowerBound      = -0.7,
    upperBound      = 0.7,
    nUpdates        = 20L,
    scaleHessian    = TRUE,
    trace           = getOption("orch.trace", FALSE),
    nMappers        = -1L,
    nReducers       = 1L,
    mapSplit        = 0)

## Specific methods for orch.neural objects
## S3 method for class 'orch.neural'
predict(object, newdata, supplemental.cols = NULL, ...)
## S3 method for class 'orch.neural'
print(x, ...)
## S3 method for class 'orch.neural'
coef(object, ...)
## S3 method for class 'orch.neural'
summary(object, ...)
```

Arguments

`formula` A `formula` object representing the neural network model to be trained.

`dfs.dat` The HDFS object specifying the data for the model. Alternatively, it can also be an RDD created using `orch.prepare`, `orch.orch.prepare.model.matrix`.

`hiddenSizes`

An integer vector, whose elements store the number of neurons in each hidden layer. `orch.neural` supports an arbitrary number of hidden layers. The length of `hiddenSizes` gives the number of hidden layers in the model, and `hiddenSizes[k]` stores the number of neurons in the `k`-th hidden layer. The `hiddenSizes` value may be `NULL`, in which case input units will be directly connected to the output neurons (no hidden structure). If any element of `hiddenSizes` is zero, then all hidden neurons will be dropped, which is equivalent to `hiddenSizes=NULL`.

Example: `hiddenSizes=c(10, 4)` specifies a neural network with two hidden layers (`length(hiddenSizes)` is 2); the first hidden layer will have 10 neurons, and the second one will have 4.

Example: `hiddenSizes=c(101, 20, 1)` specifies a neural network with three hidden layers, with 101, 20, and 1 units correspondingly.

In a typical training scenario (assuming no prior knowledge of the model), you may start with a single hidden layer and a small number of hidden neurons (for instance, `hiddenSizes=1`). You may then gradually increase the number of neurons (and possibly layers) until no further error reduction can be observed on the validation data set.

`activations`

This argument specifies activation functions for the hidden and the output neural network layers. The `orch.neural` function supports a single activation function per layer. Neurons are grouped into layers, and each layer (a subset of neurons) can be assigned its own activation function. Note: the target variable range needs to correspond to the range of the output activation function. For instance, logistic sigmoid can be used to model targets in the range of zero to one (range of the sigmoid function). The `orch.neural` function does not pre-process the input data; appropriate data normalization and scaling are strongly recommended.

If the `activations` argument is `NULL`, then the activation function for each hidden layer is bipolar sigmoid and for the output it is linear.

If `activations` is not `NULL`, then its size must be

`length(hiddenSizes) + 1`,

where the last element corresponds to the output layer.

Possible values:

`"atan"`	arctangent	$f(x) = \arctan x$
`"bSigmoid"`	bipolar sigmoid	$f(x) = \frac{1-e^{-x}}{1+e^{-x}}$
`"cos"`	cosine	$f(x) = \cos x$
`"gaussian"`	Gaussian	$f(x) = e^{-x^2}$
`"gaussError"`	Gauss error	$f(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$
`"gompertz"`	Gompertz	$f(x) = e^{-e^{-x}}$
`"linear"`	linear	$f(x) = x$
`"reciprocal"`	reciprocal	$f(x) = \frac{1}{x}$
`"sigmoid"`	logistic sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
`"sigmoidModulus"`	sigmoid modulus	$f(x) = \frac{x}{1+\|x\|}$

`"sigmoidSqrt"`	sigmoid sqrt	$f(x) = \frac{x}{\sqrt{1+x^2}}$
`"sin"`	sine	$f(x) = \sin x$
`"square"`	square	$f(x) = x^2$
`"tanh"`	hyperbolic tangent	$f(x) = \tanh x$
`"wave"`	wave	$f(x) = \frac{x}{1+x^2}$
`"entropy"`	entropy (output only)	$f(x) = \log(1 + \exp(x)) - yx$

Example: `activations=c("wave", "tanh", "linear")` corresponds to a neural network with two hidden layers. The first hidden layer is assigned the `"wave"` activation function, the second hidden layer is assigned the `"tanh"` activation function, and the output (target) layer is assigned the `"linear"`.

`gradTolerance`	Numerical optimization stopping criterion: Desired gradient norm.
`maxIterations`	Numerical optimization stopping criterion: Maximum number of iterations.
`objMinProgress`	Numerical optimization stopping criterion: minimal relative change in the objective function value.
`nUpdates`	Number of L-BFGS update pairs.
`scaleHessian`	Whether to scale the inverse of the Hessian matrix in L-BFGS updates.
`lowerBound`	Lower bound for the weight initialization (not used if weights are supplied).
`upperBound`	Upper bound for the weight initialization (not used if weights are supplied).
`weight`	Initial vector of weights (may be NULL, in which case a random starting point will be generated). Useful when using a solution from a previously solved model. Note: the previous neural network architecture (number of input, output, hidden layers and hidden neurons in each layer and the type of activation functions), should be identical to the current one.
`xlev`	A named `list` of `character` vectors specifying the `levels` for each `ore.factor` variable.
`trace`	Report iteration log
`nMappers`	Hint for number of mappers to be used for the Hadoop jobs. Hadoop defaults are used.
`nReducers`	Hint for number of reducers to be used for the Hadoop jobs. Default is 1.
`mapSplit`	Number of records to supply at once to a mapper. See `map.split` in `mapred.config`
`object, x`	An `orch.neural` object.
`newdata`	The HDFS object, test data.
`supplemental.cols`	Additional columns to include in the prediction result from the `newdata` data set.
`...`	Additional arguments.

Details

The `orch.neural` function solves multilayer feed-forward neural network models. It supports an arbitrary number of hidden layers and an arbitrary number of neurons per layer. The L-BFGS algorithm is used to solve the underlying unconstrained nonlinear optimization problem.

Value

`orch.neural` returns an object of class `orch.neural`. Some of its components are as follows:

`weight`	Weight coefficients.
`nLayers`	Number of layers.

`summary.orch.neural` returns a `summary.orch.neural` object.

`predict.orch.neural` returns an `hdfs.id` object which corresponds to the output HDFS file.

The output file contains a key column in addition if `newdata` had a key. The value of the key column can be used to associate a record in `newdata` with its corresponding record in the output file.

The format of the output file is as follows: If key column is present it will appear first. This will be followed by the remaining columns in `newdata` specified by `supplemental.cols` argument. The ordering among these columns is preserved. Finally, the column corresponding to the prediction results follows.

`coef.orch.neural` returns the coefficients of the `orch.neural` object as a named numeric vector.

Execution Scenarios

`orch.neural` can compute the model from data in HDFS over Hadoop or Spark (if connected using spark.connect). Different scenarios for invocation of `orch.neural` are described below:

1) *Spark not connected:* In this case, all computations are performed over Hadoop. `orch.prepare` & `orch.prepare.model.matrix` are both a no-op if Spark is not connected.

```
For example:
  IRIS <- hdfs.put(iris)
  sformula <- Petal.Length ~ Petal.Width + Sepal.Length
  fit <- orch.neural( formula     = sformula,
                      dfs.dat     = IRIS,
                      hiddenSizes = c(20L, 5L),
                      activations = c("bSigmoid", "tanh", "linear"),
                      maxIterations = 5L )
```

2) *Spark connected but data not prepared:* In this case, if the input data is in Text CSV format and the formula is simple, then computations will be performed over Spark. Though Spark cache is not utilised without the use of prepare functions.

```
For example:
  spark.connect("<spark_master_address>", memory="2g",
                dfs.namenode="<hdfs_name_node_address>")
  IRIS <- hdfs.put(iris)
  sformula <- Petal.Length ~ Petal.Width + Sepal.Length
  fit <- orch.neural( formula     = sformula,
                      dfs.dat     = IRIS,
                      hiddenSizes = c(20L, 5L),
                      activations = c("bSigmoid", "tanh", "linear"),
                      maxIterations = 5L )
  spark.disconnect()
```

3) *Spark connected and data cached:* In this case, data has been cached using `orch.prepare` into Spark cache memory. The computations happen over Spark with a significant performance improvement with the use of cache. But the model matrix for the specific formula will be computed for all iterations.

```
For example:
  spark.connect("<spark_master_address>", memory="2g",
                dfs.namenode="<hdfs_name_node_address>")
  IRIS <- hdfs.put(iris)
  sformula <- Petal.Length ~ Petal.Width + Sepal.Length
  IRISprep <- orch.prepare(IRIS)
  fit <- orch.neural( formula     = sformula,
                      dfs.dat     = IRISprep,
                      hiddenSizes = c(20L, 5L),
                      activations = c("bSigmoid", "tanh", "linear"),
                      maxIterations = 5L )
  spark.disconnect()
```

4) *Spark connected and model matrix cached:* In this case the model matrix specific to the formula is cached in Spark memory using `orch.prepare.model.matrix`. The data is read once and the model matrix is cached. All the iterations use this model matrix directly. The neural model computation performance is highest in this case.

```
For example:
  spark.connect("<spark_master_address>", memory="2g",
                dfs.namenode="<hdfs_name_node_address>")
  IRIS <- hdfs.put(iris)
  sformula <- Petal.Length ~ Petal.Width + Sepal.Length
  IRISprepMat <- orch.prepare.model.matrix(sformula, IRIS)
  fit <- orch.neural( formula     = sformula,
                      dfs.dat     = IRISprepMat,
                      hiddenSizes = c(20L, 5L),
                      activations = c("bSigmoid", "tanh", "linear"),
                      maxIterations = 5L )
  spark.disconnect()
```

References

Christopher Bishop (1996) *Neural Networks for Pattern Recognition*

Simon Haykin (2008) *Neural Networks and Learning Machines (3rd Edition)*

Stephen Marsland (2009) *Machine Learning: An Algorithmic Perspective*

Examples

```
##############################################################
# Two hidden layers (20 neurons in the first layer, 5 hidden  #
# neurons in the second layer).                               #
#                                                             #
# Use bipolar sigmoid activation function for the first       #
# hidden layer, hyperbolic tangent for the second hidden      #
# layer, and linear activation function for the output layer. #
#                                                               #
# Note that the dimension (number of elements) of the          #
# "activations" argument is always greater by exactly one      #
# than the dimension of "hiddenSizes".                         #
#                                                               #
# Least-squares objective function.                            #
################################################################
IRIS <- hdfs.put(iris)

fit <- orch.neural(Petal.Length ~ Petal.Width + Sepal.Length,
  dfs.dat     = IRIS,
  hiddenSizes = c(20L, 5L),
  activations = c("bSigmoid", "tanh", "linear"),
  maxIterations = 5L)

ansPred <- predict(fit, newdata = IRIS,
  supplemental.cols = c("Petal.Length"))

ans <- hdfs.get(ansPred)

################################################################
# Entropy objective function.                                  #
################################################################
INFERT <- hdfs.put(infert)

fit <- orch.neural(case ~ ., dfs.dat = INFERT,
  activations = c('entropy'), objMinProgress = 1E-7,
  maxIterations = 10L)

# Entropy (max likelihood) model with one hidden layer.
fit <- orch.neural(
  formula        = case ~ .,
  dfs.dat        = INFERT,
  hiddenSizes    = c(40L),
  activations    = c("sigmoid", "entropy"),
  lowerBound     = -0.7,
  upperBound     = 0.7,
  objMinProgress = 1E-12,
  maxIterations  = 10L)
```

orch.nmf *Nonnegative matrix factorization (NMF)*

Description

Builds an NMF model, returning an NMF model instance.

Usage

```
orch.nmf(input, method =c("jellyfish"), dfs.output = NULL, ...)
```

Arguments

input	A CSV ratings file containing entries of the form (user, item, rating). This can be one of the following

1. the HDFS directory containing the input file
2. R data.frame object
3. ore.frame object
4. name of a file in the local file system

`method`	The method to be used. Currently only `jellyfish` is supported.
`dfs.output`	The output HDFS directory where the model should be created. If not specified, this method will internally create a directory and use that as the output directory.
`...`	Optional method specific arguments are:
`latin`	Latin Square dimension for Map Reduce parallelism. This is an optional argument. The default value is computed based on the memory per mapper.
`rank`	The rank of the latent factor matrices. This is an optional argument with default value of 50.
`iterations`	Number of iterations of Incremental Gradient Descent (IGD) to be performed. This is an optional argument with default value 10.
`step`	Learning Rate / Step size to be used in IGD. This is an optional argument with default value 0.05.
`decay`	Decay parameter for step size to be used in IGD. This is an optional argument with default value 0.8.
`regularizer`	Regularization parameter to be used in IGD. This is an optional argument with default value 2.3.
`init`	Values for initialization of factors will be uniformly chosen from (0 .. init). This is an optional argument with default value 1.
`seed`	Seed value for random number generation. This is an optional argument.
`mapmem`	Amount of memory available per mapper in MB. This is an optional argument with default value 200.

Details

The `jellyfish` algorithm implements a projected incremental gradient descent method. Massive parallelization of the gradient computations are achieved by partitioning the matrix into chunks.

Value

Returns an instance of NMF model class, an object of `orch.nmf.jellyfish`

This is a list with the following components

`lmffit`	The `orch.lmf.jellyfish` LMF model that is used underneath

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

See Also

`orch.lmf`

Examples

```
## Setup the input (term, doc, freq) entries
t <- sample(1:50, 300, replace=TRUE)
d <- sample(1:100, 300, replace=TRUE)
td <- unique(cbind(t,d))
f <- sample(1:5, nrow(td), replace=TRUE)
input <- cbind(td,f)

# Fit an "orch.nmf.jellyfish" model
fit <- orch.nmf(input, latin=2, iterations=5, rank=5)
print(fit)
```

`orch.predict` *Oracle R Connectors for Hadoop Predictions Using R Models*

Description

Generic for model predictions in ORCH

Usage

```
orch.predict(object, newdata, ...)
```

Arguments

`object`	A model object.
`newdata`	An HDFS object.
`...`	Optional arguments for implemented methods.

Value

Returns an HDFS object, usually the `hdfs.id` of the HDFS file containing the predictions.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

orch.prepare.model.matrix2

Generate model matrix for orch.neural2.

Description

This function will return a model matrix RDD in Spark that is prepared for orch.neural2.

Usage

```
orch.prepare.model.matrix2(formula, data,
  maxBlockRows = 20000L, verbose = TRUE)
```

Arguments

`formula`	A formula object.
`data`	An HDFS object specifying the data for the model matrix.
`maxBlockRows`	maximum number of rows in a model matrix partition.
`verbose`	whether to report progress.

Value

model matrix RDD.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

Examples

```
library(rpart)
KYPHOSIS <- hdfs.put(kyphosis)
mm = orch.prepare.model.matrix2(Kyphosis~., KYPHOSIS)
```

orch.prepare.model.matrix

Prepare model matrix from HDFS data

Description

This function will return an HDFS id object which also refers to the cached model matrix in Spark cache. This HDFS id, if given to `orch.neural` as `dfs.dat`, will lead to model computation to happen over Spark framework and provide significant performance improvement.

Usage

```
orch.prepare.model.matrix(formula, dfs.dat, xlev = NULL)
```

Arguments

`formula`	A `formula` object representing the neural network model to be trained.
`dfs.dat`	The HDFS object specifying the data for the model.
`xlev`	A named `list` of `character` vectors specifying the `levels` for each factor variable.

Value

An enhanced HDFS object specifying the data for the model.

Attention

This function should be called after a `spark.connect`. Failing to do so won't provide any performance gain and Hadoop framework will be utilised.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`www.oracle.com/technetwork/bdc/big-data-connectors`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

spark.connect spark.connected

Examples

```
IRIS <- hdfs.put(iris)
iris_formula <- Petal.Length ~ Petal.Width
IRIS_mm <- orch.prepare.model.matrix(iris_formula, IRIS)

# Use IRIS_mm for orch.neural
if (spark.connected())
  iris_fit <- orch.neural(iris_formula, IRIS_mm, trace=TRUE)
```

`orch.prepare` *Prepare HDFS data*

Description

This function will return an HDFS id object which refers to the cached input in Spark cache. If given to `orch.neural` as `dfs.dat`, it will route the computation over Spark framework and provide significant performance improvement.

Usage

```
orch.prepare(dfs.dat)
```

Arguments

`dfs.dat`	The HDFS object specifying the data.

Value

An enhanced HDFS object specifying the data for the model.

Attention

This function should be called after doing a `spark.connect`. Failing to do so won't provide any performance gain, since existing Hadoop framework will be utilised.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`www.oracle.com/technetwork/bdc/big-data-connectors`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

See Also

spark.connect spark.connected

Examples

```
# Prepare Data
IRIS <- hdfs.put(iris)
IRIS_data <- orch.prepare(IRIS)

# Use IRIS_data for orch.neural
if (spark.connected())
  iris_fit <- orch.neural(Petal.Length ~ Petal.Width, IRIS_data, trace=TRUE)
```

`orch.princomp` *Principal Components Analysis*

Description

Principal components analysis of HDFS data.

Usage

```
orch.princomp(x, cor=FALSE, num.mappers=-1, num.reducers=1,
              reducer.serial.limit=8, task.timeout=-1,
              job.name="ORCH PCA")
```

Arguments

`x`	An `hdfs.id` object containing numeric columns. This input matrix is in dense matrix representation
`cor`	A logical value that indicates whether the principal components should be based on the correlation matrix (`cor = TRUE`) or the covariance matrix (`cor = FALSE`).
`num.mappers`	Hint for number of mappers to be used for the Hadoop jobs. Hadoop defaults are used.
`num.reducers`	Hint for number of reducers to be used for the Hadoop jobs. Default is 1.
`reducer.serial.limit`	Maximum number of records later phase reducers should process serially
`task.timeout`	Maximum time in seconds a map or reduce task is allowed to run before it gets force killed by Hadoop. Hadoop defaults are used.
`job.name`	Prefix to be used for the Hadoop job names

Details

This is a wrapper method around the `princomp` function in the **stats** package to perform Prinicpal Components Analysis on HDFS objects.

Value

A `princomp` object.

See Also

`princomp`

Examples

```
USARRESTS <- hdfs.put(USArrests)

orch.princomp(USARRESTS)
orch.princomp(USARRESTS, cor = TRUE)
```

`orch.recommend` *Recommend Top N*

Description

This function computes top N items to be recommended for each user from LMF models.

Usage

```
  orch.recommend(object, ...)

  ## S4 method for signature 'orch.mahout.lmf.als'
orch.recommend(object, dfs.output = NULL, n, maxRating)
```

Arguments

`object`	An instance of a LMF model of type `mahout-als`
`dfs.output`	The output HDFS directory where the recommendations output file will be created. If not specified, this method will internally create a directory and use that as the output directory.
`n`	Number of items to recommend for each user
`maxRating`	The maximum possible rating value per item

Value

Returns the HDFS directory containing the output file.

Methods

`signature(object = "orch.mahout.lmf.als")` This function computes top N items to be recommended for each user using the predicted ratings based on the input `orch.mahout.lmf.als` model instance.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

Examples

```
## Setup the input (user, item, rating) entries
u <- sample(1:100, 300, replace=TRUE)
i <- sample(1:10, 300, replace=TRUE)
ui <- unique(cbind(u,i))
r <- sample(1:5, nrow(ui), replace=TRUE)
input <- cbind(ui,r)

# For "mahout-als", set up an input file
inputFile <- tempfile(tmpdir='/tmp')
write.table(input, file=inputFile, sep=",", col.names=FALSE, row.names=FALSE)

# Fit using "mahout-als"
fit <- orch.lmf(inputFile, method="mahout-als", rank=3, iterations=5)

# Recommend top 2 items per user
orch.recommend(fit, n=2, maxRating=5)
```

`orch.save.model` *Save MLlib Models to HDFS.*

Description

This function saves a model created using Spark MLlib in ORAAH to hdfs for scoring/prediction later on. It also enables model sharing amongst different users if the other users have access to the path where models are saved.

Usage

```
orch.save.model(model, dfs.name, overwrite = FALSE)
```

Arguments

`model`	MLlib fit object of type among `orch.ml.logistic`, `orch.ml.linear`, `orch.ml.lasso`, `orch.ml.ridge`, `orch.ml.svm`, `orch.ml.gmm`, `orch.ml.kmeans`, `orch.ml.dt` or `orch.ml.random.forest`.
`dfs.name`	Name of the target HDFS directory or HDFS path relative to the current working directory. If the directory does not exist in HDFS it will be created. If it exists `overwrite` parameter must be considered.
`overwrite`	whether to overwrite the destination directory if it exists.

Value

HDFS absolute path to the saved model location.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.load.model

Examples

```
library(rpart)
data   <- hdfs.put(kyphosis)
model  <- orch.ml.ridge(formula = Number ~ Age, data = data)
orch.save.model(model, "ridgeKypSave", overwrite=TRUE)
model.load <- orch.load.model("ridgeKypSave")
pred   <- predict(model.load, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite=TRUE)
```

`orch.unprepare` *Uncache data from Spark cache*

Description

This function will uncache data or model matrix cached into Spark cache using `orch.prepare` or `orch.prepare.model.matrix` functions.

Usage

```
orch.unprepare(dfs.dat)
```

Arguments

`dfs.dat`	The HDFS object specifying the data.

Attention

This function should be called only with an active spark session. Also the `dfs.dat` should be cached in spark using `orch.prepare` or `orch.prepare.model.matrix`. If not, the function will error out.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

www.oracle.com/us/products/database/big-data-connectors

www.oracle.com/technetwork/bdc/big-data-connectors

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

See Also

orch.prepare.model.matrix orch.prepare

Examples

```
IRIS <- hdfs.put(iris)

# Prepare Data
IRIS_data <- orch.prepare(IRIS)

# Prepare Model Matrix
IRIS_mm <- orch.prepare.model.matrix(Petal.Length ~ Petal.Width, IRIS)

# Once cached data/model matrix is not needed
# use orch.unprepare to uncache it
orch.unprepare(IRIS_data)
orch.unprepare(IRIS_mm)
```

`predict.orch.lmf` *Predict using a Low Rank Matrix Factorization Model*

Description

This function can be used to make predictions using an LMF model. For instance, if the input consists of (user, item) pairs, then this function can be used to predict the ratings of the user on the item for each pair.

Usage

```
## S3 method for class 'orch.lmf.jellyfish'
predict(object, newdata, dfs.output=NULL)
```

Arguments

`object`	An instance of a `orch.lmf.jellyfish` model
`input`	Input containing entries of the form (user, item). This can be one of the following 1. the HDFS directory containing the input file 2. R data.frame 3. ore.frame 4. name of a file in the local file system
`dfs.output`	The output HDFS directory where the predicted ratings should be created. If not specified, this method will internally create a directory and use that as the output directory.

Value

A list with the following components

`inputDir`	The HDFS directory containing the input
`outputDir`	HDFS output directory that contains the predicted ratings

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

Examples

```
## Setup the input (user, item, rating) entries
u <- sample(1:100, 300, replace=TRUE)
i <- sample(1:10, 300, replace=TRUE)
ui <- unique(cbind(u,i))
r <- sample(1:5, nrow(ui), replace=TRUE)
input <- cbind(ui,r)

# Fit an "orch.lmf.jellyfish" model
fit <- orch.lmf(input, latin=2, iterations=5, rank=3)
print(fit)

# Get the input on which predictions are desired
# This is a subset of u and subset of i used in the training data set
up <- sample(u, 10, replace=TRUE)
ip <- sample(i, 10, replace=TRUE)
pred.input <- cbind(up, ip)

# Make the prediction using the orch.lmf.jellyfish model
pred.results <- predict(fit, newdata=pred.input)

# Get the predictions into R and display
preddf <- hdfs.get(hdfs.attach(pred.results$outputDir))
pj <- as.matrix(preddf)
pj
```

`summary.orch.neural2` *Neural network summary.*

Description

Neural network summary.

Usage

```
summary.orch.neural2(object, ...)
```

Arguments

`object`	An orch.neural2 model object.

Value

A `summary.orch.neural2` object.

Author(s)

Oracle `<oracle-r-enterprise@oracle.com>`

References

`www.oracle.com/us/products/database/big-data-connectors`

`docs.oracle.com/en/bigdata`

`docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm`

`orch.testkit` *Oracle R Advanced Analytics test kit.*

Description

The function executes Oracle R Advanced Analytics for Hadoop (ORAAH) internal unit test kit, which will test all core components, specifically ORCHcore package functionality. This test kit enables test and pre-certification of ORAAH on Hadoop distributions not (yet) officially certified by Oracle. Running the unit tests ensures that the product functions correctly, without errors, and compatible with the current Hadoop configuration.

Usage

```
orch.testkit(test, long = FALSE, severity = "fatal")
```

Arguments

`test`	Specific unit test name or a regexp pattern of a test name range. If it is not specified or NULL, then all available ORAAH unit tests will be run. This option is especially useful to re-run only failed tests from a previous run after fixing the possible cause of the test failure. Also you can specify an ORAAH API function to test and all tests specific to that function alone will be executed.
`long`	Specifies which version of the tests to run - long or short. Long version may take several hours to run but ensures that all corner cases, special functions and known bugs are tested. Short version will run only "barebone" tests, i.e. the most important and core tests. It's wise to run the "short" tests first and if they are clean then to run "long" test to make sure that the software is functioning correctly.
`severity`	Error log severity during the tests. By default, "fatal" severity is used to monitor internal ORAAH failures. See orch.dbg.on help page for the list of available severity levels. Enabling higher severity level allows to debug issue by inspecting the output log. Note that the log output can be quite large and will slow down the test execution.

Details

Any errors reported by the test kit indicate possible issues in configuration of the product itself or in Hadoop installation and configuration.

Value

TRUE if all tests have passed, otherwise FALSE.

Author(s)

Oracle <oracle-r-enterprise@oracle.com>

References

www.oracle.com/us/products/database/big-data-connectors

docs.oracle.com/en/bigdata

docs.oracle.com/cd/E37231_01/doc.20/e36961/orch.htm

Examples

```
## Not run:
orch.testkit(long=FALSE)    # run all ORAAH "short" tests
orch.unit.test("^bug")      # run all bug tests
orch.unit.test("^test")     # run all unit tests
orch.unit.test("hadoop")    # run all unit tests for "hadoop.*" functions.
orch.unit.test(orch.export) # test one function only

## End(Not run)
```

Index