Oracle R Advanced Analytics for Hadoop 2.8.2

May 15, 2020

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

Description

Invokes the Hadoop engine and sends mapper and reducer R functions for execution. If the input data does not reside in HDFS then copies the data into HDFS first. Prepares the user's mapReduce scripts for execution in the distributed Hadoop environment, invokes Hadoop engine, while 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. It represents either a directory in HDFS, or is a string with an HDFS-compliant path relative to the current working directory.
`out.name`	Output HDFS directory name or HDFS object identifier of the output data. Note that the output directory must not exist when the Hadoop job is submitted, else the job fails. If the output directory is not specified, a temporary directory is created in HDFS "/tmp".
`mapper`	Optional mapper function written in the R language. The function must accept two values: "key" and "value". The names of the arguments do not matter. Prototype is: mapper = function(k,v) . If the mapper function is not specified or is NULL, then a reduce-only job is executed.
`combiner`	Optional combiner function written in the R language. The function must accept two values: "key" and "value". The names of the arguments do not matter. Prototype is: combiner = function(k,v) .
`reducer`	Optional reducer function written in the R language. The function must accept two values: "key" and "value". The names of the arguments do not matter. Prototype is: reducer = function(k,v) . If a reducer function is not specified or is NULL, then map-only job is executed.

`export` Exported R objects. This argument copies the specified R objects from the user's R session into the server running mapReduce R scripts. The object or its clone is thus made visible to the mapReduce jobs during execution. See orch.export and examples for more details.

`init` Optional job initialization function. This function is called once before any user's mapReduce functions are invoked. It enables the user to do initial preparation, initialization, or memory allocation as required for the mapReduce algorithm. The function does not accept any arguments. Prototype is: init = function().

`final` Optional job finalization function. This function is called once after all the user's mapReduce functions have completed. It enables the user to do final data processing or memory de-allocation as required by the mapReduce algorithm. In addition keys and vales can be output in the final function (see orch.keyvals). The function does not accept any arguments. Prototype is: final = function().

`job.name` Optional name of this mapReduce job. By default Hadoop's job ID is the job name. Tip: always provide a meaningful name in order to make it easier to locate the job in the Hadoop run logs.

`config` Optional mapReduce advanced configuration class. This argument allows the user to fine-tune various aspects of the mapReduce job in order to achieve better performance or to change the behavior of the ORCH mapReduce driver. This argument is an instance of the "mapred.config" class and therefore it has this format: config = new("mapred.config", param1, param2,...).

`cleanup` If `TRUE`, runs a cleanup procedure after the mapReduce job finishes succesfully. The cleanup removes all empty "part" files and all Hadoop log files.

`overwrite` Allows overwriting of HDFS objects with the same name. By default overwrite is disabled data safety.

`attach` Enable or disable automatic attachment of the result of the Hadoop job. If disabled then the returned HDFS object identifier points to the HDFS directory without ORCH metadata.

`tmp.result` if `TRUE`, the 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 will be temporary and will be removed beyond this R session. This option disables writing ORCH metadata to HDFS and keeps the data cached in the memory 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. Unlike hadoop.run, this function never converts the results back into the original input data formats.

This function differs in design from hadoop.run. Its purpose is optimization of multi-stage mapReduce jobs when output of this job is not the final result and becomes input for the next stage. It bypasses data conversion and management procedures and therefore lowers overhead for cases when these are not required by an R workflow.

Value

Resulting HDFS object identifier if everything worked correctly. If execution fails for any reason, 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

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 *Enables the user to inspect the Hadoop cluster load.*

Description

Enables the user 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 the Hadoop engine and sends data to mapper and reducer R functions for execution. If the input data does not reside in HDFS, then hadoop.run first copies the data into HDFS. It prepares the user's mapReduce scripts for execution in the distributed Hadoop environment. It then invokes the Hadoop engine and monitors the Hadoop log for errors and failures. If execution was successful, it 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. The object type may be one of the following: • ORCH HDFS object identifier This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS. It 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 the Hadoop job is submitted otherwise the job fails. If the output directory is not specified a temporary directory is created in HDFS "/tmp".

`mapper` Optional mapper function written in the R language. The function must accept two values: "key" and "value". The names of the arguments do not matter. Prototype is: mapper = function(k,v). If mapper function is not specified or is NULL, then a reduce-only job is executed.

`combiner` Optional combiner function written in the R language. The function must accept two values: "key" and "value". The names of the arguments do not matter. Prototype is: reducer = function(k,v) . The combiner function gets executed on the same host as each mapper. It receives the same data as the reducer, but receives the date from each local individual mapper. Its output is the same as the mapper output and is fed to next reduce stage. Note that combiner is ignored if it is specified without a reducer.

`reducer` Optional reducer function written in the R language. The function must accept two values: "key" and "value". The names of the arguments do not matter. Prototype is: reducer = function(k,v) . If the reducer function is not specified or is NULL, then a map-only job is executed.

`export` Exported R objects. This argument allows the user to copy some client-side R objects (i.e. from user's R session) into the server side running mapReduce R jobs that is to the Hadoop cluster side, so that the objects are 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 to do any initial preparation, initialization, or memory allocation required for map or reduce functions logic. The function does not accept any arguments. Prototype is: init = function().

`final` Optional job finalization function. Called once after all user's mapReduce functions and enables final data processing, or memory de-allocation required by mapReduce logic. It also permits output of keyValues (See orch.keyvals). The function does not accept any arguments. Prototype is: final = function().

`job.name` Optional name of this mapReduce job. By default Hadoop's job ID is the job name. You should always give some meaningful name to make it easier to locate your job in the Hadoop run logs. Note that if this argument is used, it overrides job.name in `config`.

`config` Optional mapReduce advanced configuration class. This argument lets you fine-tune various aspects of the mapReduce job to achieve better performance or to change behavior of the ORCH mapReduce driver. This argument is an instance of the "mapred.config" class, and therefore has this format: config = new("mapred.config", param1, param2,...).

`cleanup` Run a cleanup procedure after the mapReduce job finishes successfully. Removes all empty "part" files and all Hadoop log files.

`overwrite` Overwrites HDFS objects with the same name. By default, overwrite is disabled for data safety.

Value

The results are 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 during execution any error or failure prevents successful output of the result, returns the error or 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

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 the ORCH framework by loading the metadata that describes the contents of the file (number, types and names of data columns in the files in the directory). It does this if the metadata is already present. If not, it discovers the metadata of the file by intelligent sampling of file contents. If successful, the function returns the HDFS object identifier of the HDFS attached directory, 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, header = FALSE)
```

Arguments

`dfs.name`	HDFS directory name or HDFS path relative to the current working directory. Alternatively, you can re-attach an HDFS object identifier returned by hdfs.attach() from a prior invocation (See `force` argument).
`key.sep`	Key field separator character. The character "\t" is system default. If the key separator is specified incorrectly, then key field is concatenated with the first value field and there will be **no** key identified. The key separator can have the same value as value separator.
`value.sep`	Value field separator character. The character "," is system default. If value separator is specified incorrectly then all value fields will be concatenated together. The value separator can have the same value as key separator.
`key`	Key column inde, 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, then this arguments allows you to re-attach it.
`trim`	TRUE to ignore tailing empty fields. If you suspect that the HDFS data has empty tailing columns, such as ",,," then this option can detect and exclude such redundant columns for the data description in metadata and in the data structure.
`data.frame`	If TRUE enforces the class of the attached HDFS data to be "data.frame". Otherwise the class is automatically recognized as "vector", "matrix", or "data.frame".
`na.strings`	Character vector with strings that represent NA values in the attached dataset. If this argument is not specified then "NA" and "" strings are treated as NA values by default.
`silent`	Do not print information messages to the console. Do not print the final attach summary at the end of the run.
`header`	TRUE to indicate the header is present. If the HDFS object contains a header then this option should be passed as TRUE, default value is FALSE.

Details

By default, data files in HDFS are not usable in ORCH until they are attached and until 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 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 creates the HDFS object's metadata with the user specified delimiters stored it. The content of the HDFS object attached is not changed in any way. If you specify incorrect set of delimiters, then the attach may fail. If you do not specify the delimiters then the current defaults ("\t" for key delimiter and "," for values delimiter) are used.

hdfs.attach() creates 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 the HDFS data was attached successfully, otherwise NULL if a 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 needed. Alternatively, you can attach the file for use as input to a Hadoop application.

Due to inherent limitations of the Hadoop command-line interface, the function performance may drop when attaching large HDFS files with long records. When size of one record is larger then 1KB then sampling falls back to streaming larger parts of HDFS files in order to retrieve several full records with valid structure. If the input data contains many invalid or incomplete records, then the function may try to resample larger portions of the input dataset 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 you to fine-tune behavior of the ORCH HDFS cache system. Normally, this function is not necessary because the system is pre-configured with the best 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 cache settings do not change, which allows the user to set fine-tuning options.
disable	Disable caching of one specific HDFS object. This can be an HDFS object identifier or HDFS-compliant path(s) as a character vector. The option must be set when an external change of the HDFS object by another user or third party process is expected. Note that this argument **does not** recursively disables caching of child directories.
enable	Cache a previously disabled HDFS object. This can be an HDFS object identifier or HDFS-compliant path(s) as a character vector. To enable the caching of all HDFS objects specify "*". Note that this argument **does not** recursively enable caching child directories.

`ttl` Sets the Time-To-Live (TTL) configuration parameter of the cache in seconds. Each cached entry can live up to the specified time after which it is automatically deleted. A -1 setting reverts the value to the default.

`ctl` Sets the Clicks-To-Live (CTL) configuration parameter of the cache as the number of access attempts. Each cached entry can be accessed the set number of times. After exceeding this number it is automatically deleted. A -1 setting reverts the value to the default.

Value

Always return the current state of the HDFS cache. If the `onoff` argument is specified, then the function returns 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 the 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 client's OS user name. The 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 "/". It is 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. The absolute path always uses the current ORCH root as a reference point (See hdfs.root).

Details

The ORCH current working directory is similar to the Unix shell notation for a working directory and accepts the path with a number of special symbols. ORCH HDFS path compiler walks through the user's path and denotes each special character into a sub-path converting it into an absolute HDFS path.

Like a Unix shell, ORCH allows three different types of HDFS paths:

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

Like a Unix shell, ORCH allows use of special strings in an HDFS path:

- / Parent and child directory and/or file divider. Directory and file names may not contain "/" symbol.
- . Identifies child directory, must be used as a single token between parent and child dividers (the "/" symbol). Directory and file names can contain the "." 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 a parent directory. This must be used as a single token between parent and child dividers (the "/" symbol). Directory and file names can contain the ".." symbol but in conjunction with other characters only i.e. you cannot 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 is set successfully. NULL is returned if a non-existent 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. The 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. The 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`, the 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 of this function. 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, the 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 than on the unclean version of the same data. 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 location specified by `dfs.dst`, the HDFS destination directory. If the destination directory already exists then the source object is copied there preserving its original name. If the destination directory does not exist then the source file or directory is 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 the 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 the HDFS directory and/or file if it already exists. By default, replacing is disabled.
`force`	Set this argument to TRUE to disable the confirmation prompt when copying a source that contains a wildcard(*) in it. Also disable HDFS I/O check errors, and do not return a result.

Value

TRUE if the file or directory is copied successfully, FALSE if there is an error. In case of failure, the 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 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>". The HDFS user name is the same as the 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 the Unix shell cd/pwd commands design in order to provide familiar forms for navigation and access to 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.pwd hdfs.cd hdfs.root hdfs.ls

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

Description

Returns the currently configured value 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`	Optional. 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`	Optional. 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 the data write to HDFS for each specific object. The key/value separators are stored in the HDFS object's metadata and default system-wide settings are not used when reading this object back from HDFS into 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 the tabulation character "\t" and the values separator is set to the 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 is 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. It represents a directory in HDFS. Alternatively, the user can pass a string with an 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 dataset to the client's R memory **if** the file is small enough or executes a mapReduce job for large datasets. 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 (whose task is 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 is paused 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 files that do not contain data, such as ORCH metadata, Hadoop's system files "_SUCCESS", ".checksum" and other known files that do not contain data are 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. It either represents a directory in HDFS or is a string with an 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 if NULL then all data files from dfs.id directory are downloaded in bulk. The 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 the data structure by embedding ORCH metadata and Hadoop's system data into data files.
overwrite	If TRUE the local files having same names get overwritten. Otherwise an error is reported.

Value

If the operation finished successfully, this is the local file name of the downloaded data. NULL is returned if an error occurred.

Attention

Use this function with caution, since it brings the entire contents of an HDFS directory into your local file system. Since an HDFS directory may store vast amounts of data, you may exhaust your hard drive.

Warning

Specifying download files list in dfs.file argument may significantly downgrade the function performance because each file is then downloaded separately instead of in-bulk directory download.

Note

Data files do not need to be named according to Hadoop's mapReduce convention "part-(m-)?(r-)?[0-9]5" in order to be picked up by the download function. Every file in the directory with a name that does not start 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

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

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 concurrently by several HDFS object identifiers in ORCH. If one of the objects is deleted with hdfs.rm, or it's directory is removed with hdfs.rmdir, or the referred HDFS resource gets (re)moved outside of ORCH by a third party process,then the HDFS object identifiers may become invalid and may refer to non-existing HDFS data. This is one example of a situation where it is necessary to pre-check the validity of the HDFS object using hdfs.exists.

Value

TRUE if data exists and valid, FALSE if data does not exists or if there is 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 an Apache Hive or Apache Impala table to a dfs identifier in ORCH.*

Description

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

Usage

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

Arguments

`table`	An ore.frame object or a character string representing an Apache Hive or an Apache Impala 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 Apache Hive or Apache Impala 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 Apache Hive and Apache Impala tables are supported for conversion. Partitioned tables are stored as a collection of sub-directories which does not correspond to the ORCH data storage model. Therefore, using a partitioned table as input would result in an error.

Value

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

Attention

An Apache Hive or Apache Impala staging table is created if the `table` object does not represent a physical table (e.g. transformed ore.frames, views, etcetera). The user can optionally pass in a name using `outtabname` for the staging table (if created) to be used as an ORE-HIVE or ORE-IMPALA 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, which was already attached to ORCH (see hdfs.attach) and outputs the same data contained in the binary format, but in 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. It can instead 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 directory is created in HDFS "/tmp".

`overwrite`	Allows overwriting of the output HDFS directory if it already exists with the same name. By default overwrite is disabled for the 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 parts is not specified, the function relies on Hadoop's default behavior and will generate a part file for 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 is used. If the original separator is not available, then the default ORCH global key separator is 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 is used. If the original separator is not available, then the default ORCH global value separator is used ("," by default).
`silent`	Do not print information messages to console. Do not print the 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 occurs).

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 an R in-memory object. Reads ORCH metadata with all meta files (such as levels data) and restores all attributes, including column names, data types, row names, factor levels, etcetera. If the data originated from the R environment, i.e., data was put in HDFS using hdfs.put, then these attributes are available. Otherwise, if data originated from another source and was automatically attached via hdfs.attach, then generic reverse-engineered object attributes like "val1", "val2" for columns names and default data type "data.frame" are assigned. Users can also update the metadata using hdfs.meta in order 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. It either represents a directory in HDFS, or, is a string with an 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 dataset, or NULL if the operation has failed.

Note

If the HDFS file contents can comfortably fit into an in-memory R data frame object, then use hdfs.get(). Otherwise you must fetch the HDFS files into local file system and then read 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 is 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 the HDFS object contains many part-files, then the function retrieves the head portion or the whole file from the first (lexicographically sorted by name) part-file. If the number of lines retrieved is less than `n` then head portion of the next file is appended.

Usage

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

Arguments

`dfs.id`	HDFS object identifier that indicates where to get heading data. This is a special ORCH object returned by hdfs.attach or other functions accessing HDFS. This object either represents a directory in HDFS, or, is a string whose value is an 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 returns a default head portion of the first part-file which will give the fastest possible execution time. The default size is the whole part file if small enough (<=100KB) or, an arbitrary head portion if it's too large (>100KB).

Details

Function performance degradation is a result of two factors - the number of part files in the input HDFS directory (e.g. HDFS object) and the size of each part file. Performance degradation is approximately linear, with the increase in the number of HDFS data files and the size increase of each data file. However, after reaching approximately 100KB, further file size increase will not significantly affect the runtime.

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 for behavior that is close to that of the Unix shell "head" utility, but inherits the limitations 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 the HDFS object identifier is provided, then the function merely checks for its existence (if `force` is FALSE).
`absolute`	TRUE if `dfs.x` is an absolute HDF path and must 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`	When TRUE, do not perform existence check. Default is FALSE (perform existence check).

Details

This function is equivalent to hdfs.attach, but does no metadata discovery or generation if the HDFS directory has never been attached before. It also allows you to create identifier for an (as yet) non-existent HDFS object.

Value

ORCH HDFS object identifier which points to an HDFS object if there are no errors. Returns NULL if `dfs.x` does not contain a valid HDFS path, or, if 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 a single character. If not specified then the function returns 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 the default system-wide value when this object is read back from HDFS in to ORCH. This system-wide default value is employed only when the 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 the definition of the distinct levels of each categorical/factor column in the HDFS object. The levels metadata is stored separately in a metadata file other than the main metadata file (which is stored in __ORCHMETA__). This is to minimize the file 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 or other functions manipulating HDFS. it can represent a directory in HDFS. Alternatively, the user can pass a string with an 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, then an attempt to write it again will fail. Setting this parameter to TRUE allows overwriting the existing levels.

Details

This function allows the user to read or write ORCH levels metadata for an HDFS bject 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 used within (and can be used only within) this column. At the same time, a 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 the 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 the function returns only its value without wrapping the value into a list. Otherwise if all levels were written successfully, then the function returns TRUE. It returns FALSE if any level write has failed. See examples.

Note

If column(s) levels are specified as "name=levels" in [...] parameters, then the function writes given levels into an 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 the function reads them from HDFS and returns a list of attached levels.

The "..." parameter can be specified using a vector or CSV string. In all cases, it is an instruction 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. If needed, the list can specify the HDFS path to a directory. The function will list data and system files without any differentiation. This function is equivalent to the "hadoop fs -ls" shell command.

Usage

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

Arguments

`dfs.path`	Optional. The path is relative to the current working path or absolute HDFS-compliant path. If not specified, then the list of all objects at the current working path is returned. See hdfs.cd for more details about HDFS path specification.
`pattern`	Optional. A regular expression pattern 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 is returned in the 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, silent = FALSE)
```

Arguments

`dfs.id`	HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS. It represents a directory in HDFS. Alternatively, the user can pass a string with an 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`	if TRUE do not check for invalid or unknown attributes. This allows the user 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 the function returns only its unlisted value. Otherwise, it returns TRUE if all attributes were set. It returns FALSE if an attribute was not set. 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.

The "..." parameter can be specified using a vector or CSV string. In all cases it means "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. The newly created directory is empty. This function is equivalent to the "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. The default value is FALSE.
`cd`	If TRUE then automatically sets the newly created directory as the current working directory. The default value is FALSE. See hdfs.cd for more information.

Value

A new HDFS directory absolute path as a string, or, NULL if the new directory was not created.

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 HDFS directory specified by `dfs.dst`. If the destination directory already exists, then the source object is moved there, preserving its original name. If the destination directory does not exist then the source file or directory is renamed and optionally moved there. This function is equivalent to the "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 it already exists. By default overwriting is disabled.
`force`	If TRUE, then disable confirmation of '*' moving, does not perform HDFS I/O check errors, and do not return the 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. It either represents a directory in HDFS, or, is a string with an 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., as an unattended execution. The `force` argument implicitly enables silent execution.

Value

Number of columns as an integer value. If the value is unknown and can not be computed, NA is 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

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. It either represents a directory in HDFS, or, is a string with an 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., as an unattended execution. The `force` argument implicitly enables silent execution.

Value

Number of rows as an integer value. If the value is unknown and can not be computed, NA is 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

hdfs.dim hdfs.ncol hdfs.meta

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

Description

Lists and returns the number of partitions in the HDFS object denoted by `dfs.id`. 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. It represents a directory in HDFS. Alternatively it can be a string with an 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. The default is FALSE.
`nonzero`	Count in only non-empty data files. The default is FALSE.

Details

If the HDFS object has no data files then 0 is returned. This indicates that the object exists in the HDFS file system but its directory is empty. If an HDFS object does not exist, NULL is returned 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 (is empty) the 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

The input object is a HDFS object identifier. The function returns the name of a new table containing loaded data from HDFS. The name of the table is the same as the name of the HDFS object's directory unless redefined by the `db.name` argument. The data is pulled by underlying drivers (see details about `driver` argument) and starts a number of mapreduce jobs. These jobs 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. The HDFS path exported to the database can be specified as either an absolute path or as a path relative (to the current work directory in HDFS.)
`db.name`	Optional database table name. If not specified, then the HDFS object name (its HDFS directory name) is used as a target database table name
`overwrite`	If TRUE, removes an existing database table, otherwise if the table already exists, the export fails with an error. The default is FALSE.
`sep`	Optional HDFS value fields separator. Use this argument only when exporting an HDFS directory that was never attached and does not have attached ORAAH metadata.
`driver`	Choose the RDBMS to HDFS data transfer driver. the default is selected when an RDBMS connection is established via orch.connect. You can choose a different driver for the data transfer. Available drivers are: "sqoop", "olh".

Details

If orch.connect was invoked in secure mode, then, this API prompts the user to enter the 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 batch mode as well. If orch.connect is 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 is destroyed automatically once Sqoop/OLH has read it. This is possible because the configuration file is a temporarily-unlinked file.

Value

Exported database table name that can be used in ore.sync to attach the table to the Oracle R Enterprise framework. NULL is returned if any errors are encountered.

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 fail, preventing any attempts at 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.The function 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). This starts a number of mapReduce jobs that will pull data out of the database in parallel and store it into a set of files in the 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). This can also be 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 is generated meaning that the key column will contain "" strings.
dfs.name	Optional custom name to assign the imported HDFS object. If not specified then a temporary HDFS object will be generated. This object is 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 the RDBMS to HDFS data transfer driver, the default one is selected when an RDBMS connection is established via orch.connect. This allows you to use a different driver for the data transfer. Available drivers are: "sqoop" and "olh".

Details

If orch.connect is invoked in secure mode, then this API prompts the user to enter the database password. The password is held encrypted in memory and transferred to an on-disk configuration file for use by Sqoop or another transport driver. This is also, in general, the way Sqoop is invoked in a batch mode. If orch.connect was invoked in [secure] = FALSE mode, then the password entered earlier would have been kept encrypted in memory and transferred to Sqoop configuration file on disk. The configuration file is destroyed automatically once Sqoop has read it. The configuration file is a temporary-unlinked file on Linux.

Value

HDFS object identifier if data was successfully exported or NULL if a 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 the HDFS file system. All data attributes such as column names, data types, etcetera are 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 the 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 is 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 the data itself. For more details see hdfs.levels.
`key.sep`	Key field separator character, The, "\t" character is the default. For uniform separators, set it to the same value as `value.sep`. The 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 occurs.

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>". The HDFS user name is the same as the client's 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 the design of the Unix shell, cd and pwd commands 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 are 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 an HDFS object identifier to be deleted.
force	If TRUE, disables confirmation of '*' deletion, does not run HDFS I/O check errors, and does not return the 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. Setting the argument to TRUE if an object is 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, the 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, 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.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 invalidates all HDFS object identifiers pointing to this HDFS data folder. Any ORCH operations using these invalid identifiers will result in failures. This function is equivalent to the "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. It either represents a directory in HDFS or, is a string with an HDFS-compliant directory path relative to the current working directory.
`force`	Set this argument to TRUE to disable confirmation of '*' deletion, do not perform 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. Setting it to TRUE is required if an object is 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 the event of a 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. If Hadoop is running in standalone mode (normally used to setup a test environment), this function can be used to map HDFS into one of the local folders.

Usage

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

Arguments

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

Details

Any absolute HDFS path in ORCH is always relative to the current HDFS root, For example, ORCH path "/a/b" when HDFS root in ORCH is set to "/tmp/hdfs". This actually results in accessing "/tmp/hdfs/a/b", the absolute path in HDFS. The user is not allowed access to any location above the HDFS root path. ORCH will error out if such attempt is detected.

Upon startup, ORCH sets HDFS root to "/" if Hadoop is running in distributed or pseudo-distributed mode. If hadoop is running in standalone mode, the function sets HDFS root to "/tmp/hdfs".

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, etcetra, are 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 (possibly 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. It either represents a directory in HDFS or, is 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. Specifying n=0 will return 0 records and should be used to retrieve data structure attributes, such as the columns names of a data.frame. Specifying a negative value means "at least", For example, n=-1000L will try to retrive 1000 records or more.
level	The number of HDFS part files to sample. Higher numbers assures better and closer to normal distribution, but this 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.

A Key/value separator is not required for this function, because it is stored alongside the data itself and is retrieved automatically from its ORCH metadata. This also means that the 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 dataset, or NULL if the operation fails.

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 the ORCH infrastructure are relative to the current HDFS root and the 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 the HDFS file system to be set as current HDFS root. If this argument is not provided 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 the dfs.path path is invalid or does not exist in the 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 the total size of all its data files in bytes or in human-readable form if the `units` argument is specified. Non-existent 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. It either represents a directory in HDFS or, is a string with an HDFS-compliant directory path relative to the current working directory.
`units`	If specified then the output value is converted into a human-readable form. The `units` argument can have any of the following values: "KB", "MB", "GB", "TB", or "PB".

Value

Total size of the HDFS object in "unit" bytes, or 0, if the object exists in HDFS but does not have any data. 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 the Hadoop HDFS file system.*

Description

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

Usage

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

Arguments

`dfs.id`	HDFS object identifier with which the cache must be synchronized. If this argument is not specified, then the entire HDFS cache is reset. This is a special ORCH object that represents either a directory in HDFS, or a string with an HDFS-compliant path relative to the current working directory.

Details

Currently, 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 third 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 it, without applying parsing or formatting. Due to HDFS design restrictions, the tail is concatenated from the tails of each part file of the HDFS object, not the real n last lines of an HDFS file. This function is euiqvalent to "hadoop fs -tail" shell command.

Usage

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

Arguments

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

Details

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

Two factors can degrade performance - the number of part files in the input HDFS directory (e.g. HDFS object) and the size of each part file. Performance approximately linearly degrades with the increase of number of HDFS data files and with the size increase of each data file. The cutoff is approximately 100KB. After this point, further file size increase does not significantly change runtime.

Value

Character vector of the specified length `n`. The length can be less than `n` if the specified number of lines cannot be retrieved. If the HDFS directory has no non-empty data files then a 0-size character vector is returned. NULL is returned if the HDFS object's directory does not exist or if 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 Apache Hive or Apache Impala table represented by ORE's ore.frame object.*

Description

This function converts an HDFS object identifier in ORCH to a Apache Hive or Apache Impala 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. It either represents a directory in HDFS or can be a string with an HDFS-compliant path relative to the current working directory.
`table`	A character string representing the target Apache Hive or Apache Impala table name. If `table` is NULL (default), a table with a temporary name is created. The table 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 across sessions, a non-NULL `table` argument must be passed.

Value

Returns the ore.frame object representing the Apache Hive or Apache Impala table.

Attention

ORE-HIVE supports factor types within R but, in Apache Hive or Apache Impala, the factor columns, are of the "string" type. If the input has one or more "factor" columns, they will be automatically changed to "character" type without changing any values. In order to preserve original values, the user needs to de-factorize the input data first converting integer values to strings before calling hdfs.toHive. 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 Apache Hive tables because they use uniform delimiters only. User must convert the dataset into a 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). It outputs the same data, but in the ORCH-specific 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. It either represents a directory in HDFS or can be a string with an 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 directory is created in HDFS "/tmp".
`overwrite`	Allows overwriting the output HDFS directory if it already exists under the same name. By default, overwrite is disabled for safety of data manipulations.
`parts`	The number of desired output partition files. This option directly controls the size of each "part" file, which approximately equals the total output size/number of "part" files. The function attempts to satisfy the specified requirement, but with no guarantee, because of Hadoop jobs execution restrictions. If this argument is not specified, the function relies on Hadoop's default behavior and either generates a part file per each input part file or an HDFS split, whichever is of lesser size.
`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, thereby 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 the user attempts to change the splitting or other binary data

Details

The binary format is readable by ORCH mapReduce R jobs only and gives the advantage of the 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. Splitting "part" HDFS files limits memory usage and improves 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 occurs.

Note

Output of the function is always pristine. If input HDFS data is not pristine the function will remove all unclean and invalid rows from the dataset 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 dataset in HDFS. The HDFS object 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. 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 previously, it is 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, then the target HDFS directory receives a unique ID and the HDFS file(s) are named as "part-12345". If any of the uploaded local files are larger than the `split.size` argument (in bytes), then the file automatically 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 a directory is specified then all files in this directory are uploaded into HDFS. You can mix file and directory names.
`dfs.id`	Name of the target HDFS directory, or the HDFS path relative to the current working directory, or an HDFS object identifier. If the directory does not exist in HDFS it is created. If it exists, then the `overwrite` parameter must be considered.
`dfs.file`	Vector of strings that specifies the desired names of files uploaded to HDFS. Its length must either be the same as number of files to be uploaded into HDFS or 1. If 1, the string is used as a prefix for every HDFS file name. If not specified or NULL, then the HDFS file is in the form "part-12345".
`header`	TRUE if local files have a header in the first line which should be removed before uploading to HDFS. You can also specify the number of rows to remove by assigning a numeric value to this argument.
`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 copies the local files as-is and automatically creates its metadata with the specified delimiters. The content of the file copied into HDFS will not change. If you specify an incorrect set of delimiters, then the attach of the copied data fails. If you do not specify the delimiters, then the current ORCH defaults are used.

Value

HDFS object identifier of the loaded data if attached. The HDFS absolute path to the uploaded data if it is not attached. NULL if an error occurs.

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 value or sets a new value for 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	Optional. A new value separator value to set. Must be single character only. If not specified then the function returns the value that is currently set.

Details

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

---

hdfs.write *Writes a dataframe as a CSV file.*

Description

This function is used to write a Spark dataframe 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.neural2
- 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
- orch.ml.gbt
- orch.elm
- orch.helm

Written data in HDFS will preserve all metadata required for retreiving it later from within 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.

Attention

If your Spark Dataframe has non-atomic columns (Vector type), once written using hdfs.write cannot be read using hdfs.get. However, they can be read forcefully but altering the meta data using hdfs.meta.

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

```
data <- hdfs.put(iris)
lm_model <- orch.lm2(Petal.Length ~ Sepal.Length + Petal.Width, data = data, verbose = FA
pred <- predict(lm_model, data, supplemental=c("Sepal.Length", "Petal.Width"), verbose =
dfs.id <- hdfs.write(pred, outPath = "destination", overwrite = TRUE)
head(hdfs.get(dfs.id))
hdfs.rm(dfs.id)
```

---

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

Description

Verifies if the R object specified by x contains an ORCH type HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS. It 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 an "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 an ORCH type HDFS object identifier. This is a special ORCH object returned by hdfs.attach and other functions accessing HDFS. It 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 a Spark session and contains a corresponding Spark 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 for adjusting and fine-tuning a mapReduce job launched with hadoop.run or hadoop.exec ORCH functions. These are useful for cases where the out-of-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 is used. Tip: assign a meaningful name. This will help you to locate your job in the Hadoop execution logs if needed.

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

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

`min.split.size`**:** Changes HDFS split size which is by default equal to the HDFS block size (typically 64MB). Split size indirectly controls the number of mappers and reducers launched by Hadoop, because 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 is force killed by Hadoop. The dfault value is 600 seconds. This option sets Hadoop's property "mapred.task.timeout".

`skip.na.recs`**:** This option enables a cleanup procedure in the ORCH driver. Any input record containing NA in any of its fields is removed and the mapper and reducer user's function receives 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 dataset is copied into HDFS one of its data columns can be used as a key. However, in this case, the values provided to a mapper function do not have key values. If the user's code expects the original data structure with key column present in values, then this option should be enabled.

`map.filter`**:** Works in conjunction with `map.valkey` and indicates to the ORCH driver that the mapper output should have exactly the same structure as the input provided to it. The only operation it performs is filtering of some of the records. If `map.valkey` is enabled and keys are inserted in values, then these are automatically removed from the mapper output.

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

`reduce.filter`**:** Works in conjuction with `reduce.valkey` and indicates to the ORCH driver that the reducer output should have exactly the same structure as the input provided to it. The only operation it performs is filtering of some of the records. If `reduce.valkey` was enabled and key was inserted in values, then filter automatically removes it 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 are converted into "character" data type. Otherwise, the usual coercion hierarchy (logical < integer < double < complex) is used, i.e., all-logical data frames will be coerced to a logical matrix, mixed logical-integer will give an integer matrix, etcetera.
- "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. The user's mapper function can output via orch.keyvals or orch.keyval an arbitrary data structure. For ORCH to correctly configure the 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 assumes that mapper output has the same structure as input data. A 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 the mapper output will be key-less, but it 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 one time to a mapper. In order to limit memory usage and prevent R running out of memory, the user can set an upper limit to the number of rows that the ORCH driver can supply to a user's mapper function at one time. The last invocation may have fewer rows due to split boundary. Values accepted:

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

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

`reduce.input`**:** R data type name expected by user's reduce function. It 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.The user's reducer function can output (via orch.keyvals or orch.keyval) an arbitrary data structure. For

ORCH to correctly configure the Hadoop job and store the ORCH metadata alongside the output HDFS dataset it must know the structure up front. If `reduce.output` is not specified then ORCH will samples the output data and automatically attaches it, which results in the generation of the ORCH metadata. For more details, see the `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 one time 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 that the ORCH driver can supply to the user's reducer function at one time. The last invocation may have fewer rows due to the split boundary limitation. Note that if there are more values with the same key than the `reduce.split` limit, then key block is split into parts and the reducer function must correctly handle duplicated key blocks. For more detail see the `map.split` definition.

`reduce.eos`**:** Send the End Of Stream (EOS) signal to a user's reducer function after all data is streamed in and given to the function. The very last invocation of the reducer will be with NULL key and NULL values, indicating the 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 the reduce code to deal with chunks of rows at a time, with an EOS flagging the 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 perform any other HDFS file system manipulation normally available to a user in an ORCH client, but inside of a server-side mapper and reducer user's function. By default, the current working HDFS directory in every mapper and reducer is set to the same path as it is in the ORCH client at the time that the Hadoop job is launched.

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

`output.pristine`**:** Tells ORCH that output of the mapReduce job is expected to be "pristine". The definition of "pristine" data is that when every character value in each field stored in HDFS is converted to its column data type as specified in ORCH metadata. It does not produce an `NA` result except for the special values "NA" and "". For example, if a column type is "numeric" and if there is any empty value or a value not convertible to a numeric form in its data, then the entire dataset is not considered "pristine". Having the dataset in "pristine" mode greatly improves data read and parse performance in ORCH mapReduce jobs. But specifying a non-conforming dataset as "pristine" results in Hadoop job execution failure. If this is a map-only job, then the user's mapper function output is considered "pristine", otherwise, the user's reducer output is marked as pristine.

`output.key.sep`**:** Output key field separator character. The "\t" character is the default. For uniformity in 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 it is applied to the user's reducer output.

`output.value.sep`**:** Output value fields separator character. The "," character is the default . For uniformity in 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 applies to the user's mapper function output, else it applies to the user's reducer output.

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

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

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

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

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

`queue`: Name of queue where mapReduce job will be queued. If not specified the job will be queued to default mapReduce Job queue.

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_CLASSPATH *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 set the CLASSPATH used by ORAAH client's Java Virtual Machine (JVM). Setting this environment variable overrides the default CLASSPATH environment value. So, if both ORCH_CLASSPATH and CLASSPATH environment variables are set, then ORAAH prioritize use of ORCH_CLASSPATH. Also, Wildcard characters are supported. For example, having a path `/usr/lib/hadoop/lib/*.jar` in ORCH_CLASSPATH or CLASSPATH will add all jars from `/usr/lib/hadoop/lib` to rJava JVM's CLASSPATH.

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-envvar ORCH_JAVA_XMX

Examples

```
## Not run:
 csh: setenv ORCH_CLASSPATH "/usr/lib/hadoop/lib/*.jar:/usr/lib/spark/*.jar"
 bash: export ORCH_CLASSPATH="/usr/lib/hadoop/lib/*.jar:/usr/lib/spark/*.jar"

## End(Not run)
```

orch.connected *Checks if ORAAH is connected to Oracle Database.*

Description

Checks if ORAAH is connected to Oracle Database.

Usage

```
orch.connected()
```

Value

`TRUE` if ORAAH 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.reconnect` `orch.dbinfo`

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

Description

This function connects ORAAH with an instance of Oracle Database. All following database import and export operation are performed using this connection. After the connection is established it is validated by reading the USER variable from the database. It displays connection attributes and error messages if a problem is detected. If the user password is not supplied , it prompts for the password 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, pdb = NULL, 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 running Oracle Database in non-CDB mode.
pdb	Oracle Pluggable Database's service name that uniquely identifies a particular PDB in the CDB database on a system. If [pdb] is not specified, then the Oracle database is considered to be running in non-CDB mode and the [sid] is used for database connection.
host	The host name or IP address of the database server that is the target of the connection.
password	The database password (optional). If not specified, then the user is prompted to enter the password.
port	The database server connection port. The default is 1521.
secure	Chooses ORAAH to Database connection mode. In secure mode, ORAAH does not store the password and the user is prompted for the password at each attempt to access the database. The default setting is TRUE.
driver	Specifies the database to HDFS data transfer driver. The "sqoop" driver is the default. Available drivers are: "sqoop", "olh".
silent	If TRUE, does not print connection information to the R console. Otherwise, the user, host, port and SID/PDB of the established connection are displayed. The default setting is FALSE.
dbcon	Provides an alternative way to specify all connection parameters as a single orch.dbcon object. See orch.dbcon for more details.

Details

By default, [secure] is set to TRUE which means that the user is always prompted for the password. In the secure mode, the password is requested for every attempt to connect to a database. The [secure] = FALSE mode is intended for testing purposes. In this mode, the password is encrypted in memory and subsequent APIs that require this password will **not** prompt the user to enter password each time. Be sure to set secure to TRUE in production environments.

If there is a connection failure or any other errors, the connection is rolled back to the connection established prior to calling this function.

Value

TRUE if the connection was successfully established and validated. FALSE if the 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. An 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 a 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 dataset with specific data characteristics for testing of the ORCH functionality as well as 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 dataset using the other parameters. The output dataset 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 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 may be different.
`row.pattern`	This argument can be used in conjunction with `numeric.col.count`, `integer.col.count`, `factor.col.count`, and `character.col.count` to specify the order of the columns. It can also be used by itself, which automatically sets number of columns of each type. This is 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 is generated with a number of distinct integer values specified by this parameter.
`key.sep`	Key field separator character. The default system-wide value is used if this argument is not specified (normally "\t").

value.sep	Value field separator character. The default system-wide value is used if this argument is not specified (normally ",").
out.name	HDFS directory name or an HDFS object identifier of the output data. Note that the output directory must not already exist when the Hadoop job is submitted, otherwise the job fails. If the output directory is not specified, a temporary directory is created in HDFS "/tmp".
overwrite	Allows overwriting of HDFS objects with the same name. By default, overwrite is disabled for safety of data. The default is FALSE.
task.timeout	Maximum time in seconds a map or reduce task is allowed to run before it is force killed by Hadoop. The default value is 600 seconds.

Details

The number of records in the generated dataset 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 the 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 1to factor.levels. Integer values are generated using the sample function with integer.sample.size parameter, and are 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 dataset approximates the input parameters 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 if any other error occurs.

---

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, compact way for the user to switch among several databases without entering credentials each time there is a attemp to re-establish the connection. The current database connection object can be retrieived using the function orch.dbcon. This object can be reused to reconnect to the database with the orch.[re]connect() function. If there is no connection to the database, then orch.dbcon() returns an empty dbcon object.

Returns current Database connection object.

Usage

```
orch.dbcon()
```

Value

Current Database connection object. The object can be used to connect to a database once again with `orch.reconnect` function. If the database is not connected then returns an empty `orch.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 or if connecting to an Oracle Database running in CDB mode.

`pdb`: Oracle Pluggable Database's service name that uniquely identifies a particular PDB in a CDB database on a system. If `pdb` is not specified, the Oracle database is considered to be running in non-CDB mode and `sid` is used for database connection.

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

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

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

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

Author(s)

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

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

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.connect` `orch.reconnect`

`orch.dbg.lasterr` *Returns the very last error message reported. Messages considered to be errors are those of severity level ERROR, CRITICAL, or FATAL.*

Description

Returns the very last error message reported. Messages considered to be errors are those of 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 this returns as an invisible value.

`orch.dbg.off` *Globally disables debugging in the ORCH framework.*

Description

Globally disables debugging in the ORCH framework. The `severity` parameter turns off individual message severity logging only. This option 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 messages until it is resumed with an orch.dbg.on() function call. Turns off asserts also. For example: orch.dbg.off(). • vector – List of severities to disable individually. Only those severities are not logged. Asserts are still enabled. For example: orch.dbg.off(c("info","trace")). • string – Comma-separated list of severity names to disable individually. Only those severities listed are not logged. Asserts will be still enabled. For example: orch.dbg.off("info,trace"). • "all" – Completely turns off debugging and resets 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 or severities are turned off or set to "all" the assert settings do not change. If debugging is disabled globally, then asserts are also disabled.

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. The `severity` argument lets the user sets a 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 as a 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" – 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 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 is not changed and only additional list severities are enabled individually. For example: orch.dbg.on(",trace")
- "~" – If the first value is "only" or "~" then only listed severities are enabled and the global severity level is 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 an individual severity or set of severities are turned off or "all" the assert settings do not change. If debugging is enabled globally, then asserts are also enabled.

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

Lets the user set a new ORCH debug log output stream or a file name. If the new output is not specified, then it is set to `stdout` by default. Upon ORCH startup, the 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. This also can also be a file name as a string, or `stdout()`, or `stderr()`. "" is considered 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 the `dbcon` argument is specified). This is informational only. 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. If not specified, then the currently 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 a mapReduce job run within the same R session from which the job was submitted. The user can set debug breakpoints in their mapper, reducer, combiner, or in any function that was exported into the ORCH mapReduce job environment via the export argument of the hadoop.run function. When "debug" mode is enabled, ORCH prepares the mapReduce driver script as always, but instead of submitting scripts to a Hadoop cluster, it load scripts locally and runs its own local implementation of the Hadoop pipeline, invoking the local user's functions.

Usage

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

Arguments

onoff	TRUE, to enable the "debug" mode. FALSE, to disable the "debug" mode. If not specified then only the current setting for the "debug" mode is returned.

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 employ built-in R debug tools or any third party debug library of their choice.

Value

Current "debug" mode. If the onoff argument is not specified then, this option returns the value visibly. Otherwise, the value is returned 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.destroyConf *Removes stored values for startup checks*

Description

Removes the temp file with stored values for checks

Usage

```
orch.destroyConf()
```

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.reconf

orch.df.collect *Collects a Spark data frame to client's memory, and returns an R data frame.*

Description

Collects a Spark data frame to client's memory, and returns an R data frame.

Usage

```
orch.df.collect(data)
```

Arguments

data	Spark DataFrame.

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

```
iris_hdfs <- hdfs.put(iris)
iris_df <- orch.df.fromCSV(csvPath = iris_hdfs)
iris_desc <- orch.df.describe(iris_df)
desc <- orch.df.collect(iris_desc)
print(desc)
hdfs.rm(iris_hdfs)
```

orch.df.createView *Creates or replaces a temporary Spark SQL view.*

Description

Creating a Spark SQL view is needed is you wish to run Spark SQL query on an exisiting Spark data frame. This functions registers a Spark data frame as a SQL view. The SQL queries can be submitted using orch.df.sql.

Usage

```
orch.df.createView(data, viewName)
```

Arguments

data	Spark data frame.
viewName	Spark SQL view 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

```
iris_hdfs <- hdfs.put(iris)
iris_df <- orch.df.fromCSV(csvPath = iris_hdfs)
orch.df.createView(iris_df, "iris_view")
sql_df <- orch.df.sql("select Petal_Length, Sepal_Width from iris_view where Petal_Length
sql_df$show()
hdfs.rm(iris_hdfs)
```

---

orch.df.describe — *Computes and returns statistics for numeric columns. If no columns are given, this function computes statistics for all numerical columns.*

Description

Computes and returns statistics for numeric columns. If no columns are given, this function computes statistics for all numerical columns.

Usage

```
orch.df.describe(data, columnList = NULL)
```

Arguments

columns	List of column names.

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

```
iris_hdfs <- hdfs.put(iris)
iris_df <- orch.df.fromCSV(csvPath = iris_hdfs)
iris_desc <- orch.df.describe(iris_df)
iris_desc$show()
hdfs.rm(iris_hdfs)
```

---

orch.df.fromCSV *Creates a Spark data frame from comma-separated values data source.*

Description

Creates a Spark data frame from comma-separated values data source.

Usage

```
orch.df.fromCSV(csvPath, minPartitions = -1L,
  headerPresent = TRUE, fieldSeparator = ",",
  quote = "\"", na = "NA", verbose = TRUE)
```

Arguments

csvPath	Any Hadoop-supported file system URI. For instance an HDFS directory, or a local file system (if local, then it must be available on all nodes and specified using file://<file_path>).
minPartitions	Suggested minimum number of partitions. If minPartitions <= 0, the default will be used.
headerPresent	Whether each file contains the names of the variables as their first line.
fieldSeparator	CSV field separator character.
quote	CSV quotation mark (most often it is the double quotation mark).
na	Missing value representation. For instance, "NA" (Not Available).
verbose	Whether to report some performance statistics.

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

```
iris_hdfs <- hdfs.put(iris)
iris_df <- orch.df.fromCSV(csvPath = iris_hdfs)
iris_df$show(5L)
iris_df$printSchema()
hdfs.rm(iris_hdfs)
```

---

orch.df.persist *Persists Spark data frame.*

Description

Persists Spark data frame.

Usage

```
orch.df.persist(data, storageLevel, verbose = TRUE)
```

Arguments

data	Spark data frame.
storageLevel	The desired storage level. The vaild choices are "NONE", "DISK_ONLY", "DISK_ONLY_2", "MEMORY_ONLY", "MEMORY_ONLY_2", "MEMORY_ONLY_SER", "MEMORY_ONLY_SER_2", "MEMORY_AND_DISK", "MEMORY_AND_DISK_2", "MEMORY_AND_DISK_SER", "MEMORY_AND_DISK_SER_2", "OFF_HEAP". Check Spark documentation for more information on Storage Level differences.
verbose	Show the description of the resultant storage.

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

```
iris_hdfs <- hdfs.put(iris)
iris_df <- orch.df.fromCSV(csvPath = iris_hdfs)
orch.df.persist(iris_df, "DISK_ONLY")
hdfs.rm(iris_hdfs)
```

`orch.df.scale` *Scale numerical columns of a data frame.*

Description

Scale numerical columns of a data frame.

Usage

```
orch.df.scale(data, method)
```

Arguments

`data`	Input Spark data frame.
`method`	Scaling technique

- "standardization" $\frac{x-mean}{sd}$
- "unitization" $\frac{x-mean}{range}$
- "unitization_zero_minimum" $\frac{x-min}{range}$
- "normalization" normalization with zero being the central point $\frac{x-midrange}{range/2}$
- "normalization_2" normalization in range [-1, 1] $\frac{x-mean}{max(abs(x-mean))}$
- "normalization_3" $\frac{x-mean}{sqrt(sum((x-mean)^2))}$
- "quotient_sd" $\frac{x}{sd}$
- "quotient_range" $\frac{x}{range}$
- "quotient_max" $\frac{x}{max}$
- "quotient_mean" $\frac{x}{mean}$
- "quotient_sum" $\frac{x}{sum}$
- "quotient_sqrt_ssq" $\frac{x}{sqrt(sum(x^2))}$

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

```
iris_hdfs <- hdfs.put(iris)
iris_df <- orch.df.fromCSV(csvPath = iris_hdfs)
scaled_iris_df <- orch.df.scale(iris_df, method = "quotient_max")
scaled_iris_df$show(5L)
hdfs.rm(iris_hdfs)
```

orch.df.sql *Executes a Spark SQL query.*

Description

This function is used to run an Apache Spark SQL query on a Spark SQL view created using orch.df.createView. The results of the SQL query can then be collected in R session using orch.df.collect.

Usage

```
orch.df.sql(query)
```

Arguments

query	Spark SQL query to submit.

Value

Returns the results of the Spark SQL query as a Spark data frame.

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

```
iris_hdfs <- hdfs.put(iris)
iris_df <- orch.df.fromCSV(csvPath = iris_hdfs)
orch.df.createView(iris_df, "iris_view")
sql_df <- orch.df.sql("select Petal_Length, Sepal_Width from iris_view where Petal_Length
sql_df$show()
hdfs.rm(iris_hdfs)
```

orch.df.summary *Creates and returns a summary Spark data frame.*

Description

Creates and returns a summary Spark data frame.

Usage

```
orch.df.summary(data, verbose = TRUE)
```

Arguments

data	Input Spark data frame.
verbose	Whether to report progress. Default value is 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

Examples

```
iris_hdfs <- hdfs.put(iris)
iris_df <- orch.df.fromCSV(csvPath = iris_hdfs)
iris_summ <- orch.df.summary(iris_df)
iris_summ$show()
hdfs.rm(iris_hdfs)
```

orch.df.unpersist *Unpersists Spark data frame.*

Description

Unpersists Spark data frame.

Usage

```
orch.df.unpersist(data, storageLevel)
```

Arguments

data	Spark data frame.

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

```
iris_hdfs <- hdfs.put(iris)
iris_df <- orch.df.fromCSV(csvPath = iris_hdfs)
orch.df.persist(iris_df, "DISK_ONLY")
orch.df.unpersist(iris_df)
hdfs.rm(iris_hdfs)
```

---

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 error out upon attempt to communicate with the database, since the connection is broken.

Usage

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

Arguments

silent	Do not print connection status messages to the R console. The default setting is FALSE.
dbcon	Return current database connection object orch.dbcon after the disconnect is performed. This allows you to re-establish the same connection using the same connection object.

Value

Can return two types, depending on the dbcon argument value:

- If the dbcon argument is set to TRUE, returns the previous database connection object of class orch.dbcon. The object can be used to reconnect to the database with the orch.reconnect function. If the previous connection database is already disconnected, then the return value is NULL.
- If the dbcon argument is set to FALSE, then returns TRUE if the connection was successfully dropped, or, FALSE if the database connectionis already terminated.

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 puts the input data into a local temprorary directory and generate a csh shell compliant command line that simulates the execution of the scripts in the 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 specified then only the current setting for the "dry run" mode is returned.
direct.io	Use direct local file system read/write IO in the ORCH driver instead of streaming data in/out via OS stdin/stdout. This eliminates streaming overhead for benchmarking.

Details

If there are any failures, the shell command line can be retrieved from the ORCH debug log and used standalone outside of ORCH in order to repeat the run or/and debug the map and reduce scripts. 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 it is returned it visibly. Otherwise, it returns invisibly.

Note

direct.io can be switched on or off only when "dryrun" mode is enabled. As soon as "dryrun" is disabled. "direct.io" is also turned off.

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

Passes local objects to the 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 an unnamed value, which can not be exported is provided (e.g. orch.export(1)), 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 objects are embedded into mapReduce R script as source code. In the case of "rdata", all exported R objects are 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 tasks. Only variables and named values are included. If there are no variables to export, then NULL is 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 accommodate the debug facility of the ORCH framework. For instance, all global functions are from the export list because they are accessible as-is. Functions defined inside of scope of other functions (or other environments) are 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 and will be displayed 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 is issued upon loading of the ORCH library and the version specified by `ORCH_HAL_VERSION` is 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 in 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 in order 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_JAVA_MAX_PERM *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 set the flag `-XX:MaxPermSize` for the ORAAH client's Java Virtual Machine (JVM). This flag specifies the size for Permanent Generation, which is is where the classes, methods, internalized strings, and similar objects used by the JVM are stored. The default value of this flag for ORAAH is 256MB.

Details

This memory flag can be specified in multiple sizes, such as kilobytes (k), megabytes (m), gigabytes (g) and so on. See examples for specification. You can increase this memory size for the ORAAH client JVM using this environment variable for a new R session if you encounter `java.lang.OutOfMemoryError: PermGen space`.

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-envvar ORCH_JAVA_XMX

Examples

```
## Not run:
 csh: setenv ORCH_JAVA_MAX_PERM "1g"
 csh: setenv ORCH_JAVA_MAX_PERM "512m"
 bash: export ORCH_JAVA_MAX_PERM="256m"
 bash: export ORCH_JAVA_MAX_PERM="1g"

## End(Not run)
```

ORCH_JAVA_XMS *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 set the flag `-Xms` for the ORAAH client's Java Virtual Machine (JVM). This flag specifies the initial memory allocation pool for a JVM, which means that your JVM will be able to use an initial size of Xms amount of memory. The default value of this flag for ORAAH is 256 MB.

Details

This memory flag can be specified in multiple sizes, such as kilobytes (k), megabytes (m), gigabytes (g) and so on. See examples for specification.

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

ORHC_JAVA_XMX ORCH-envvar ORCH_JAVA_MAX_PERM

Examples

```
## Not run:
 csh: setenv ORCH_JAVA_XMS "4g"
 csh: setenv ORCH_JAVA_XMS "512m"
 bash: export ORCH_JAVA_XMS="10g"
 bash: export ORCH_JAVA_XMS="400m"

## End(Not run)
```

ORCH_JAVA_XMX *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 set the flag `-Xmx` for the ORAAH client's Java Virtual Machine (JVM). This flag specifies the maximum memory allocation pool for a JVM, which means that your JVM will be able to use a maximum of Xmx amount of memory. The default value of this flag for ORAAH is 1GB.

Details

This memory flag can be specified in multiple sizes, such as kilobytes (k), megabytes (m), gigabytes (g) and so on. See examples for specification. You can increase the memory available to the ORAAH client JVM using this environment variable for a new R session if you encounter `java.lang.OutOfMemoryError`.

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-envvar ORCH_JAVA_MAX_PERM

Examples

```
## Not run:
 csh: setenv ORCH_JAVA_XMX "4g"
 csh: setenv ORCH_JAVA_XMX "512m"
 bash: export ORCH_JAVA_XMX="10g"
 bash: export ORCH_JAVA_XMX="400m"

## End(Not run)
```

orch.jdbc.close *Closes JDBC connection created using orch.jdbc*

Description

Closes the JDBC connection created in the object of type "orch.jdbc". It is recommended to close the JDBC connection once the desired data has been ingested by the solvers that support "orch.jdbc" input type.

Usage

```
orch.jdbc.close(object)
```

Arguments

object	An "orch.jdbc" object created using orch.jdbc.

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:
jdbc_info <- orch.jdbc(driverClass = "com.mysql.jdbc.Driver",
  url = "jdbc:mysql://mysql.example.com:3306/mydb", user = "user",
  password = "password", table= "mytable",
  classpath = "/usr/lib/mysql/lib/mydriver.jar")
orch.jdbc.close(jdbc_info)

## End(Not run)
```

orch.jdbc *Create JDBC input object*

Description

Creates a JDBC connection descriptor object of type "orch.jdbc". This object can be used for specifying inputs for Spark and Spark MLlib analytics from ORCHstats package. Also, it can be used as input for ORCHmpi package solvers.

Usage

```
orch.jdbc(driverClass, url, user, password, table,
  classpath = "", identifier.quote = "`")
```

Arguments

driverClass	Name of the Java class of the JDBC driver to load.
url	A database/JDBC URL of the form: jdbc:[subprotocol]://[node]/[databaseName]. For example, "jdbc:mysql://mysqlserver.example.com:3306/mydb"
user	The username for database connection.
password	The password for the database connection.
table	The name of the input table.
classPath	Class path that needs to be appended in order to load the desired JDBC driver. Usually it is the path to the JAR file containing the driver.
identifier.quote	Character to use for quoting identifiers in automatically generated SQL statements or NA if the back-end doesn't support quoted identifiers.

Value

An object of type "orch.jdbc" which can be used as an input for Spark analytics available in packages ORCHstats and ORCHmpi.

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:
jdbc_info <- orch.jdbc(driverClass = "com.mysql.jdbc.Driver",
  url = "jdbc:mysql://mysql.example.com:3306/mydb", user = "user",
  password = "password", table= "mytable",
  classpath = "/usr/lib/mysql/lib/mydriver.jar")

## 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 absent.

Usage

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

Arguments

key	The key. Must be one-value vector or factor only. It may not consist of complex structures such as the list of data.frame. NULL value indicates key-less output (like "val1,val2"). A "" value indicates no-key output (such as "\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 are specified then only primitive types like numeric, integer, etc. can be used. Complex structures such as list and data.frame are not 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 outputs N records containing repeations of this key and 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. The user can invoke this function multiple times and any location within a of mapReduce R function, or may choose not to invoke the function at all which will result in no output. Every invocation pushes key and values into the ORCH driver's internal buffer, which continues 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 at time points determined by ORCH. 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 absent.

Usage

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

Arguments

`key`	The key. Must be a vector or factor of the same length as the value argument `....` Complex structures such as the list of data.frame may not be used. If only one key is specified and value argument `...` multiple records then this key will be replicated for each record. A NULL key indicates key-less output (auch as "val1,val2"). A "" key indicates no-key output (such as "\tval1,val2). See examples.
`...`	Key's value(s). Can be a data.frame, vector, factor, matrix, or list. If it is a data.frame or a matrix then each row is treated as a separate (key,value) record. If it is a vector or a factor then each value is treated as an individual record. In the case of 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`i, which may be one value only. In that case, the same key is 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 understood as a "return" expression of a mapReduce user function which does not break function execution. The user can invoke this function multiple times in any part of a mapReduce R function. Or, they may choose not to invoke this function, which will result in no output at all. Each invocation will push key and values into the ORCH driver's internal buffer which continues to accumulate returned values till the function finishes.

Depending on the values `...` data type this function behaves differently. The best option is to use data.frame as value type, because its native storage type is ORCH and it guarantees the best output performance. The next favourable type is vector, which is slightly slower, then matrix and then list, which provides the slowest 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 is 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 may 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 logging. 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 update or to retrieve: • none: Get the list of all ORCH options. • options_name=value[, ...]: Set one or more options. • "options_name"[,...]: Get one or more 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 specifying the penalty (see options("scipen")). Missing values correspond to the current default penalty.

Value

List of all ORCH option values if no options are specified in `...`. If the options are being retrieved, e.g., `orch.options(c("digits","scientific"))` then the functions returns only their values. Otherwise the function returns 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 more 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 its complexity and structure. The output is a string in a proprietary format based on base64 encoding which is guaranteed to not contain any special symbols like "\10","\t","\n", etc., or separators like ",", ",", etc. This packed string is safe to be used as a 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. The encoding will take longer but guarantees that the output has the smallest size possible. • "smart": Enables base64 "smart" compression setting. It is the same as "auto", but it will not attempt any compression on small 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(...), i.e., one can use it as pack(a=1, b='x'). The main motivation is to provide an ability to output complex datasets from a mapper or a 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.reconf *Reruns the checks for the session*

Description

Removes the temporary file with stored environment variable values Forces startup checks to rerun and assign new values Create a new file and store these values in the file

Usage

```
orch.reconf()
```

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 orch.destroyConf

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

Description

After a user invokes `orch.disconnect(dbcon=TRUE)` to drop the database connection, ORAAH 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, HIVE bucket sampling, HIVE bucket sampling or HIVE random 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
`type`	Character string specifying the type of sampling to apply on HIVE table. Default is <e2><80><98>block<e2><80><99>

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. When `type` argument is 'block' (default) HIVE uses block sampling as default, 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. When `type` argument

is ‘random‘, HIVE uses HIVE random sampling the sample size will be equal to the desired size, specified through the `percent` argument. HIVE random sampling counts the total row count of table and calculate the number of rows in sample, using `percent` argument. Then it runs qry to extract a sample of exact sample as specified. When `type` argument is ‘bucket‘, HIVE uses bucket sampling to sample the data. Size of the sample returned may vary depending on the distribution of data in buckets.

Consider the following when choosing Hive sampling type. ‘random‘ type of sampling is the slowest one since it runs two MapReduce jobs in a sequence to create a "true" sample calculating exact number of records that need to be kept in the sample and applying a random function to each record with random distribution and order. It should be used when you can allow extra time for the sample data to be generated. ‘bucket‘ type of sampling uses Hive’s "tablesample" function to bucket rows randomly and return of the buckets as a sample. Its sample size may vary slightly from requested sample size and this approach provide a good balance between quality of the sample and speed of sampling. Bucket sampling is the recommended method to use. ‘block‘ sampling is the fastest one of all other types but it’s not "true" sampling by far as it will return data at granularity of HDFS block level, i.e. if block size is 256MB, even if n percent of input size is only 100MB, you get 256MB of data.

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 dataset 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))

# Hive random sampling
zzz <- orch.sample(iris1, percent=10, output = "samp_rand10", type='random')
print(nrow(zzz))
```

`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 = "local", memory = NULL,
  dfs.namenode = NULL, name = NULL, disconnect = FALSE,
  logLevel = "ERROR", enableHive = 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. If not set the default value "local" is used.
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 an HDFS object into RDD object. If not set, then the active Namenode is determined for the cluster and used. The Namenode being used will be reported as an INFO message. If auto-detection of Namenode fails, then local file-system will be used, which is again reported to the user.
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".
disconnect	Explicitely disconnect an existing active Spark session first if set to TRUE, otherwise if there is an active Spark session already then it will remain active consuming Spark cluster resources. See description for more details.
logLevel	The log4j logging level for Spark connection. logLevel can be one of these types: "ALL", "DEBUG", "ERROR", "FATAL", "INFO", "OFF", "TRACE" or "WARN". The default logging level is "ERROR".
enableHive	Enables Hive Support within Spark Session. By default, Hive Support is enabled if you are connected to HIVE using ore.connect(..., type = "HIVE"). For Hive support to be enabled in Spark session you need to have your Hive configuration available at the client side in the form of 'hive-site.xml' file in your 'CLASSPATH'. You can do so by adding your 'HIVE_CONF_DIR' to the 'CLASSPATH' before starting R and loading ORCH library. Alternatively, you can specify the Hive metastore details as part of spark.connect by specifying parameters like hive.metastore.uris="thrift://METASTORE_NAME:METASTORE_POR If your Hive is kerberized then you need to additionally specify other parameters like hive.metastore.sasl.enabled="true", hive.metastore.kerberos.pri If you do not wish to enable Hive Support in Spark while being connected to Hive, then use enableHive=FALSE to start a session without it.
...	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.instances: The number of parallel Spark worker instances to create. If using master='yarn-client', check with the cluster administrator, since a good default might be the number of nodes in the cluster that can run YARN containers. The default is set to '2'. 2. spark.executor.cores: The number of cores to use on each Spark 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. If using master='yarn-client', check with the cluster administrator for the maximum cores per YARN Container. The default is set to '2'. 3. 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.defaultC on Spark's standalone cluster manager. It needs to be at least spark.executor.cores * s

4. spark.driver.memory: Spark Driver memory might be important for large problems. Suggestion in case of memory problems could be to start with '1g' and grow if necessary.
5. 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 currently an "active" Apache Spark session (i.e., this function was already invoked once before) then it will be made "inactive" but will **not** be terminated immediately, i.e. Apache Spark cluster resources will remain allocated until this session is explicitely teminated with spark.disconnect call. Otherwise if there are no references to this session left in R environment, then it will be terminated automatically at some point by R and Java garbage collectors and all Apache Spark cluster resources will be released. This behavior can be altered using disconnect parameter that will cause immediate termination of the current Apache Spark session.

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:
# Using "yarn-client" - deprecated method (Spark 2.0+)
spark.connect(master="yarn-client", memory="1g",
  dfs.namenode="mynamenode.example.com", spark.executor.cores=2)
# Using "yarn" - correct method
spark.connect(master="yarn", spark.submit.deployMode="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

`orch.summary` *Hive Data Summary*

Description

Generates descriptive statistics for `ore.frame` objects within flexible row aggregations.

Usage

```
orch.summary(data, var, stats = c("n", "mean", "min", "max"),
             class = NULL, types = NULL, ways = NULL, weight = NULL,
             order = NULL, maxid = NULL, minid = NULL, mu = 0,
             no.type = FALSE, no.freq = FALSE)
```

Arguments

data	An ore.frame object of data.
var	A vector of character strings specifying the names of numeric columns in argument data to which to apply all of the statistical calculations in argument stats, or a list of character string vectors. If the var argument is a list, then the length of the list must be either 1 or the same as the length of stats. If it's a list of length 1, it's equivalent to a vector of strings. If it's a list of length greater than 1, each element of the var list specifies the columns of data to which to apply the statistical calculation in the corresponding position in stats.
stats	A vector of character strings specifying the statistical calculations for argument var. If the name of the vector element is specified, the name becomes the output column name. The values of this argument can be one or more of the following: "n" or "freq" (Count of non-missing values), "count" or "cnt" (Count of all observations), "nmiss" (Count of missing values), "mean" or "avg" (Average of values), "min" (Minimum of values) "max" (Maximum of values), "css" (Corrected sum of squares), "uss" (Uncorrected sum of squares), "cv" (Coefficient of variation), "sum" (Sum of values), "sumwgt" (Weighted sum of values), "range" (Range of values), "stddev" or "std" (Standard deviation of values), "stderr" or "stdmean" (Standard error for the mean), "variance" or "var" (Variance of values), "kurtosis" or "kurt" (Kurtosis), "skewness" or "skew" (Skewness), "loccount<" or "loc<" (Number of observations whose values are less than the supplied mu), "loccount>" or "loc>" (Number of observations whose values are greater than the supplied mu), "loccount!" or "loc!" (Number of observations whose values are not equal to the supplied mu), "loccount" or "loc" (Number of observations whose values are equal to the supplied mu), Percentiles Types: "p0", "p1", "p5", "p10", "p25" or "q1", "p50" or "q2" or "median", "p75" or "q3", "p90", "p95", "p99", "p100" (Percentile or quantile), "qrange" or "iqr" (Interquartile range, Q3-Q1), "mode" (Most frequently occurring value), "lclm" (Two-sided left confidence limit with confidence level of the interval equal to 0.95), "rclm" (Two-sided right confidence limit with confidence level of the interval equal to 0.95),

"clm" (Two-sided confidence interval with confidence level of the interval equal to 0.95),

"t" (Student's t-test statistic),

"probt" or "prt" (Two-tailed p-value for student's t-test)

class — A vector of character strings specifying the names of categorical columns within argument data. If not specified, the aggregation of the entire data is returned.

types — A list of character string vectors specifying the combinations of the column names in class within which the aggregations will be executed in the returning summary.

ways — A vector of integers with each value indicating the number of columns in class that are used to generate types. With one integer number, it generates types of all possible combinations with the specified number of columns in class. The types generated by ways will be combined with the types specified in types with redundency removed automatically.

weight — An optional single character string specifying a numeric column within data to use as analytic weights. By default, the weight for each non-missing observation is 1. The statistics in stats that can take weight are "sum", "sumwgt", "mean", "css", "uss", "cv", "stddev", "variance", and "stderr". The weight argument is ignored when specified with other statistics.

order — A vector of character strings specifying the sorting criteria. The values of this argument can be one or more of the following:

"freq" or "-freq" (Ascending or descending sorts based on count statistics),

"type" or "-type" (Ascending or descending sorts based on type),

"class" or "-class" (Ascending or descending sorts based on the columns in class).

maxid — A named vector of character strings, each element of which specifies two columns in data. The name of an element specifies an over-column and the value of the element specifies an id-column. Each element results in an additional column in the returned ore.frame object. Each additional column contains the value from the id-column that corresponds to the observation that has the maximum value in the over-column.

minid — A named vector of character strings, each element of which specifies two columns in data. The name of an element specifies an over-column and the value of the element specifies an id-column. Each element results in an additional column in the returned ore.frame object. Each additional column contains the value from the id-column that corresponds to the observation that has the minimum value in the over-column.

mu — A single number or a vector of numbers whose elements correspond to each value in var, to supply additional numeric parameters for some statistics. The default value is 0. The statistics that use mu are "loccount<", "loccount>", "loccount", "loccount!", "t", and "probt". The mu argument is ignored when specified with other statistics.

no.type — A logical value indicating whether to drop the TYPE column from the output.

no.freq — A logical value indicating whether to drop the FREQ column from the output.

Details

The function `ore.summary` generates descriptive statistics for `ore.frame` objects within user specified aggregation sub-groups.

The argument `class` specifies the columns to be used to define aggregation sub-groups. The arguments `types` and `ways` define the sub-groups. If `class` is NULL, the function aggregates the entire data without sub-groups. If `class` is specified, but both `types` and `ways` are NULL, the function returns aggregations of all possible sub-groups by the columns in `class`. The number of sub-groups increases exponentially over the number of `class` columns. Oracle recommands using `types` and `ways` to specify the sub-groups of interest.

Value

Returns an `ore.frame` object.

Author(s)

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

References

Oracle R Enterprise

Examples

```
ore.create(iris,"iris1")

orch.summary(iris1, c("sepal_length", "petal_length"))

orch.summary(iris1, c("sepal_length", "petal_length"), c("mean", "std", "p10"),
  class="species")

orch.summary(iris1, list(c("sepal_length", "petal_length"),
  "sepal_width"), c(AVG="mean", "std"), class="species")

orch.summary(iris1, c("sepal_length", "petal_length"), c("mean", "std"),
  class="species", weight="sepal_width")

orch.summary(iris1, c("sepal_length", "petal_length"), c("mean", "std"),
  class=c("species", "petal_width"),
  types=list("species", c("species", "petal_width")),
  order=c("type", "-freq", "class"))

orch.summary(iris1, c("sepal_length", "petal_length"), c("mean", "std"),
  class=c("species", "petal_width"),
  ways=1, order=c("type", "-freq", "class"))

orch.summary(iris1, c("sepal_length", "petal_length"), c("mean", "prt"),
  class="species", mu=c(5.8, 3.7))

orch.summary(iris1, c("sepal_length", "petal_length"), "mean",
  class="species",
  maxid=c(sepal_length="sepal_width"))

ore.drop("iris1")
```

`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 this argument is not specified, the method 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, method = "irls",
  relObjDiff = 1e-08, relVarDiff = 1e-08,
  maxIterations = 20L, maxBlockRows = 20000L,
  storageLevel = "MEMORY_ONLY", 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	Input data for model fitting. The different input types supported are as follows: • 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. • An Oracle Distributed Model Matrix object prepared using orch.model.matrix function. • An ore.frame object, when connected to "HIVE" or "IMPALA" using ore.connect. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using orch.jdbc(...).

method	Algorithm to solve the underlying optimization model. "irls" iteratively reweighted least squares (recommended, default); or "lbfgs" Limited-memory Broyden-Fletcher-Goldfarb-Shanno, recommended for models with very large number (e.g. more than 10000) of numerical features, or whenever memory available to each Spark worker is severely limited. For L-BFGS it is important to increase the maximum number of iterations to, say, at least 200.
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. This parameter is deprecated, and unused by the L-BFGS method.
maxIterations	Maximum number of IRLS iterations. By default, 20 iterations is used as the stopping criterion.
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.
storageLevel	To control the storage of the Input Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are "" (empty string, means to use the default Spark storage level), "NONE", "DISK_ONLY", "DISK_ONLY_2", "MEMORY_ONLY", "MEMORY_ONLY_2", "MEMORY_ONLY_SER", "MEMORY_ONLY_SER_2", "MEMORY_AND_DISK", "MEMORY_AND_DISK_2", "MEMORY_AND_DISK_SER", "MEMORY_AND_DISK_SER_2", "OFF_HEAP". Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".
verbose	Whether to report progress and performance statistics. Default value is TRUE.

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)).

When method = "irls" the implementation will use a parallel distributed Iteratively Reweighted Least Squares (IRLS) algorithm. To carry out IRLS iterations ORAAH GLM utilizes efficient parallel distributed linear algebra algorithms, including parallel supernodal Cholesky factorization. When method = "lbfgs" GLM2 will switch to a parallel distributed Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm. L-BFGS is an experimental feature in this release, and should be used when the number of numerical features is large, or memory available for Spark worker processes is severly limited. When using L-BFGS solver the maximum number of iterations maxIterations parameter should be increated to be at least 200. Note: relVarDiff is unused when method = "lbfgs".

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, storageLevel = "MEMORY_ONLY",
  maxBlockRows = 20000L, 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` Input data for model fitting. The different input types supported are as follows:

- 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.
- An Oracle Distributed Model Matrix object prepared using `orch.model.matrix` function.
- An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`.
- A Spark dataframe created using any other external method or Spark API directly.
- An 'orch.jdbc' object created using `orch.jdbc(...)`.

`storageLevel` To control the storage of the Input Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`, `"MEMORY_AND_DISK_SER_2"`, `"OFF_HEAP"`. Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`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 value is `TRUE`.

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. The 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 ORAAH Models from HDFS.*

Description

This function loads a model created using Spark analytics 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

Model object of type among orch.glm2, orch.lm2, orch.neural2, 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 or orch.ml.gbt 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.mdf *Creates an MLlib Data Frame (MDF).*

Description

MLlib machine learning and statistical algorithms require a special row-based distributed frame for training and scoring.

Usage

```
orch.mdf(formula, data, factorMode = "one_hot",
  maxBlockRows = 20000L, storageLevel = "MEMORY_ONLY",
  verbose = TRUE, ...)
```

Arguments

formula	A formula representing the model to be fit (see "details" section below for more information.)
data	Input data for prediction. The different input types supported are as follows: • 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. • An ore.frame object, when connected to "HIVE" or "IMPALA" using ore.connect. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using orch.jdbc(...).
factorMode	Factor mode. "one_hot" 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.
storageLevel	To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are "" (empty string, means to use the default Spark storage level), "NONE", "DISK_ONLY", "DISK_ONLY_2", "MEMORY_ONLY", "MEMORY_ONLY_2", "MEMORY_ONLY_SER", "MEMORY_ONLY_SER_2", "MEMORY_AND_DISK", "MEMORY_AND_DISK_2", "MEMORY_AND_DISK_SER", "MEMORY_AND_DISK_SER_2", "OFF_HEAP". Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".
verbose	Whether to report progress and performance statistics. Default is TRUE.
...	additional arguments.

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

MDF 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.mdf(Kyphosis ~ Number, data = data)
hdfs.rm(data)
```

---

orch.ml.dt *MLlib Decision Tree.*

Description

MLlib Decision Tree.

Usage

```
orch.ml.dt(formula, data, type = NULL, impurity = NULL,
  maxDepth = NULL, maxBins = NULL,
  minInstancesPerNode = NULL, minInfoGain = NULL,
  maxCategories = 32L, threshold = NULL,
  maxBlockRows = 20000L, storageLevel = "MEMORY_ONLY",
  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	Input data for model fitting. The different input types supported are as follows: • 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. • An MLlib input Dataframe object prepared using orch.model.matrix(..., factorMo function. • An MLlib input Dataframe object prepared using orch.mdf(..., factorMode="none" function.

- An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`.
- A Spark dataframe created using any other external method or Spark API directly.
- An 'orch.jdbc' object created using `orch.jdbc(...)`.

`type` Can be set to "classification" or "regression". Default value is `NULL`, in which case it will be determined automatically based on the input dataset 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 `type`.

`maxDepth` Maximum depth of the decision tree, default value `5`.

`maxBins` Maximum number of bins used for splitting features, default value `32`.

`minInstancesPerNode` Minimum number of instances each child must have after a split.

`minInfoGain` Minimum information gain for a split to be considered at a tree node.

`maxCategories` Features with levels higher than `maxCategories` distinct values are treated as continuous.

`threshold` Thresholds are used in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold. This parameter will not have any effect for `type="regression"` models.

`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.

`storageLevel` To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`, `"MEMORY_AND_DISK_SER_2"`, `"OFF_HEAP"`. Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`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*:

$$IG\left(\mathcal{R}, \mathcal{R}_1, \mathcal{R}_2\right) = I\left(\mathcal{R}\right) - \frac{N_1}{N} I\left(\mathcal{R}_1\right) - \frac{N_2}{N}\left(\mathcal{R}_2\right)$$

.

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 $I\left(\cdot\right)$ 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: $I^{\text{Gini}}\left(\mathcal{R}\right) = \sum_{i=1}^{C} f_i\left(1 - f_i\right)$;
- Entropy: $I^{\text{Entropy}}\left(\mathcal{R}\right) = -\sum_{i=1}^{C} f_i \log f_i$.

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:

$$I^{\text{Variance}}\left(\mathcal{R}\right) = \text{VAR}\left(y\right), \text{ subject to } x_i \in \mathcal{R}.$$

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, type="classification"
pred  <- predict(model, newdata = data, supplemental = c("Kyphosis", "Age"))
hdfs.write(pred, outPath = "kyphosisPrediction", overwrite = TRUE)
```

---

orch.ml.gbt *MLlib gradient Boosted Trees.*

Description

MLlib Gradient-Boosted Trees (GBTs) are ensembles of decision trees. GBTs iteratively train decision trees in order to minimize a loss function. Like decision trees, GBTs handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions. MLlib supports GBTs for binary classification and for regression, using both continuous and categorical features.

Usage

```
orch.ml.gbt(formula, data, type = NULL,
  maxIterations = 100L, maxCategories = 32L,
  threshold = NULL, maxBlockRows = 20000L,
  storageLevel = "MEMORY_ONLY", 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	Input data for model fitting. The different input types supported are as follows: • 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. • An MLlib input Dataframe object prepared using orch.model.matrix(..., factorMo function. • An MLlib input Dataframe object prepared using orch.mdf(..., factorMode="none" function. • An ore.frame object, when connected to "HIVE" or "IMPALA" using ore.connect. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using orch.jdbc(...).
type	Can be set to "classification" or "regression". Default value is NULL, in which case it will be determined automatically based on the input dataset and formula.
maxIterations	Maximum number of iterations. By default, 100 iterations is used as the stopping criterion.

maxCategories
: Features with levels higher than maxCategories distinct values are treated as continuous.

threshold
: Thresholds are used in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold. This parameter will not have any effect for type="regression" models.

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.

storageLevel
: To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are "" (empty string, means to use the default Spark storage level), "NONE", "DISK_ONLY", "DISK_ONLY_2", "MEMORY_ONLY", "MEMORY_ONLY_2", "MEMORY_ONLY_SER", "MEMORY_ONLY_SER_2", "MEMORY_AND_DISK", "MEMORY_AND_DISK_2", "MEMORY_AND_DISK_SER", "MEMORY_AND_DISK_SER_2", "OFF_HEAP". Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

verbose
: Whether to report progress and performance statistics. Default value is TRUE.

Value

Gradient Boosted Tree fit object, orch.ml.gbt.

Attention

MLlib Gradient-Boosted Trees do not yet support multiclass classification. For multiclass problems, please use decision trees (orch.ml.dt) or Random Forests (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.gbt oracle.model.matrix

Examples

```
library(rpart)
data  <- hdfs.put(kyphosis)
model <- orch.ml.gbt(formula = Kyphosis ~ Number + Age, data = data, type="classification
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,
  seed = as.integer(1e+08 * runif(1)),
  maxBlockRows = 20000L, storageLevel = "MEMORY_ONLY",
  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` Input data for model fitting. The different input types supported are as follows:

- 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.
- An MLlib input Dataframe object prepared using `orch.model.matrix(..., type="md` function.
- An MLlib input Dataframe object prepared using `orch.mdf` function.
- An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`.
- A Spark dataframe created using any other external method or Spark API directly.
- An 'orch.jdbc' object created using `orch.jdbc(...)`.

`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.

`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.

`storageLevel` To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`,

"MEMORY_ONLY_SER_2", "MEMORY_AND_DISK", "MEMORY_AND_DISK_2", "MEMORY_AND_DISK_SER", "MEMORY_AND_DISK_SER_2", "OFF_HEAP". Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`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, initializationMode = "k-means||",
  seed = as.integer(1e+08 * runif(1)),
  maxBlockRows = 20000L, storageLevel = "MEMORY_ONLY",
  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	Input data for model fitting. The different input types supported are as follows: • 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. • An MLlib input Dataframe object prepared using orch.model.matrix(..., type="md function. orch.mdf function. • An ore.frame object, when connected to "HIVE" or "IMPALA" using ore.connect. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using orch.jdbc(...).
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.

`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.

`storageLevel` To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`, `"MEMORY_AND_DISK_SER_2"`, `"OFF_HEAP"`. Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`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 $\mathcal{D} = \{\mathbf{x}_i : i = 1, 2, \ldots m\}$. Let cluster centroid points be $\{\mu_j : j = 1, 2, \ldots k\}$, and the paritioning of the input points into clusters based on nearest centroid criteria at any moment $\mathcal{S} = \{\mathcal{S}_j : j = 1, 2, \ldots k\}$. 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, regParam = 0.3,
  convergenceTol = 1e-04, maxIterations = 100L,
  standardization = TRUE, maxBlockRows = 20000L,
  storageLevel = "MEMORY_ONLY", 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` Input data for model fitting. The different input types supported are as follows:

- 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.
- An MLlib input Dataframe object prepared using `orch.model.matrix(..., type="md`
function.
- An MLlib input Dataframe object prepared using `orch.mdf` function.
- An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`.
- A Spark dataframe created using any other external method or Spark API directly.
- An 'orch.jdbc' object created using `orch.jdbc(...)`.

`regParam` Regularization parameter, default value `0.3`.

`convergenceTol`
Convergence tolerance. By default, `1E-4` is used as the stopping criterion.

`maxIterations`
Maximum number of iterations. By default, `100` iterations is used as the stopping criterion.

`standardization`
Whether to standardize the training features before fitting the model. 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.

`storageLevel` To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`, `"MEMORY_AND_DISK_SER_2"`, `"OFF_HEAP"`. Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`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

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:

$$\boldsymbol{\beta}^{(j+1)} = \text{prox}_{\lambda\alpha^{(j)}\|\cdot\|_1}\left(\boldsymbol{\beta}^{(j)} + \frac{\alpha^{(j)}}{|\mathcal{S}_j|}\sum_{i}^{\boldsymbol{x}_i\in\mathcal{S}_j} r_i^{(j)}\boldsymbol{x}_i\right),$$

where $\text{prox}_{\lambda\alpha^{(j)}\|\cdot\|_1}(\cdot)$ 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:

$$\text{prox}_{\gamma\|\cdot\|_1}(f) = \begin{cases} f-\gamma, & f > \gamma; \\ 0, & -\gamma \le f \le \gamma; \\ f+\gamma, & f < -\gamma. \end{cases}$$

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, elasticNetParam = 0.8,
  regParam = 0.3, convergenceTol = 1e-04,
  maxIterations = 100L, standardization = TRUE,
  maxBlockRows = 20000L, storageLevel = "MEMORY_ONLY",
  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` Input data for model fitting. The different input types supported are as follows:

- 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.
- An MLlib input Dataframe object prepared using `orch.model.matrix(..., type="md` function.
- An MLlib input Dataframe object prepared using `orch.mdf` function.
- An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`.
- A Spark dataframe created using any other external method or Spark API directly.
- An 'orch.jdbc' object created using `orch.jdbc(...)`.

`elasticNetParam` The ElasticNet mixing parameter, default value `0.8`.

`regParam` Regularization parameter, default value `0.3`.

`convergenceTol` Convergence tolerance. By default, `1E-4` is used as the stopping criterion.

`maxIterations` Maximum number of iterations. By default, `100` iterations is used as the stopping criterion.

`standardization` Whether to standardize the training features before fitting the model. 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.

`storageLevel` To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`, `"MEMORY_AND_DISK_SER_2"`, `"OFF_HEAP"`. Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`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

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, maxIterations = 100L,
  threshold = NULL, maxBlockRows = 20000L,
  storageLevel = "MEMORY_ONLY", 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	Input data for model fitting. The different input types supported are as follows: • 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. • An MLlib input Dataframe object prepared using orch.model.matrix(..., type="md function. • An MLlib input Dataframe object prepared using orch.mdf function. • An ore.frame object, when connected to "HIVE" or "IMPALA" using ore.connect.

- A Spark dataframe created using any other external method or Spark API directly.
- An 'orch.jdbc' object created using `orch.jdbc(...)`.

`maxIterations`
: Maximum number of iterations. By default, `100` iterations is used as the stopping criterion.

`threshold`
: Thresholds are used in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.

`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.

`storageLevel`
: To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`, `"MEMORY_AND_DISK_SER_2"`, `"OFF_HEAP"`. Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`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.random.forest

MLlib Random Forest.

Description

MLlib Random Forest.

Usage

```
orch.ml.random.forest(formula, data, nTrees = 1L,
  type = NULL, impurity = NULL, maxDepth = 5L,
  maxBins = 32L, featureSubsetStrategy = "auto",
  minInstancesPerNode = 1L, minInfoGain = 0,
  maxCategories = 32L, threshold = NULL,
  maxBlockRows = 20000L, storageLevel = "MEMORY_ONLY",
  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	Input data for model fitting. The different input types supported are as follows: • 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. • An MLlib input Dataframe object prepared using orch.model.matrix(..., factorMo function. • An MLlib input Dataframe object prepared using orch.mdf(..., factorMode="none" function. • An ore.frame object, when connected to "HIVE" or "IMPALA" using ore.connect. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using orch.jdbc(...).
nTrees	Number of trees in the forest, default value is 1. Generally you want as many trees as will improve your model. More trees also mean more computational cost and after a certain number of trees, the improvement is negligible. After sometime there is no significant improvement in error rate even if we are increasing no of tree.
type	Can be set to "classification" or "regression". Default value is NULL, in which case it will be determined automatically based on the input dataset 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 dataset 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.

`minInstancesPerNode`
: Minimum number of instances each child must have after a split.

`minInfoGain`
: Minimum information gain for a split to be considered at a tree node.

`maxCategories`
: Features with levels higher than `maxCategories` distinct values are treated as continuous.

`threshold`
: Thresholds are used in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold. This parameter will not have any effect for `type="regression"` models.

`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.

`storageLevel`
: To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`, `"MEMORY_AND_DISK_SER_2"`, `"OFF_HEAP"`. Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`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, type="classification")
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, regParam = 0.3,
  convergenceTol = 1e-04, maxIterations = 100L,
  standardization = TRUE, maxBlockRows = 20000L,
  storageLevel = "MEMORY_ONLY", 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	Input data for model fitting. The different input types supported are as follows: • 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. • An MLlib input Dataframe object prepared using orch.model.matrix(..., type="md function. • An MLlib input Dataframe object prepared using orch.mdf function. • An ore.frame object, when connected to "HIVE" or "IMPALA" using ore.connect. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using orch.jdbc(...).
regParam	Regularization parameter, default value 0.3.
convergenceTol	Convergence tolerance. By default, 1E-4 is used as the stopping criterion.
maxIterations	Maximum number of iterations. By default, 100 iterations is used as the stopping criterion.

`standardization`
: Whether to standardize the training features before fitting the model. 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.

`storageLevel`
: To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`, `"MEMORY_AND_DISK_SER_2"`, `"OFF_HEAP"`. Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`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

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.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,
  maxIterations = 100L, regParam = 0.01,
  threshold = NULL, maxBlockRows = 20000L,
  storageLevel = "MEMORY_ONLY", 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	Input data for model fitting. The different input types supported are as follows: • 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. • An MLlib input Dataframe object prepared using orch.model.matrix(..., type="md function. • An MLlib input Dataframe object prepared using orch.mdf function. • An ore.frame object, when connected to "HIVE" or "IMPALA" using ore.connect. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using orch.jdbc(...).
convergenceTol	Convergence tolerance. By default, 1E-4 is used as the stopping criterion.
maxIterations	Maximum number of iterations. By default, 100 iterations is used as the stopping criterion.
regParam	Regularization parameter, default value 0.01.
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.
storageLevel	To control the storage of the MLlib Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are "" (empty string, means to use the default Spark storage level), "NONE", "DISK_ONLY", "DISK_ONLY_2", "MEMORY_ONLY", "MEMORY_ONLY_2", "MEMORY_ONLY_SER", "MEMORY_ONLY_SER_2", "MEMORY_AND_DISK", "MEMORY_AND_DISK_2", "MEMORY_AND_DISK_SER", "MEMORY_AND_DISK_SER_2", "OFF_HEAP". Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".
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

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:

$$L_{\text{hinge}}\left(\boldsymbol{\beta}\right) = \max\left(0, 1 - y\boldsymbol{\beta}^{\top}\mathbf{x}\right);$$

$$R\left(\boldsymbol{\beta}\right) = \frac{1}{2}\left\|\boldsymbol{\beta}\right\|_2^2;$$

$$L\left(\boldsymbol{\beta}\right) = L_{\text{hinge}}\left(\boldsymbol{\beta}\right) + \lambda R\left(\boldsymbol{\beta}\right).$$

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, factorMode = "one_hot",
  type = "dmm", maxBlockRows = 20000L,
  storageLevel = "MEMORY_ONLY", verbose = TRUE, ...)
```

Arguments

`formula`	A `formula` representing the model to be fit (see "details" section below for more information.)
`data`	Input data for prediction. The different input types supported are as follows: • 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. • An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using `orch.jdbc(...)`.
`factorMode`	Factor mode. "one_hot" and "none" are supported.
`type`	"dmm" distributed model matrix type; "mdf" MLlib dataframe for input to MLlib algorithms 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.
`storageLevel`	To control the storage of the Model matrix created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`,

"MEMORY_AND_DISK_SER_2", "OFF_HEAP". Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

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.multivar` *Multivariate statistics for HDFS objects*

Description

Multivariate numerical aggregation methods for `hdfs.id` objects based on function in R's **stats** package.

Usage

```
orch.cov(x, use="everything", num.mappers=-1, num.reducers=1,
         reducer.serial.limit=8, task.timeout=-1,
         job.name="ORCH Covariance Matrix")
orch.cor(x, use="everything", num.mappers=-1, num.reducers=1,
         reducer.serial.limit=8, task.timeout=-1,
         job.name="ORCH Correlation Matrix")
```

Arguments

`x`	An `hdfs.id` object containing numeric columns. This input matrix is in dense matrix representation
`use`	A method of computation when missing values are present. One of `"everything"`, `"all.obs"`, `"complete.obs"`, or `"na.or.complete"`.
`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

These statistics are calculated using multiple Map Reduce jobs. Computation of `cov` can be broken up into computing the `crossproduct`, `colSums` and number of rows of the input matrix.

Each of these can be computed in parallel by having tasks work on disjoint sets of rows of the input matrix. The reducer then puts all these together. For more parallelism during reducer computation, a tree of reducers are used.

Computation of `cor` is achieved by invoking `cov2cor` on the Covariance matrix.

Unlike the `cor` and `cov` functions in the **stats** package, `use = "pairwise.complete.obs"` and `method %in% c("kendall", "spearman")` are not supported.

Value

An in memory R matrix of dimension `ncol(x)` by `ncol(x)`.

Author(s)

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

See Also

cor, cov,

Examples

```
LONGLEY <- hdfs.put(longley)
all.equal(orch.cor(LONGLEY), cor(longley), check.attributes=FALSE)
all.equal(orch.cov(LONGLEY), cov(longley), check.attributes=FALSE)
```

---

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 can 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-08, maxIterations = 200L,
  objMinProgress = 1e-06, lowerBound = -0.7,
  upperBound = 0.7, seed = as.integer(1e+08 * runif(1)),
  nUpdates = 20L, scaleHessian = TRUE,
  maxBlockRows = 20000L, storageLevel = "MEMORY_ONLY",
  verbose = getOption("orch.trace", FALSE))
```

Arguments

formula	A formula object.
data	Input data for model fitting. The different input types supported are as follows: • 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. • An Oracle Distributed Model Matrix object prepared using orch.model.matrix function. • An ore.frame object, when connected to "HIVE" or "IMPALA" using ore.connect. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using orch.jdbc(...).

`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)	$f(x) = \log(1 + \exp(x)) - yx$
`"softmax"`	softmax (output only)	$f_i(x) = exp(x_i)/sum_j(exp(x_j))wherexistheinputvectorandx_i(orx_j)isits$

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 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.

`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.

`storageLevel` To control the storage of the Input Spark dataframe, created from HDFS CSV data, ore.frame objects or JDBC connection. The vaild choices are `""` (empty string, means to use the default Spark storage level), `"NONE"`, `"DISK_ONLY"`, `"DISK_ONLY_2"`, `"MEMORY_ONLY"`, `"MEMORY_ONLY_2"`, `"MEMORY_ONLY_SER"`, `"MEMORY_ONLY_SER_2"`, `"MEMORY_AND_DISK"`, `"MEMORY_AND_DISK_2"`, `"MEMORY_AND_DISK_SER"`, `"MEMORY_AND_DISK_SER_2"`, `"OFF_HEAP"`. Check Spark documentation for more information on Storage Level differences. The default value is "MEMORY_ONLY".

`verbose` A logical value that indicates whether to print out execution information.

Details

ORAAH Neural2 is used to train multilayer feed-forward neural network models. Multilayer means that the neurons are groupped into layers, forming a directed acyclic (feed-forward) graph.

The numerical optimization solver implements a parallel distributed L-BFGS algorithm with a line search. The line search termination criteria are based on Armijo sufficient decrease and Wolfe curvature conditions.

ORAAH Neural2 can efficiently handle both numeric and high cardinality factor variables. ORAAH Neural2 automatically switches to an out-of-core mode, if the input data does not fit into the distributed memory.

Value

A neural network model object, `orch.neural2`.

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.neural2 oracle.model.matrix

Examples

```
# regression with iris dataset
    IRIS  <- hdfs.put(iris)
    model <- 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(model)
    p <- predict(model, 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)
    model <- orch.neural2(formula     = Kyphosis ~.,
                          data        = KYPHOSIS,
                          hiddenSizes    = c(20, 20),
                          activations    = c("sigmoid", "sigmoid", "entropy"),
                          seed        = 0,
                          verbose     = TRUE)
    p <- predict(model, 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 preprocess 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	$f(x) = \frac{x}{1+\lvert x\rvert}$
`"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-kmeans

ORCH Predictions Using `kmeans` *and* `orch.kmeans` *Models*

Description

ORCH method for generating predictions using `kmeans` and `orch.kmeans` Models.

Usage

```
  ## S4 method for signature 'kmeans'
orch.predict(object, newdata, skip.vals, num.mappers,
                    task.timeout, job.name, ...)
  ## S4 method for signature 'orch.kmeans'
orch.predict(object, newdata, skip.vals, num.mappers,
                    task.timeout, job.name, ...)
```

Arguments

object	A kmeans or orch.kmeans model object.
newdata	An HDFS 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.
num.mappers	Hint for number of mappers to be used for the Hadoop jobs. Hadoop defaults are used.
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.
...	Optional arguments.

Value

Returns an hdfs.id object. This corresponds to the output HDFS file. The column named "ORCH_classes" contains the cluster classifications and the column named "ORCH_distance" contains the distance of the row from its corresponding center.

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 among these columns is preserved. Finally, the columns corresponding to the prediction results, "ORCH_classes" and "ORCH_distance" follow in that order.

Author(s)

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

See Also

kmeans. orch.kmeans

Examples

```
iris4dir <- hdfs.put(iris[,1:4])

ick <- kmeans(as.matrix(iris[,1:4]), centers = 3)
kout <- orch.predict(ick, iris4dir)
head(hdfs.get(kout))

ico <- orch.kmeans(iris4dir, centers = 3, iter.max=2)
oout <- orch.predict(ico, iris4dir)
head(hdfs.get(oout))
```

`orch.predict-princomp`

ORCH Predictions Using `princomp` *Models*

Description

ORCH method for generating predictions using `princomp` Models.

Usage

```
  ## S4 method for signature 'princomp'
orch.predict(object, newdata, skip.vals, num.mappers, task.timeout,
           job.name, ...)
```

Arguments

`object`	A `princomp` object.
`newdata`	An HDFS 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.
`num.mappers`	Hint for number of mappers to be used for the Hadoop jobs. Hadoop defaults are used.
`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.
`...`	Optional arguments.

Details

Prediction works independently on each row of the input HDFS object. Thus, this can be performed in parallel using `predict.princomp` in a mapper only job.

If the original fit used a formula or a data frame or a matrix with column names, `newdata` must contain columns with the same names. Otherwise, it must contain the same number of columns, to be used in the same order. The key column in `newdata`, if there is one, is not included in this consideration.

Value

Returns an `hdfs.id` object. This corresponds to the output HDFS file containing the rotated columns of `newdata`.

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, the rotated columns will appear next. The remaining columns in newdata will appear at the end if and only if `skip.vals==FALSE`. The order within the remaining columns is preserved.

Author(s)

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

See Also

orch.predict, princomp.

Examples

```
irisModel <- princomp(~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
                      data = iris)
IRIS <- hdfs.put(iris)
orch.predict(irisModel, IRIS)


USARRESTS <- hdfs.put(USArrests)
arrestsModel <- orch.princomp(USARRESTS, cor = TRUE)

res <- orch.predict(arrestsModel, USARRESTS)
head(hdfs.get(res))
```

---

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.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 analytics 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, orch.ml.gbt, orch.glm2, orch.lm2, orch.neural2.
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.glm2 *Summary for the high performance logistic regression, for the class* orch.glm2.

Description

Summary for the high performance logistic regression, for the class orch.glm2.

Usage

```
summary.orch.glm2(object, ...)
```

Arguments

object	an object of class "orch.glm2", returned by a call to orch.glm2. The object comprises the following components • coefficients matrix of coefficients, standard errors, z-values and p-values. • nIterations number of iterations. • deviance negative deviance.

- nRows number of rows (observations) in the input model matrix.
- nullDeviance deviance for the null model.

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.glm2 predict.orch.glm2 print.summary.orch.glm2 oracle.model.matrix

---

summary.orch.lm2 *Summary for the high performance linear regression, for the class* orch.lm2.

Description

Summary for the high performance linear regression, for the class orch.lm2.

Usage

```
summary.orch.lm2(object, ...)
```

Arguments

object	an object of class "orch.lm2", returned by a call to orch.lm2.

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.lm2 predict.orch.lm2 print.summary.orch.lm2 oracle.model.matrix

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

as.matrix.orch.drm *Collect a distributed matrix into an R matrix*

Description

Collect a distributed matrix into an R matrix

Usage

```
as.matrix.orch.drm(x)
```

Arguments

x	A distributed matrix

Value

an R dense matrix

Examples

```
dec <- orch.dssvd(data, formula = ~ . - 1, k = 2L, saveLoc = "svdOut")
cf <- coef(dec)

as.matrix(cf$U)
```

---

as.matrix.orch.mx *Tranform an in-memory Java matrix to an R matrix*

Description

Tranform an in-memory Java matrix to an R matrix

Usage

```
as.matrix.orch.mx(x)
```

Arguments

x	an in-memory Java matrix

Value

an R dense matrix

---

coef.orch.dspca *Overloaded coef() for D-SPCA*

Description

Overloaded coef() for D-SPCA

Usage

```
coef.orch.dspca(object, ...)
```

Arguments

object	The DSPCA "model"

Value

An R list with (mu, U, V, sigma)

Examples

```
data <- hdfs.put(iris)

dec <- orch.dspca(formula = ~ . - 1, data = data, k = 2L)

cf <- coef(dec)
as.matrix(cf$U)
```

coef.orch.dssvd *coef() for DSSVD "model"*

Description

coef() for DSSVD "model"

Usage

```
coef.orch.dssvd(object, ...)
```

Arguments

object	A DSSVD "model"

Value

An R list of (U,V,sigma)

Examples

```
data <- hdfs.put(iris)

dec <- orch.dssvd(formula = ~ . - 1, data = data, k = 2L, saveLoc = "svdOut",
  verbose = FALSE, overwrite = TRUE)

cf <- coef(dec)
cf$s
```

coef.orch.elm *ELM coef()*

Description

pick model parameters off the ELM model

Usage

```
coef.orch.elm(object, ...)
```

Arguments

object	the ELM model

Value

A list of the model parameter tensors

Examples

```
model <- orch.elm.load(hdfsLoc)
cfs <- coef(model)
```

coef.orch.helm *HELM coef()*

Description

Pick model parameters off the H-ELM model

Usage

```
coef.orch.helm(object, ...)
```

Arguments

object	The ELM model

Value

A list of the model parameter tensors

Examples

```
model <- orch.helm.load(hdfsLoc)
cfs <- coef(model)
```

dim.orch.drm *dim() for a distributed matrix*

Description

Obtain the dimensions of a distributed matrix

Usage

```
dim.orch.drm(x)
```

Value

The dimensions of ‘x‘

Examples

```
dec <- orch.dssvd(data, formula = ~ . - 1, k = 2L, saveLoc = "svdOut")
cf <- coef(dec)

dim(cf$U)
```

`dim.orch.mx` *dim() overload for in-memory Java matrix*

Description

Obtain the dimensions of an in-memory Java matrix

Usage

```
dim.orch.mx(x)
```

Arguments

`x`	An in-memory Java matrix

Value

The dimensions of 'x'

`foldInMx.orch.dspca` *SPCA fold-in (output: DRM)*

Description

SPCA fold-in (output: DRM)

Usage

```
foldInMx.orch.dspca(object, newdata, supplemental = NULL,
  verbose = TRUE, numPartitions = 1L)
```

Arguments

`object`	The DSPCA model.
`newdata`	New observation data. The supported input types are as follows: • 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. • An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using `orch.jdbc(...)`.
`supplemental`	In this implementation this parameter is not used.
`verbose`	If TRUE, provide more formula output to standard output.
`numPartitions`	The number of partitions to re-partition into (if positive).

Details

The fold-in PCA operation computes aproximation of new datapoints folded into PCA space as

$$\tilde{\mathbf{U}}_k \mathbf{\Sigma}_k \approx \left(\tilde{\mathbf{A}} - \boldsymbol{\mu}\right) \mathbf{V}_k,$$

where $\tilde{\mathbf{A}}$ is the matrix induced by new observations; $\tilde{\mathbf{U}}_k$ is the matrix corresponding to new observations in the scaled PCA space. Note that result of this routine is unscaled PCA space.

Value

A distributed matrix of new observations folded into PCA space.

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

Examples

```
data <- hdfs.put(iris)

dec <- orch.dspca(formula = ~ . - 1, data = data, k = 2L)

r <- foldInMx.orch.dspca(dec, data)
as.matrix(r)
```

---

foldIn.orch.dspca *D-SPCA fold-in (output: Spark data frame)*

Description

D-SPCA fold-in (output: Spark data frame)

Usage

```
foldIn.orch.dspca(object, newdata, supplemental = NULL,
  predictColPrefix = ".predict", verbose = TRUE,
  numPartitions = 1L)
```

Arguments

`object`	The DSPCA model.
`newdata`	New observation data. The supported input types are as follows: • 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. • An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using `orch.jdbc(...)`.
`supplemental`	The input columns to be joined with the results.
`predictColPrefix`	Column name prefix for the columns mapped into data frame attributes.
`verbose`	If TRUE provide more formula output to standard output.
`numPartitions`	The number of partitions to re-partition into (if positive).

Details

The fold-in PCA operation computes aproximation of new datapoints folded into PCA space as

$$\tilde{\mathbf{U}}_k \mathbf{\Sigma}_k \approx \left(\tilde{\mathbf{A}} - \boldsymbol{\mu}\right) \mathbf{V}_k,$$

where $\tilde{\mathbf{A}}$ is the matrix induced by new observations; $\tilde{\mathbf{U}}_k$ is the matrix corresponding to new observations in the scaled PCA space. Note that result of this routine is unscaled PCA space.

Value

A Spark 'DataFrame' contianing 'supplemental' columns concatenated with PCA component columns. The PCA component columns are named per 'predictColPrefix'_i, where 'i' is the component number, starting with 1.

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

Examples

```
data <- hdfs.put(iris)

dec <- orch.dspca(formula = ~ . - 1, data = data, k = 2L)

r <- foldIn.orch.dspca(dec, data, supplemental = "Species")
r$show()
```

`orch.dspca` *D-SPCA*

Description

Run formula transform + DSPCA. Produce predictMetadata + SVD artifacts (as requested).

Usage

```
orch.dspca(formula, data, k, p = 15L, q = 0L,
  formU = TRUE, formV = TRUE, saveLoc = NULL,
  verbose = FALSE, numPartitions = 0L, overwrite = FALSE)
```

Arguments

`formula`	R formula to use.
`data`	The dataset to be converted to input matrix. The supported input types are as follows: • 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. • An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using `orch.jdbc(...)`.
`k`	SSVD's reduced rank (perhaps no more than 200..500).
`p`	Oversampling, default: 15.
`q`	The number of power iterations. Suggested values are 0, 1 or 2. Having more than 0 power iterations may significantly increase the computational cost.

`formU`	If TRUE, form output matrix U
`formV`	If TRUE, form output matrix V
`saveLoc`	If not NULL, the HDFS location to save decomposition results into.
`verbose`	If TRUE, verbose output
`numPartition`	If greater than 0, the number of Spark partitions to repartition the formula output into before feeding into D-SSVD solver.
`overwrite`	Whether to overwrite the `saveLoc` directory, if it exists.

Details

The algorithm runs formula transformation to produce input matrix (A) and applies distributed SPCA (stochastic PCA).

The SPCA algorithm flow is equivalent to subtracting colmeans from rows of observatisons and then running stochastic SVD (S-SVD) on it (althout it is not doing exactly that verbatim):

$$\left(\mathbf{A} - \mathbf{1}\boldsymbol{\mu}^\top\right) \approx \mathbf{U}_k\boldsymbol{\Sigma}\mathbf{V}_k^\top,$$

where k is the rank of the SVD decomposition: $\mathbf{A} \in R^{m\times n}$, $\mathbf{U}_k \in R^{m\times k}$, $\mathbf{V} \in R^{n\times k}$; and $\boldsymbol{\mu}$ is the colmeans of $\mathbf{A}$:

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{A}_{i*} = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i.$$

Subsequently, U_k and V_k contains first k singular vectors, and $\boldsymbol{\Sigma}_k$ contains first k singular values of the S-SVD.

Thus, $U_k\Sigma_k$ (or U_k) correspond to original data points converted to PCA (or normalized PCA) spaces; and V_k can be used for subsequent fold-in(s) of new observations into the PCA (normalized PCA) spaces obtained by the original decomposition.

The computed SPCA model is thus $\{\boldsymbol{\mu}, \mathbf{U}_k, \mathbf{V}_k, \boldsymbol{\sigma}\}$ plus information related to formula transformation.

Value

DSPCA model: formula metadata + (mu, U, V, sigma)

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

Examples

```
data <- hdfs.put(iris)

dec <- orch.dspca(formula = ~ . - 1, data = data, k = 2L)
cf <- coef(dec)
as.matrix(cf$U)

r <- foldIn.orch.dspca(dec, data, supplemental = "Species")
r$show()
```

---

orch.dssvd.load *Load the D-SSVD model from HDFS*

Description

Load the D-SSVD model from HDFS

Usage

```
orch.dssvd.load(loc)
```

Arguments

loc	The location of HDFS model on HDFS.

Value

DSSVD "model": formula metadata + (U,V, Sigma).

Examples

```
data <- hdfs.put(iris)

dec <- orch.dssvd(formula = ~ . - 1, data = data, k = 2L, saveLoc = "svdOut",
  verbose = FALSE, overwrite = TRUE)

decClone <- orch.dssvd.load("svdOut")
cf <- coef(decClone)
cf$s
```

`orch.dssvd` *D-SSVD algorithm*

Description

D-SSVD algorithm

Usage

```
orch.dssvd(formula, data, k, p = 15L, q = 0L,
  formU = TRUE, formV = TRUE, saveLoc = NULL,
  verbose = FALSE, numPartitions = 0L, overwrite = FALSE)
```

Arguments

`formula`	R formula to use
`data`	The dataset to be converted to input matrix. The supported input types are as follows: • 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. • An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using `orch.jdbc(...)`.
`k`	SSVD's reduced rank (perhaps no more than 200..500).
`p`	Oversampling, default: 15.
`q`	The number of power iterations. Suggested values are 0, 1 or 2. Having more than 0 power iterations may significantly increase the computational cost.
`formU`	If TRUE, form output matrix U.
`formV`	if TRUE, form output matrix V.
`saveLoc`	If not NULL, the HDFS location to save decomposition results into.
`verbose`	If TRUE, verbose output.
`numPartition`	If greater than 0, the number of Spark partitions to repartition the formula output into before feeding into D-SSVD solver.
`overwrite`	Whether to overwrite the `saveLoc` directory, if it exists.

Details

The algorithm computes reduced, approximate k-rank SVD decomposition

$$\mathbf{A} \approx \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top,$$

where $\mathbf{A} \in R^{m \times n}$ is the input matrix formed by applying R formula to the data frame input; $\mathbf{U}_k \in R^{m \times k}$ and $\mathbf{V}_k \in R^{n \times k}$ are orthonormal matrices containing first k left and right singular vectors; $\mathbf{\Sigma}_k \in R^{k \times k}$ is diagonal matrix containing k singular values $\{\sigma_i : i = 1, 2, \ldots k\}$. Alternatively, we denote information carried by $\mathbf{\Sigma}$, by its diagonal vector $\boldsymbol{\sigma}$.

Value

DSSVD model: formula metadata + (U, V, Sigma)

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

Examples

```
data <- hdfs.put(iris)

dec <- orch.dssvd(formula = ~ . - 1, data = data, k = 2L, saveLoc = "svdOut",
  verbose = FALSE, overwrite = TRUE)

cf <- coef(dec)
as.matrix(cf$U)
```

---

orch.elm.load *ELM model load*

Description

load the ELM model off HDFS.

Usage

```
orch.elm.load(loc)
```

Arguments

loc	The hdfs location (directory) to load the model from.

Value

The loaded ELM model

Examples

```
data <- hdfs.put(iris)

# create and save a model
model <- orch.elm(formula = Species ~ . - 1, data = data, zScoreX = TRUE,
  l = 10, lambda = 1e-12, saveLoc = "example-elm", overwrite = TRUE)

## Not run:
  # The current Spark session can be disconnected using spark.disconnect().
  # In a new Spark session, you can load this model again.

## End(Not run)

# load the model and predict new observations
model <- orch.elm.load("example-elm")
predOut <- predict(model, newdata = data, supplemental = "Species")
predOut$show()
```

`orch.elm` *ORCH elm fit using formula.*

Description

ORCH elm fit using formula.

Usage

```
orch.elm(formula, data, zScoreX = FALSE, zScoreT = FALSE,
  l, lambda, l1 = FALSE, l1FistaIterations = 50,
  g = "tanh", saveLoc = NULL, verbose = FALSE,
  numPartitions = -1, overwrite = FALSE)
```

Arguments

`formula`	The R formula to be used.
`data`	The input data. The supported input types are as follows: • 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. • An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using `orch.jdbc(...)`.
`zScoreX`	If true, normalize the predictors upon the application of formula transformations.

zScoreT	If true, normalize the target upon the application of formula transformations.
l	The size of the hidden layer.
lambda	The regularization of the output ELM layer fit.
l1	If true, use L1 regularization (LASSO), otherwise use L2 regularization (ridge).
l1FistaIterations	The number of Fista iterations (default 50) if L1 regularization is requested.
g	The hidden layer activation function. Currently, one of: 'linear', 'sigmoid', 'tanh'.
saveLoc	*optional* HDFS location to save the resulting model tree.
verbose	If true, provide more formula output to standard output.
numPartitions	If positive, repartition formula output into this many partitions before handing over to the fitter algorithm. This allows manipulating degrees of parallelism during fitting algorithm execution.
overwrite	Whether to overwrite the saveLoc directory, if it exists.

Details

For information on ELM technique, see:

Huang, Guang-Bin, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: theory and applications." Neurocomputing 70.1 (2006): 489-501.

For information on L1 FISTA fitting used in this method, see:

Beck, Teboulle. A fast iterative shrinkage-thresholding algorithm with application to wavelet- based image deblurring.

Use cases: Multiclass classifier, regression.

Model persistence via HDFS, can be shared between different user sessions.

Value

A trained ELM model.

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

Examples

```
data <- hdfs.put(iris)

model <- orch.elm(formula = Species ~ . - 1, data = data, zScoreX = TRUE,
  l = 10, lambda = 1e-12)

summary(model)
cfs <- coef(model)
names(cfs)

#collect coefficient matrix to front end
as.matrix(cfs$Beta)

# predict new observations
predOut <- predict(model, newdata = data, supplemental = "Species")
predOut$show()
```

---

orch.helm.load *HELM model load*

Description

load the HELM model off HDFS.

Usage

```
orch.helm.load(loc)
```

Arguments

loc	The hdfs location (directory) to load the model from.

Value

The loaded HELM model

Examples

```
data <- hdfs.put(iris)

# create and save the model
model <- orch.helm(data, formula = Species ~ . - 1, zScoreX = TRUE,
  l = c(10L, 50L), lambdaAEnc = 1e-3, lambdaELM = 1e-9,
  saveLoc = "example-helm", overwrite = TRUE)

## Not run:
  # The current Spark session can be terminated using spark.disconnect().
```

```
  # In a new Spark session, you can load this model again.

## End(Not run)

# load the model and predict new observations
model <- orch.helm.load("example-helm")
predOut <- predict(model, newdata = data, supplemental = "Species")
predOut$show()
```

`orch.helm` *H-ELM fit*

Description

H-ELM fit

Usage

```
orch.helm(formula, data, zScoreX = FALSE,
  zScoreXRows = FALSE, zScoreT = FALSE, l, lambdaAEnc,
  lambdaELM, useL1inAE = TRUE, l1FistaIterations = 50L,
  g = "linear", s = 1, saveLoc = NULL, verbose = FALSE,
  numPartitions = 0L, overwrite = FALSE)
```

Arguments

`formula`	The R formula to be used.
`data`	The input data. The supported input types are as follows: • 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. • An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using `orch.jdbc(...)`.
`zScoreX`	If true, normalize the predictors upon the application of formula transformations.
`zscoreXRows`	If true, row-normalize the input.
`zScoreT`	If true, normalize the target upon the application of formula transformations.
`l`	The sizes of the hidden layers. The first n-1 are the sizes of ELM-AE layers; the last number is the size of the last ELM layer.
`lambdaAenc`	The regularization rates of the AEs (must be of size n-1).
`lambdaELM`	The regularization rate of the final ELM fit. This is always L2.
`useL1inAE`	If true, use L1 regularization (Lasso); otherwise, use the L2.
`l1FistaIterations`	The number of Fista iterations (default 50) if L1 regularization is requested.

`g`	The AE hidden layer activation function. Currently, one of: 'linear', 'sigmoid', 'tanh'.
`s`	The ELM pre-scaling factor, defualt 1.0 (no effect).
`saveLoc`	*optional* HDFS location to save the resulting model tree.
`verbose`	If true, provide more formula output to standard output.
`numPartitions`	If positive, repartition formula output into this many partitions before handing over to the fitter algorithm. This allows manipulating degrees of parallelism during fitting algorithm execution.
`overwrite`	Whether to overwrite the `saveLoc` directory, if it exists.

Details

H-ELM references:

* [H-ELM] Tang et. al. Extreme Learning Machine for Mutilayer Perceptron

* [FISTA] Beck, Teboulle. A fast iterative shrinkage-thresholding algorithm with application to wavelet-based image deblurring

* [ELM-ML] Kasun et. al. Representational learning with ELM for Big Data

Deviations from vanilla method and clarifications

1. Interlayer rescaling

What was not clearly articulated in the paper is that rescaling between layers is performed per

$$f_{AE}\left(\mathbf{H}\right) = \left(\mathbf{H} - \mathbf{1}\boldsymbol{c}_{min}^{\top}\right) \circ \left[\mathbf{1}\left(\boldsymbol{c}_{max} - \boldsymbol{c}_{min}\right)^{\top}\right]^{\circ -1},$$

where $\boldsymbol{c}_{min}$ and $\boldsymbol{c}_{max}$ are column-wise minimums and maximums of an AE's (autoencoder's) hidden layer output $\mathbf{H}$.

Further, to preserve intermediate output accumulated sparsity, we replaced that normalization with one centered around 0:

$$f_{AE}^{*}\left(\mathbf{H}\right) = \mathbf{H} \circ \left(\mathbf{1}\boldsymbol{c}_{\text{maxabs}}^{\top}\right)^{\circ -1},$$

where $c_{\text{maxabs},i} = \max\left[\text{sgn}\left(\mathbf{H}_{*i}\right)\mathbf{H}_{*i}\right]\ \forall i = 1, 2, \ldots n$ is the maximum absolute value in the i-th column of the matrix $\mathbf{H}$.

2. No image prep by default

Additionally, the original publication's implementation applied row-wise normalization of the input, which is only beneficial in case of black-and-white equally ranged pixel data (simple autocontrast of sorts), as a part of the preparation. This is by default disabled by the parameter 'zscoreXRows' . To re-enable this behavior, set 'zscoreXRows' to TRUE.

Use cases

The method handles supervised regression and multiclass targets. The type of problem is determined by the target in the R formula.

Unsupervised formulas are rejected with an error.

Prediction output is provided in the form of Spark 'DataFrame'.

Value

A trained HELM model

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

Examples

```
data <- hdfs.put(iris)

model <- orch.helm(formula = Species ~ . - 1, data = data, zScoreX = TRUE,
  l = c(10L, 50L), lambdaAEnc = 1e-3, lambdaELM = 1e-9,
  saveLoc = "example-helm", overwrite = TRUE)

summary(model)
cfs <- coef(model)
names(cfs)

as.matrix(cfs$Beta1)
as.matrix(cfs$elmQ)

predOut <- predict(model, data, supplemental = "Species")
predOut$show()
```

---

orch.mpiAvailable *MPI subsystem check*

Description

Check if proper MPI subsystem is available

Usage

```
orch.mpiAvailable()
```

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

`orch.mpi.cleanup` *MPI Cleanup*

Description

Clean up stuck MPI processes and shared memory segments on the cluster using Spark tasks.

Usage

```
orch.mpi.cleanup(global = FALSE)
```

Arguments

`global`	If TRUE, will attempt to cleanup the ENTIRE cluster. If FALSE, we only attempt to cleanup processes belonging to the current OS user.

Details

This is only necessary as a last recourse if less-than-graceful crash occurred during MPI phase execution, AND the driver process (which otherwise automatically cleans up failed MPI jobs) has failed as well.

Value

The list of cleanup task errors grouped by host (if any). Errors are not necessarily an indication of objective failure.

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:
MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.
MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:
Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.
In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.
The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

`ORCH_MPI_LIBS` *ORCHmpi system control environment variable.*

Description

This control environment variable is used to locate the MPI libraries on the cluster system. MPI should be available at the same location on all the nodes of the cluster. By default, it is set in your Renviron.site by the client installer to point to the pre-built MPI library, which is available at '/usr/lib64/R/lib/mpich/lib'.

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_MPI_MPIEXEC ORCH_MPI_MAX_GRID_SIZE

Examples

```
## Not run:
  csh: setenv ORCH_MPI_LIBS /usr/lib/mpi/lib
  bash: export ORCH_MPI_LIBS=/usr/lib/mpi/lib

## End(Not run)
```

ORCH_MPI_MAX_GRID_SIZE *ORCHmpi system control environment variable.*

Description

This control environment variable is used to set the maximum number of MPI workers (not counting the leader process) that MPI computation may spawn on the cluster per submission. It is recommended to set this to an integer value, with maximum being no more than 60 percent of available cluster CPU cores.

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_MPI_LIBS ORCH_MPI_MPIEXEC

Examples

```
## Not run:
  csh: setenv ORCH_MPI_MAX_GRID_SIZE 50
  bash: export ORCH_MPI_MAX_GRID_SIZE=50

## End(Not run)
```

ORCH_MPI_MPIEXEC *ORCHmpi system control environment variable.*

Description

This control environment variable is used to locate the 'mpiexec' program on the cluster system. MPI should be available at the same location on all the nodes of the cluster. By default, it is set in your Renviron.site by the client installer to point to the pre-built MPI library, which is available at '/usr/lib64/R/lib/mpich/bin/mpiexec'.

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_MPI_LIBS ORCH_MPI_MAX_GRID_SIZE

Examples

```
## Not run:
  csh: setenv ORCH_MPI_MPIEXEC /usr/lib/mpi/bin/mpiexec
  bash: export ORCH_MPI_MPIEXEC=/usr/lib/mpi/bin/mpiexec

## End(Not run)
```

---

orch.mpi.options *MPI Options*

Description

Set MPI stage execution options.

Usage

```
orch.mpi.options(maxGridSize = NULL, mpiRetries = NULL,
  mpiCleanupTasks = NULL)
```

Arguments

maxGridSize	Integer, maximum number of MPI workers (not counting the leader process) that MPI computation may spawn on the cluster per submission.
mpiRetries	Number of retries if MPI computation fails. 0 means do not try again, 1 means try again once if the initial submission fails, etc.
mpiCleanupTasks	The number of spark tasks for MPI cleanup to spawn in `orch.mpi.cleanup()`.

orch.scalapackAvailable *Scalpack subsystem check*

Description

Scalpack subsystem check

Usage

```
orch.scalapackAvailable()
```

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

predict.orch.elm *ELM predict*

Description

Score ELM model (overloaded)

Usage

```
predict.orch.elm(object, newdata, supplemental = NULL,
  verbose = TRUE)
```

Arguments

`object`	The ELM model produced by orch.elm().
`newdata`	The new observation data. The supported input types are as follows: • 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. • An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using `orch.jdbc(...)`.
`supplemental`	The array of names for supplemental data to be joined with predictions; could be empty if no join with original dataset columns is desired.

Value

A Spark 'DataFrame' instance of scores joined by 'supplemental' columns (if any).

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

Examples

```
data <- hdfs.put(iris)

model <- orch.elm(data, formula = Species ~ . - 1, zScoreX = TRUE,
  l = 10, lambda = 1e-12)

# predict new observations
predOut <- predict(model, newdata = data, supplemental = "Species")
predOut$show()
```

`predict.orch.helm` *HELM predict*

Description

Score H-ELM model (overloaded)

Usage

```
predict.orch.helm(object, newdata, supplemental = NULL,
  verbose = TRUE, numPartitions = 1L)
```

Arguments

`object`	The H-ELM model produced by orch.helm().
`newdata`	The new observation data. The supported input types are as follows: • 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. • An `ore.frame` object, when connected to "HIVE" or "IMPALA" using `ore.connect`. • A Spark dataframe created using any other external method or Spark API directly. • An 'orch.jdbc' object created using `orch.jdbc(...)`.
`supplemental`	The array of names for supplemental data to be joined with predictions; could be empty if no join with original dataset columns is desired.

Value

A Spark 'DataFrame' instance of scores joined by 'supplemental' columns (if any)

Current MPI Limitations

Support for MPI integration is experimental.

Worker process lifecycle and resource requirements:

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous life cycle. Consequently, at the very least all MPI individual submission data must fit into machine cluster memory, comfortably.

MPI cohabitates resources along with Spark resource manager and may cause cluster oversubscription if resources are not properly allowed.

Limited sparse algebra support:

Currently, our MPI algorithms support dense algebra only. It means sparse and extra-sparse problems may likely not perform as well as dense problems, as they will have to be transformed into a dense problem first.

In particular, solutions using formula one-hot transformations of categorical variables with high category cardinalities, as well as interactions of such, may produce vectorization of extreme sparsity, and may suffer from the aforementioned significant problem expansion.

The support for sparse MPI algebra will be added in future releases. For now, it is recommended to avoid extra sparse scenarios with MPI based solvers.

Examples

```
data <- hdfs.put(iris)

model <- orch.helm(data, formula = Species ~ . - 1, zScoreX = TRUE,
  l = c(10L, 50L), lambdaAEnc = 1e-3, lambdaELM = 1e-9)

# predict new observations
predOut <- predict(model, newdata = data, supplemental = "Species")
predOut$show()
```

---

summary.orch.elm *ELM summary*

Description

Print out summary of the ELM model

Usage

```
summary.orch.elm(object, ...)
```

Arguments

object	the ELM model

Examples

```
model <- orch.elm(data, formula = Species ~ . - 1, zScoreX = TRUE, l = 10, lambda = 1e-12
summary(model)
```

---

summary.orch.helm *H-ELM summary*

Description

Print out summary of the HELM model

Usage

```
summary.orch.helm(object, ...)
```

Arguments

object	the H-ELM model

Examples

```
clonedModel <- orch.helm.load("example-helm")
summary(clonedModel)
```

`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