Package 'ibmdbR'

June 3, 2018

Version 1.50.0

Title IBM in-Database Analytics for R

Author Toni Bollinger [aut, cre],
Alexander Eckert [aut],
Michael Wurst [aut],
Craig Blaha [ctb] (documentation),
IBM Corporation [cph]

Maintainer Toni Bollinger <db2-analytics@de.ibm.com>

Depends R (>= 2.15.1), methods, RODBC, Matrix, arules

Imports MASS, rpart, rpart.plot, ggplot2

Suggests SparkR

Description Functionality required to efficiently use R with IBM(R) Db2(R) Warehouse offerings (formerly IBM dashDB(R)) and IBM Db2 for z/OS(R) in conjunction with IBM Db2 Analytics Accelerator for z/OS.
Many basic and complex R operations are pushed down into the database, which removes the main memory boundary of R and allows to make full use of parallel processing in the underlying database.

License GPL-3

LazyLoad Yes

NeedsCompilation no

Repository CRAN

Date/Publication 2018-06-03 15:17:31 UTC

R topics documented:

`ibmdbR-package` *IBM In-Database Analytics*

Description

In-database analytics functions operate directly on data in a database, rather than requiring that the data first be extracted to working memory. This lets you analyze large amounts of data that would be impractical or impossible to extract. It also avoids security issues associated with extracting data, and ensures that the data being analyzed is as current as possible. Some functions additionally use lazy loading to load only those parts of the data that are actually required, to further increase efficiency.

This package also contains a data structure called a `ida.list`, which you can use to store R objects in the database. This simplifies the sharing of R objects among users. Each user is assigned two tables for R object storage: a private table, to which only that user has access, and a public table, which can be read by other users. Use a IDA list to generate a pointer to either of these tables, and use the pointer to list, store, or retrieve R objects.

`as.ida.data.frame` *Convert an R object to an IDA data frame*

Description

This function creates an IDA data frame `ida.data.frame` from a local R `data.frame` by creating a table in the database.

Usage

```
as.ida.data.frame(x, table=NULL, clear.existing=FALSE, case.sensitive=TRUE,
                  rownames=NULL, dbname=NULL, asAOT=FALSE)
```

Arguments

`x`	The name of the input object that is to be converted to a IDA data frame.
`table`	The name of the database table that is to be created to hold the contents of the IDA data frame. The specified name is folded to uppercase. If this parameter is not specified, a name is generated automatically.
`clear.existing`	If the `table` parameter is specified, this parameter indicates whether the existing table is to be dropped (TRUE) or whether the `as.ida.data.frame` statement is to be ignored and a warning message issued (FALSE).
`case.sensitive`	If the `table` parameter is specified for an existing table, this parameter specifies whether the column names in that table name are to be treated as case-sensitive (TRUE) or not case-sensitive (FALSE).
`rownames`	The name of the column for the unique row id. If the value of this parameter is NULL, this column is not added to the output table.
`dbname`	DB2 for z/OS only parameter: the name of the database where the table should be created in.
`asAOT`	DB2 for z/OS only parameter: the table should be created as an "accelerator only table".

Value

A IDA data frame that points to the newly created table.

See Also

`as.data.frame`

Examples

```
## Not run:

#Add an ID column to iris
iris2 <- iris
iris2$ID <- 1:150

#Upload it and overwrite if already exists
idf <- as.ida.data.frame(iris2,"IRIS",clear.existing=T)


## End(Not run)
```

`ida.col.def methods` *Available methods for ida.col.def*

Description

`ida.col.def` objects are used to define new columns of a `ida.data.frame` based on existing ones.

For details see the documentation of `ida.data.frame`.

`ida.data.frame methods`
Available methods for ida.data.frame

Description

`ida.data.frame` objects provide many methods that will behave exactly like or very similar to methods defined on a regular `data.frame`. The following is a list of currently supported methods: as.data.frame, sd, max, mean, min, length, print, names, colnames, summary, NROW, NCOL, dim, var, head, hist, cor, cov. Furthermore, the $ and [] operators allow you to select columns and rows and the $<- operator will allow you to add columns. For details see the documentation of `ida.data.frame`.

`ida.data.frame, is.ida.data.frame`
Create an IDA data frame

Description

This function creates an IDA data frame (that is, an object of the class `ida.data.frame`). It does not store any data in local memory, but aggregates metadata used to determine the exact table subset (columns - `SELECT` clause; and/or rows - `WHERE` clause) and creates a pointer to a table located in the database.

Usage

```
ida.data.frame(table)
is.ida.data.frame(x)
```

Arguments

`table`	Name of a table or view in the current database.
`x`	An `ida.data.frame` object.

Details

The argument table must be a valid table or view name and the table/view must exist.

If schema or table are set in quotes, they will be treated case sensitive otherwise they are automatically converted to the default schema of the database. Columns are always treated case sensitive.

A subset of columns and/or rows may be specified using the indexing operator `[]` (which is translated to the `SELECT` clause for columns and/or the `WHERE` clause for rows). Note that columns are treated case sensitive.

One limitation is that rows cannot be selected using their numbers. Instead, you must specify value-based conditions, for example `d[d$ID > 10,]` which means "all rows where the value of the first column is greater than 10". The `$` operator may be also used to select an `ida.data.frame` column.

You can also add and alter columns in an ida.data.frame. Currently, a limited set of functions and operators is supported to define columns based on other columns. The following is supported:

- Arithmetic operators are +,-,/,*,^
- Mathematical functions are abs, sqrt, log, log10, exp, floor, round, ceiling
- Casting functions: as.numeric, as.integer, as.character
- Comparison and logical operators: <,<=,>,>=,!=,==,!,&,|
- Conditional functions: ifelse
- Special functions: is.db.null (checks whether column value is NULL in the table)

There are several rules for adding columns:

1. You can not combine columns from different tables or from ida.data.frames that have different WHERE conditions.
2. You cannot add a column to an ida.data.frame that was defined on columns from another ida.data.frame
3. You can only add columns that evaluate to non-logical, atomic values

The package does basic type checking to enforce these rules, however, it is still possible that the database will refuse a query that was not properly defined.

`is.ida.data.frame` checks if the given object's class is `ida.data.frame`.

Value

`ida.data.frame` returns an IDA data frame.

`is.ida.data.frame` returns a logical value that indicates whether the specified object is an IDA data frame.

Examples

```
## Not run:
idf <- ida.data.frame('IRIS')
is.ida.data.frame(idf)

#Select only certain rows or columns
#The following creates an ida.data.frame that only selects rows with
#Species=='setosa' and the first three columns of the table
idf2 <- idf[idf$Species=='setosa',1:3]

#Define new columns based on existing ones
idf$SepalLengthX <- idf$SepalLength+1
idf$SepalLengthY <- ifelse(idf$SepalLengthX>4.5,idf$ID,10)

#Take a look at the newly defined columns
head(idf)



## End(Not run)
```

---

ida.list *Store and retrieve R objects in the database*

Description

A user can elect to store R objects in a database table rather than storing them in a workstation file system. This makes it easier for users to share objects, and simplifies backup tasks.

Each user has two R object storage tables:

- A private table, for objects that other users are not to be able to access

- A public table, for objects that other users are to be able to read

Use the `ida.list` function to create a pointer to either of your own R object storage tables, or to the public R object storage table of another user. You can then use the pointer to store objects in or retrieve objects from the corresponding table. (If the table belongs to another user you can only retrieve objects from it, not store objects in it.)

Please note that whether public tables might not have effect on databases that do not allow to set permissions accordingly, for instance, in multi-tenant environments. To enable the sharing of objects in DB2, an administrator needs to first create a role names R_USERS_PUBLIC and assign it to all users who should be allowed to share objects. For Db2, roles will be setup automatically if in the scope of the plan.

Usage

```
ida.list(type='public',user=NULL)
```

Arguments

`type`	The type (private or public) of the table. You can specify 'private' only if `user` is NULL or is set explicitly to your own user ID.
`user`	The user ID of the owner of the R object storage table. If set to NULL, the user ID is that of the current user. The user ID is treated case-sensitive.

Value

A pointer to an R object storage table.

Examples

```
## Not run:
# Create a pointer to the private R object storage table of the current user.
myPrivateObjects <- ida.list(type='private')

# Use the pointer created in the previous example to store a series of numbers in an object with
# the name 'series100' in the private R object storage table of the current user.
myPrivateObjects['series100'] <- 1:100

# Retrieve the object with the name 'series100' from the
# private R object storage table of the current user.
x <- myPrivateObjects['series100']

# Delete the object with name 'series100' from the
# private R object storage table of the current user.

myPrivateObjects['series100'] <- NULL

# List all objects in the private R object storage table of the current user.
names(myPrivateObjects)

# Return the number of objects in the private R object storage table of the current user.
length(myPrivateObjects)
```

```
# Create a pointer to the public R object storage table of the current user.
myPublicObjects <- ida.list(type="public")


## End(Not run)
```

`ida.list methods` *Available methods for ida.list*

Description

`ida.list` objects provide methods that will behave exactly like or very similar to methods defined on a regular `list`. The following methods are currently supported: length, names, print.

For details see the documentation of `ida.list`.

`idaArule` *Association Rule Mining*

Description

This function calculates association rules on a database table.

Usage

```
idaArule(
data,
tid,
item,
maxlen=5,
maxheadlen=1,
minsupport=NULL,
minconf=0.5,
nametable=NULL,
namecol=NULL,
modelname=NULL
)

idaApplyRules(modelname, newdata, tid, item, nametable=NULL, namecol=NULL, ...)
```

Arguments

data	An ida.data.frame object pointing to the data to be mined.
tid	Input table column that identifies the transactions by an id.
item	Input table column that identifies items in transactions.
maxlen	The maximum length of a rule. Must be two or larger.
maxheadlen	The maximum length of the rule head.
minsupport	The minimal support of a rule to be considered.
minconf	The minimal confidence of a rule to be considered.
nametable	A database table containing a mapping between the items in the input table and their name. The table must contain at least two columns, the first column is named as the column indicated in the item parameter and the second column is named as indicated in parameter namecol.
namecol	The name of the column containing the item name in case nametable was specified.
modelname	The name of the model in-database. If NULL, it is automatically generated.
newdata	A table to which to apply the rules.
...	Additional stored procedure parameters.

Details

idaArule finds association rules in transactional data. The input data must be in transactional format, thus each row of the table contains exactly one item and an identifier of which transaction this item is assigned to. These two columns need to be specified using the tid and item parameters. If the items are referred to with numeric IDs in the transaction table, it is often useful to add a name mapping to produce rules that contain names instead of item IDs. This can be achieved by setting the parameters nametable and namecol.

Models are stored persistently in database under the name modelname. Model names cannot have more than 64 characters and cannot contain white spaces. They need to be quoted like table names, otherwise they will be treated upper case by default. Only one model with a given name is allowed in the database at a time. If a model with modelname already exists, you need to drop it with idaDropModel first before you can create another one with the same name. The model name can be used to retrieve the model later (idaRetrieveModel).

idaApplyRules applies a rule model stored in the database to a table with transactions.

Value

idaArule returns an object of class rules compatible with the packages arules and arulesViz
idaApplyRules returns an object of class ida.data.frame, pointing to a table that contains a mapping between transaction IDs and matched rules.

Examples

```
## Not run:

idf <- ida.data.frame("GOSALES.ORDER_DETAILS")
```

```
r <- idaArule(idf,tid="ORDER_NUMBER",item="PRODUCT_NUMBER",minsupport=0.01)

inspect(r)

applyResult <- idaApplyRules(idaGetModelname(r),idf,"ORDER_NUMBER","PRODUCT_NUMBER")


## End(Not run)
```

`idaConnect, idaClose` *Open or closes a IDA database connection*

Description

These functions are used to open or close an existing IDA database connection.

Usage

```
idaConnect(dsn, uid = "", pwd = "", conType = "odbc",
  dsnLookup = c("auto", "default", "store"), ...)

idaClose(idaConn, conType = "odbc")
```

Arguments

`dsn`	The DSN of the data base.
`uid`	The user name.
`pwd`	The password.
`conType`	The connection type.
`dsnLookup`	This parameter only is used when ibmdbR is loaded in an RStudio instance of IBM Data Science Experience. Per default ("auto") ibmdbR automatically detects if the provided dsn value is an usual DSN string or the name of a connection in the local connection store. The lookup method can also be manually set to "default", if the dsn parameter should be treated as an usual DSN string. If set to "store" the connection store of RStudio on DSX will be used.
`...`	Additional arguments for DSN lookup.
`idaConn`	The connection object.

Details

Opens or closes a connection to a database. Currently, RODBC is used as underlying library, this might change, however, in the future.

Examples

```
## Not run:
#Connect locally
con <- idaConnect('BLUDB','','')

#Close the connection
idaClose(con)

## End(Not run)
```

`idaCreateView, idaDropView`
Create or drop a view

Description

Use these functions to create or drop a view that is based on a `ida.data.frame`.

Usage

```
idaCreateView(x, newColumn = NULL)
idaDropView(v)
```

Arguments

`x`	`ida.data.frame` for which a view is to be created.
`newColumn`	The expression specifying the column to be added.
`v`	Name of the view to be dropped.

Details

The `idaCreateView` function creates a view from the specified IDA data frame. The `idaDropView` function drops the specified view.

Value

The `idaCreateView` function returns the view name. The `idaDropView` function does not return a value.

Examples

```
## Not run:
idf <- ida.data.frame('IRIS')

#Create a view based on the IDA data frame
vname <- idaCreateView(idf)

#Drop the view
```

```
idaDropView(vname)

## End(Not run)
```

`idaDeleteTable, idaExistTable, idaGetValidTableName, idaIsView`
Miscellaneous tools

Description

These functions simplify working with database tables.

Usage

```
idaAppend(df, table)

idaDeleteTable(table)

idaExistTable(tableName)

idaIsView(tableName)

idaGetValidTableName(prefix="DATA_FRAME_")
```

Arguments

`df`	A `data.frame` object.
`table`	The name of a database table or an `ida.data.frame`.
`tableName`	The name of a database table.
`prefix`	Keyword used to specify the prefix of a table name.

Details

Use the `idaAppend` function to append a `data.frame` to the specified table.

Use the `idaDeleteTable` function to drop the specified table. The specified table must exist in the current database.

Use the `idaExistTable` function to determine whether the specified table exists in the database.

Use the `idaGetValidTableName` function to obtain a table name that is not yet in use. This name will be the specified or default prefix followed by a number, for example, data_frame_7.

Value

The idaDeleteTable function does not return a value.

The idaExistTable function returns a logical value (TRUE or FALSE) that indicates whether the specified table exists in the database.

The idaGetValidTableName function returns a string representing a table name.

Examples

```
## Not run:

#Check whether a table with a given name exists
idaExistTable('IRIS')

#Create a pointer to the table
idf <- ida.data.frame('IRIS')

#Obtain a unique table name for a copy
copyTableName <- idaGetValidTableName(prefix = "COPY_")

#Create a copy of the original table
idfCopy <- as.ida.data.frame(as.data.frame(idf),copyTableName)

#Delete the copy again
idaDeleteTable(copyTableName)


## End(Not run)
```

---

idadf,idaSave,idaUpdate

Query, store and update data in the database.

Description

These functions allow to query, store and update data in the database. Usually, it is easier to use idaQuery,ida.data.frame and as.ida.data.frame instead of these methods.

They can be useful, however, if an explicit connection object is needed, e.g. if there are several connections to different databases.

Usage

```
idadf(idaConn, query)
idaSave(idaConn, dfrm, tblName = "", rowName = "", conType = "odbc")
idaUpdate(db2Conn, updf, dfrm, idaIndex = "", conType = "odbc")
```

Arguments

idaConn	The IDA connection object.
db2Conn	The IDA connection object.
query	A query.
dfrm	A data.frame to store.
tblName	Name of the table to which to store the data.
rowName	Name of the row name column.
updf	Name of the table to update.
idaIndex	Name of the index column.
conType	Type of the connection.

Details

idadf,idaSave and idaUpdate are simple wrappers around the RODBC functions sqlQuery, sqlSave and sqlUpdate.

Usually, it is easier to use idaQuery,ida.data.frame and as.ida.data.frame instead of these methods.

See Also

idaQuery,ida.data.frame,as.ida.data.frame,sqlQuery,sqlSave,sqlUpdate

Examples

```
## Not run:
# create connection to DB
con <- idaConnect("BLUDB", "", "")

# create data.frame from table
df <- idadf(con, "SELECT * FROM IRIS")

# close the connection again
idaClose(con)

## End(Not run)
```

---

idaDivCluster *Hierarchical (divisive) clustering*

Description

This function generates a hierarchical (divisive) clustering model based on the contents of an IDA data frame (ida.data.frame) by applying recursively the K-means algorithm.

Usage

```
idaDivCluster(
    data,
    id,
    distance="euclidean",
    maxiter=5,
    minsplit=5,
    maxdepth=3,
    randseed=12345,
    outtable=NULL,
    modelname=NULL
)

## S3 method for class 'idaDivCluster'
print(x,...)
## S3 method for class 'idaDivCluster'
predict(object, newdata, id,...)
```

Arguments

data	An IDA data frame that contains the input data for the function. The input IDA data frame must include a column that contains a unique ID for each row.
id	The name of the column that contains a unique ID for each row of the input data.
distance	The distance function that is to be used. This can be set to "euclidean", which causes the squared Euclidean distance to be used, or "norm_euclidean", which causes normalized euclidean distance to be used.
maxiter	The maximum number of iterations to perform in the base K-means Clustering algorithm
minsplit	The minimum number of instances per cluster that can be split.
maxdepth	The maximum number of cluster levels (including leaves).
randseed	The seed for the random number generator.
outtable	The name of the output table that is to contain the results of the operation. When NULL is specified, a table name is generated automatically.
modelname	The name under which the model is stored in the database. This is the name that is specified when using functions such as idaRetrieveModel or idaDropModel.
object	An object of the class idaDivCluster to used for prediction, i.e. for applying it to new data.
x	An object of the class idaDivCluster to be printed.
newdata	An IDA data frame that contains the data to which to apply the model.
...	Additional parameters to pass to the print or predict method.

Details

The idaDivCluster clustering function builds a hierarchical clustering model by applying the K-means algorithm recursively in a top-down fashion. The hierarchy of clusters is represented in a

binary tree structure (each parent node has exactly 2 child nodes). The leafs of the cluster tree are identified by negative numbers.

Models are stored persistently in the database under the name `modelname`. Model names cannot have more than 64 characters and cannot contain white spaces. They need to be quoted like table names, otherwise they will be treated upper case by default. Only one model with a given name is allowed in the database at a time. If a model with `modelname` already exists, you need to drop it with `idaDropModel` first before you can create another one with the same name. The model name can be used to retrieve the model later (`idaRetrieveModel`).

The output of the print function for a idaDivCluster object is:

- A vector containing a list of centers
- A vector containing a list of cluster sizes
- A vector containing a list of the number of elements in each cluster
- A data frame or the name of the table containing the calculated cluster assignments
- The within-cluster sum of squares (which indicates cluster density)
- The names of the slots that are available in the idaDivCluster object.

Value

The idaDivCluster function returns an object of class `idaDivCluster`.

See Also

`idaRetrieveModel`, `idaDropModel`, `idaListModels`

Examples

```
## Not run:

#Create ida data frame
idf <- ida.data.frame("IRIS")

#Create a DivCluster model stored in the database as DivClusterMODEL
dcm <- idaDivCluster(idf, id="ID",modelname="DivClusterMODEL")

#Print the model
print(dcm)

#Predict the model
pred <- predict(dcm,idf,id="ID")

#Inspect the predictions
head(pred)


## End(Not run)
```

idaDropModel *Drop a predictive model from the database*

Description

Use this function to drop from the database a model that was created by using a function like idaNaiveBayes, idaLm, idaTree, idaArule or idaKMeans.

Usage

```
idaDropModel(modelname)
```

Arguments

modelname	The name of the predictive model to be dropped.

Examples

```
## Not run:

#Drop the model with the name KMEANSMODEL
idaDropModel("KMEANSMODEL");

## End(Not run)
```

idaGetModelName *Get the name of a model*

Description

Use this function to get the name under which a model is stored in-database. This function can be applied to objects returned by functions like idaNaiveBayes, idaKMeans or idaArule.

Usage

```
idaGetModelname(object)
```

Arguments

object	The object representing the model.

Value

The fully qualified name of the model, as stored in-database. This name is used, e.g. in conjunction with the idaRetrieveModel or with the idaDropModel function.

Examples

```
## Not run:

#Get the name of a model stored in variable km
modelname <- idaGetModelname(km)


## End(Not run)
```

---

idaGlm *Generalized Linear Models (GLM)*

Description

This function computes generalized linear models on the contents of an ida.data.frame.

Usage

```
idaGlm( form, data, id = "id", intercept = T, family = "bernoulli", family_param = -1,
          link = "logit", link_param = 1, maxit = 20, eps = 1e-3, tol = 1e-7,
          method = "irls", trials = NULL, incolumn = "", interaction = "",
          modelname = NULL, format = "glm", raw.resid = F, dropAfter = F, ...)

## S3 method for class 'idaGlm'
print(x, ...)
## S3 method for class 'idaGlm'
predict(object, newdata, id, outtable = NULL, ...)
```

Arguments

form	A formula object that describes the GLM to build.
data	An ida.data.frame object that stores the data to be used for GLM building.
id	The ID column name.
intercept	The intercept.
family	The type of error distribution. It can have one of the follwing values: "bernoulli", "gaussian", "poisson", "binomial", "negativebinomial", "wald", "gamma"
family_param	A family-specific parameter.
link	Type of the link function. It can have one of the follwoing values: "clog", "cloglog", "gaussit", "identity", "log", "logit", "oddspower", "power", "probit", and "sqrt". For Db2 for z/OS it can have the following values as well: "canbinom", "cangeom", "cannegbinom", "cauchit", "inverse", "invnegative", "invsquare", "loglog" .
link_param	Link parameter, 1 by default.
maxit	Maximum number of iterations. 20 by default.

eps	Maximum (relative) error used as a stopping criterion. This should be sufficiently small for the algorithm to work.
tol	The tolerance for the linear equation solver to consider a value equal to be zero. This should be sufficiently small for the algorithm to work.
method	Computing algorithm: either "irls" ("iteratively reweighted least square") or "psgd" ("parallel stochastic gradient descent").
trials	The input table column containing the number of trials for the binominal distribution. Ignored unless family is 'binomial'.
incolumn	Overwrite automatic creation of incolumn parameter and specify your own incolumn here.
interaction	Overwrite automatic creation of interaction parameter and specify your own interaction here.
modelname	Name of the model that will be created in the database. Will be created automatically if not specified.
format	Specify output format. Either "glm" for output looking like stats::glm or raw for downloading all results as data.frames.
raw.resid	If format equals "raw", whether to download the residuals or return NULL instead.
dropAfter	Whether to drop the results after downloading them as specified in format.
x	An idaGlm object.
object	An idaGlm object.
newdata	New data used for prediction as ida.data.frame.
outtable	The name of the table the results will be written in.
...	Additional parameters.

Details

For more details on the GLM algorithm and requirements to the data, please refer to the documentation of the nza..GLM stored procedure in the Netezza In-Database Analytics Reference Guide or Netezza In-Database Analytics Developers Guide.

Value

- The function idaGlm returns the generalized linear regression model of classes glm and idaGlm if format equals "glm" or a list of data.frames if format equals "raw".
- The functions print and summary have no return values.
- The function predict returns an ida.data.frame that contains the predicted values.

Examples

```
## Not run:
#Add isSetosa column to iris data frame
iris2 <- iris
iris2$isSetosa <- ifelse(iris2$Species=="setosa", 1, 0)
```

```
#Store the iris2 data frame in the IRIS2 table
idf <-as.ida.data.frame(iris2, table="IRIS2", clear.existing=T, rownames="ID")

#Calculate GLM model in-db
glm <- idaGlm(isSetosa~PetalLength+SepalLength*SepalWidth+PetalWidth, idf, id="ID")

#Print the model
print(glm)

#Apply the model to data
idf2 <- predict(glm, idf, "ID")

#Inspect the results
head(idf2)


## End(Not run)
```

`idaInit` *Initialize the In-Database Analytics functions*

Description

This function initializes the In-Database Analytics functions.

Usage

```
idaInit(con,jobDescription=NULL)
```

Arguments

`con`	An open RODBC connection.
`jobDescription`	Optional argument that allows to assign a description to the jobs submitted from the R session.

Details

Use an existing RODBC connection to initialize the IDA in-database analytics functions. All commands are sent through this connection.

Value

No value is returned.

Examples

```
## Not run:

#Initialize the IDA Analytics functions
con <- idaConnect('BLUDB','','')

#Initialize the in-database functionality
idaInit(con)


## End(Not run)
```

---

idaKMeans *k-means clustering*

Description

This function generates a k-means clustering model based on the contents of a IDA data frame (ida.data.frame).

Usage

```
idaKMeans(
data,
id,
k=3,
maxiter=5,
distance="euclidean",
outtable=NULL,
randseed=12345,
statistics=NULL,
modelname=NULL
)

## S3 method for class 'idaKMeans'
print(x,...)
## S3 method for class 'idaKMeans'
predict(object, newdata, id,...)
```

Arguments

data	An IDA data frame that contains the input data for the function. The input IDA data frame must include a column that contains a unique ID for each row.
id	The name of the column that contains a unique ID for each row of the input data.
k	The number of clusters to be calculated.

maxiter	The maximum number of iterations to be used to calculate the k-means clusters. A larger number of iterations increases both the precision of the results and the amount of time required to calculate them.
distance	The distance function that is to be used. This can be set to "euclidean", which causes the squared Euclidean distance to be used, or "norm_euclidean", which causes normalized euclidean distance to be used.
outtable	The name of the output table that is to contain the results of the operation. When NULL is specified, a table name is generated automatically.
randseed	The seed for the random number generator.
statistics	Denotes which statistics to calculate. Allowed values are "none","columns" and "all". If NULL, the default of the database system will be used.
modelname	The name under which the model is stored in the database. This is the name that is specified when using functions such as idaRetrieveModel or idaDropModel.
object	An object of the class idaKMeans to be used for prediction, i.e. for applying it to new data.
x	An object of the class idaKMeans to be printed.
newdata	A IDA data frame that contains the data to which to apply the model.
...	Additional parameters to pass to the print or predict method.

Details

The idaKMeans function calculates the squared Euclidean distance between rows, and groups them into clusters. Initial clusters are chosen randomly using a random seed, and the results are adjusted iteratively until either the maximum number of iterations is reached or until two iterations return identical results. Variables with missing values are set zero for distance calculation.

Models are stored persistently in database under the name modelname. Model names cannot have more than 64 characters and cannot contain white spaces. They need to be quoted like table names, otherwise they will be treated upper case by default. Only one model with a given name is allowed in the database at a time. If a model with modelname already exists, you need to drop it with idaDropModel first before you can create another one with the same name. The model name can be used to retrieve the model later (idaRetrieveModel).

The output of the print function for a idaKMeans object is:

- A vector containing a list of centers
- A vector containing a list of cluster sizes
- A vector containing a list of the number of elements in each cluster
- A data frame or the name of the table containing the calculated cluster assignments
- The within-cluster sum of squares (which indicates cluster density)
- The names of the slots that are available in the idaKMeans object

Value

The idaKMeans function returns an object of class idaKMeans and kmeans.

See Also

idaRetrieveModel, idaDropModel, idaListModels

Examples

```
## Not run:

#Create ida data frame
idf <- ida.data.frame("IRIS")

#Create a kmeans model stored in the database as KMEANSMODEL
km <- idaKMeans(idf, id="ID",modelname="KMEANSMODEL")

#Print the model
print(km)

#Predict the model
pred <- predict(km,idf,id="ID")

#Inspect the predictions
head(pred)


## End(Not run)
```

---

idaListAccelerators, idaSetAccelerator, idaGetAccelerator, idaGetAcceleratorDetails

Show and set accelerator settings

Description

Use these functions for DB2 for z/OS connections to retrieve the list of available accelerators and to set and get the current accelerator settings.

Usage

```
idaListAccelerators()
idaSetAccelerator(acceleratorName, queryAcceleration="ENABLE")
idaGetAccelerator()
idaGetAcceleratorDetails()
```

Arguments

`acceleratorName`
: The name of the accelerator where the analytics functions (like idaKMeans or idaTree) are executed.

`queryAcceleration`
: The value which the DB2 for z/OS register CURRENT QUERY ACCELERATION is set to. Possible values are "NONE", ENABLE", "ENABLE WITH FALLBACK", "ELIGIBLE" and "ALL"

Value

`idaListAccelerators` returns a data frame that contains a list of the accelerators available for the current DB2 for z/OS connection.

`idaSetAccelerator` sets the accelerator to be used for the subsequent calls of the analytics functions.

`idaGetAccelerator` retrieves the name of the accelerator.

`idaGetAcceleratorDetails` retrieves the name of accelerator together with the encoding (like UNICODE or EBCDIC) of its data and the value for query acceleration (like ENABLE or ELIGIBLE) and returns these values in a list object with elements "Accelerator", "Encoding" and "QueryAcceleration".

Examples

```
## Not run:
#Get a list of all accelerators
q <- idaListAccelerators();

#Set accelerator to "MYACCEL"
idaSetAccelerator("MYACCEL");

#Get name of current accelerator"
idaGetAccelerator();

#Get name of current accelerator together with its encoding and query acceleration"
idaGetAcceleratorDetails();

#Get encoding of current acccelerator
idaGetAcceleratorDetails()$Encoding

## End(Not run)
```

---

idaListModels and idaModelExists

List all predictive models in the database

Description

Use these function to list all models in the schema of the current user that were created using the functions like `idaNaiveBayes` or `idaKMeans` or check whether a model with a specific name exists.

Usage

```
idaListModels()
idaModelExists(modelname)
```

Arguments

`modelname`	The name of a predictive model.

Value

`idaListModels` returns a data frame that contains a list of the predictive models that are stored in the current schema and information about each one.

`idaModelExists` returns a boolean value depending on whether the model exists or not.

Examples

```
## Not run:
#Get a list of all models
q <- idaListModels();

## End(Not run)
```

`idaLm` *Linear regression*

Description

This function performs linear regression on the contents of an `ida.data.frame`.

Usage

```
idaLm(form, idadf, id = "id", modelname = NULL, dropModel = TRUE, limit = 25)

## S3 method for class 'idaLm'
print(x, ...)
## S3 method for class 'idaLm'
predict(object, newdata, id, outtable = NULL, ...)
## S3 method for class 'idaLm'
plot(x, names = TRUE, max_forw = 50, max_plot = 15, order = NULL,
lmgON = FALSE, backwardON = FALSE, ...)
```

Arguments

`form`	A `formula` object that specifies both the name of the column that contains the continuous target variable and either a list of columns separated by plus symbols or a single period (to specify that all other columns in the ida.data.frame are to be used as predictors). The specified columns can contain continuous or categorical values. The specified formula cannot contain transformations.
`idadf`	An ida.data.frame that contains the input data for the function.
`id`	The name of the column that contains a unique ID for each row of the input data. An id column needs to be specified, if a model contains categorical values, more than 41 columns or when dropModel is set to FALSE. If no valid id column was specified, a temporary id column will be used (not for DB2 for z/OS).
`modelname`	Name of the model that will be created in the database.

dropModel	logical: If TRUE the in database model will be dropped after the calculation.
limit	The maximum number of levels for a categorical column. Its default value is 25. This parameter only exists for consistency with older version of idaLm.
x	An object of the class idaLm.
object	An object of the class idaLm
newdata	An ida.data.frame that contains data that will be predicted.
outtable	The name of the table where the results will be written in.
names	logical: If set to TRUE then the plot will contain the names of the attributes instead of numbers.
max_forw	integer: The maximum number of iterations the heuristic forward/backward will be calculated.
max_plot	integer: The maximum number of attributes that will appear in the plot. It must be bigger than 0.
order	Vector of attribute names. The method will calculate the value of the models with the attributes in the order of the vector and plot the value for each of it.
lmgON	logical: If set TRUE the method will calculate the importance metric lmg. This method has exponential runningtime and is not supported for more than 15 attributes
backwardON	logical: If set TRUE the method will calculate the backward heuristic. By default (FALSE) it will do the forward heuristic.
...	Additional parameters.

Details

The idaLm function computes a linear regression model by extracting a covariance matrix and computing its inverse. This implementation is optimized for problems that involve a large number of samples and a relatively small number of predictors. The maximum number of columns is 78.

Missing values in the input table are ignored when calculating the covariance matrix. If this leads to undefined entries in the covariance matrix, the function fails. If the inverse of the covariance matrix cannot be computed (for example, due to correlated predictors), the Moore-Penrose generalized inverse is used instead.

The output of the idaLm function has the following attributes:

$coefficients is a vector with two values. The first value is the slope of the line that best fits the input data; the second value is its y-intercept.

$RSS is the root sum square (that is, the square root of the sum of the squares).

$effects is not used and can be ignored.

$rank is the rank.

$df.residuals is the number of degrees of freedom associated with the residuals.

$coefftab is a is a vector with four values:

- The slope and y-intercept of the line that best fits the input data
- The standard error

- The t-value
- The p-value

$Loglike is the log likelihood ratio.

$AIC is the Akaike information criterion. This is a measure of the relative quality of the model.

$BIC is the Bayesian information criterion. This is used for model selection.

$CovMat the Matrix used in the calculation ("Covariance Matrix"). This matrix is necessary for the Calculation in plot.idaLm and the statistics.

$card the number of dummy variables created for categorical columns and 1 for numericals.

$model the in database modelname of the idaLm object.

$numrow the number of rows of the input table that do not contain NAs.

$sigma the residual standard error.

The `plot.idaLm` function uses R^2 as a measure of quality of a linear model. R^2 compares the variance of the predicted values and the variance of the actual values of the target variable.

$First: Returns the R^2 value of the linear model for each attribute alone.

$Usefulness: Returns the R^2 value reduction of the linear model with all attributes to the linear model with one attribute taken away.

$Forward_Values: Is only calculated if backwardON=FALSE. This is a heuristic that adds in each step the attribute which has the most R^2 increase.

$LMG: Is only calculated if lmgON=TRUE. It returns the increase of R^2 of each attribute averaged over every possible permutation. By grouping some of the permutations we only need to average over every possible subset. For n attributes there are 2^n subsets. So LMG is an algorithm with exponential runningtime and is not recommended for more than 15 attributes.

$Backward_Values: Is only calculated if backwardON=TRUE. Similar to the forward heuristic. This time we choose in each step of the algorithm that has minimal R^2 reduction when taking it out of the model, starting with all attributes.

$Model_Values: Is only calculated if order is a vector of attributes. In this case the function calculates the R^2 value for the models that we get when we add one attribute of order in each step.

RelImpPlot.png: If lmgON=FALSE. This plot shows a stackplot of the values Usefulness,First and the Model_Value of the heuristic. Note that usually Usefulness<First<Model_Value and that the bars overlap each other. If lmgON=TRUE. This plot shows the LMG values of the attributes in the order of the heuristic forward, backward or order.

Value

The procedure returns a linear regression model of class `idaLm`.

Examples

```
## Not run:
#Create a pointer to table IRIS
idf <- ida.data.frame("IRIS")

#Calculate linear model in-db
lm1 <- idaLm(SepalLength~., idf)
```

```
library(ggplot2)
plot(lm1)

#Calculating linear models with categorical values requires an id column
lm1 <- idaLm(SepalLength~., idf, id="ID")

## End(Not run)
```

---

idaMerge *Merge IDA data frames*

Description

This function merges two IDA data frames(that is, two objects of the class ida.data.frame).

Usage

```
idaMerge(x, y, by=intersect(x@cols, y@cols), by.x=by, by.y=by,
    all=FALSE, all.x=all, all.y=all, sort=TRUE,
    suffixes=c("_x", "_y"), table=NULL)
```

Arguments

x	The first ida.data.frame object to be merged.
y	The second ida.data.frame object to be merged.
by	Specification of the common columns; see the *Details* section.
by.x	Specification of the common columns; see the *Details* section.
by.y	Specification of the common columns; see the *Details* section.
all	Whether non-matching columns of x and y are to be appended to the result. If set to FALSE, only columns common to both x and y are included in the output. This parameter overrides the all.x and all.y parameters. In SQL database terminology, specifying all=FALSE results in an inner join that is equivalent to a natural join, and specifying all=TRUE results in a full outer join. In a full outer join, the columns that are common to both x and y are followed by the remaining columns in x, which are followed by the remaining columns in y.
all.x	If columns from only one of the IDA data frames being merged are to be included in the output, set its corresponding parameter to TRUE and the other parameter to FALSE. In SQL database terminology, specifying all.x=TRUE and all.y=FALSE results in a left outer join, and specifying all.x=FALSE and all.y=TRUE results in a right outer join. If TRUE, then extra rows are added to the output, one for each row in x that has no matching row in y. These rows have a value of NA in those columns that are typically filled with values from y. The default is FALSE, so that only rows with data from both x and y are included in the output. If all.x is true, all the non matching cases of x are also appended to the result, with a value of NA filled in the corresponding columns of y

all.y	Analogous to all.x
sort	This parameter is ignored. The output is never sorted regardless of the setting of this parameter.
suffixes	Two 2-character strings, each of which specifies a suffix that is used when generating column names. By specifying different suffixes, you can ensure that each column can be uniquely attributed to either x or y. Note that a dot (.) is not a valid character for a column name.
table	Name of the output IDA data frame.

Details

This function merges two IDA data frames on the columns that they have in common. The rows in the two data frames that match on the specified columns are extracted, and joined together. If there is more than one match, all possible matches contribute one row each. For the precise meaning of 'match'.

If by or both by.x and by.y are of length 0 (a length zero vector or NULL), the result, r, is the Cartesian product of x and y, that is, a *cross join*.

If non-merged columns of the data frames have identical names and are to be included in the output, suffixes are appended to the names of the corresponding columns in the output to make their names unique.

Note that this function creates, in the current database, a view that corresponds to the output object. Within the current session, this view can be accessed using the same IDA data frame object. However, it is persistent and, after it is no longer needed, it must be dropped manually.

Value

A ida.data.frame object.

See Also

ida.data.frame

Examples

```
## Not run:

idf <- ida.data.frame('IRIS')

#Perform a self-join
idf2 <- idaMerge(idf,idf,by="ID")


## End(Not run)
```

idaNaiveBayes *Naive Bayes Classifier*

Description

This function generates a Naive Bayes classification model based on the contents of an IDA data frame (ida.data.frame).

Usage

```
idaNaiveBayes(form,data,id="id",modelname=NULL)

## S3 method for class 'idaNaiveBayes'
predict(object,newdata,id, withProbabilities=FALSE,...)
## S3 method for class 'idaNaiveBayes'
print(x,...)
```

Arguments

form	A formula object that describes the model to fit.
data	An ida.data.frame object.
id	The name of the column that contains unique IDs.
modelname	Name for the model. Will be created automatically unless specified otherwise.
object	An object of the class idaNaiveBayes to used for prediction, i.e. for applying it to new data.
newdata	An IDA data frame that contains the data to which to apply the model.
withProbabilities	A boolean value indicating if the probabilities for each class value are included in the result of the predict function.
x	An object of the class idaNaiveBayes to be printed.
...	Additional parameters to pass to the print and predict method.

Details

idaNaiveBayes builds a Naive Bayes classification model, thus a model that assumes independence of input variables with respect to the target variable.

Continuous input variables are discretized using equal width discretization. Missing values are ignored on a record and attribute level when calculating the conditional probabilities.

Models are stored persistently in the database under the name modelname. Model names cannot have more than 64 characters and cannot contain white spaces. They need to be quoted like table names, otherwise they will be treated upper case by default. Only one model with a given name is allowed in the database at a time. If a model with modelname already exists, you need to drop it with idaDropModel first before you can create another one with the same name. The model name can be used to retrieve the model later (idaRetrieveModel).

Value

The function idaNaiveBayes returns an object of class "idaNaiveBayes" and "naiveBayes" compatible with Naive Bayes objects produced by the e1071 package.

The predict.idaNaiveBayes method applies the model to the data in a table and returns an IDA data frame that contains a list of tuples, each of which comprises one row ID and one prediction.

Examples

```
## Not run:
#Create ida data frame
idf <- ida.data.frame("IRIS")

#Create a naive bayes model
nb <- idaNaiveBayes(Species~SepalLength,idf,"ID")

#Print the model
print(nb)

#Apply the model to data
idf2 <- predict(nb,idf,"ID")

#Inspect the results
head(idf2)


## End(Not run)
```

---

idaQuery, idaScalarQuery

Run an SQL query on the database

Description

Use these functions to run any SQL query on the database and put the results into a data.frame.

Usage

```
idaQuery(..., as.is=TRUE, na.strings = "NA")

idaScalarQuery(..., as.is=TRUE)
```

Arguments

...	Any number of query parts which are passed to paste.
as.is	Specifies whether the result columns are to be converted using RODBC type conversions (as.is=FALSE) or left unconverted (as.is=TRUE). For more information about RODBC type conversions, see the descriptions of the functions sqlGetResults and type.convert.
na.strings	character vector of strings to be mapped to NA when reading character data.

Details

All parts of the input query are concatenated with `paste(..., sep="")` and the result is passed to the database.

Value

The `idaQuery` function returns a data frame that contains the result of the specified query.

The `idaScalarQuery` function returns the result of the specified query coerced to a single scalar value.

Examples

```
## Not run:
#idaScalarQuery returns a single value
v <- idaScalarQuery("SELECT COUNT(*) FROM IRIS")

#idaQuery returns a data.frame
df <- idaQuery("SELECT * FROM IRIS")

#idaQuery and idaScalarQuery automatically paste all arguments into a single query
#This is convenient if you use variables

tableName <- "IRIS"
df <- idaScalarQuery("SELECT COUNT(*) FROM ",tableName)

## End(Not run)
```

`idaRetrieveModel` *Retrieve a predictive model from the database*

Description

Use this function to retrieve from the database a model that was created using a function like `idaNaiveBayes` or `idaKMeans`.

Usage

```
idaRetrieveModel(modelname)
```

Arguments

`modelname`	The name of the predictive model to be retrieved.

Value

This function returns an R object that contains a representation of the retrieved model. The class of the returned object depends on the function that was used to create the model.

Examples

```
## Not run:

#Retrieve the model with name "MYKMEANSMODEL" from the database
trCopy <- idaRetrieveModel("KMEANSMODEL");

## End(Not run)
```

---

idaSample *Taking a random sample from a IDA data frame*

Description

This function draws a random sample from a IDA data frame (that is, an object of the class ida.data.frame).

Usage

```
idaSample(bdf, n, stratCol=NULL,stratVals=NULL,stratProbs=NULL,
dbPreSamplePercentage=100,fetchFirst=F);
```

Arguments

bdf	The IDA data frame from which the sample is to be drawn.
n	The number of rows of sample data to be retrieved.
stratCol	For stratified sampling, the column that determines the strata.
stratVals	For stratified sampling, a vector of values that determine the subset of strata from which samples are to be drawn.
stratProbs	For stratified sampling, a vector of explicit sampling probabilities. Each value corresponds to a value of the vector specified for stratVals.
dbPreSamplePercentage	The percentage of the IDA data frame from which the sample is to be drawn (see details).
fetchFirst	Fetch first rows instead of using random sample.

Details

If stratCol is specified, a stratified sample based on the contents of the specified column is taken. Unless stratVals is also specified, each unique value in the column results in one stratum. If stratVals is also specified, only the values it specifies result in strata, and only rows that contain one of those values are included in the sample; other rows are ignored.

Unless stratProbs is also specified, the number of rows retrieved for each stratum is proportional to the size of that stratum relative to the overall sample.

To undersample or oversample data, use stratProbs to specify, for each value of stratVals, the fraction of the rows of the corresponding stratum that are to be included in the sample.

For each stratum, the calculated number of rows is rounded up to the next highest integer. This ensures that there is at least one sample for each stratum. Consequently, the number of samples that is returned might be higher than the value specified for `n`.

The value of `dbPreSamplePercentage` is a numeric value in the range 0-100 that represents the percentage of the IDA data frame that is to serve as the source of the sample data. When working with an especially large IDA data frame, specifying a value smaller than 100 improves performance, because less data must be processed. However, the proportionality of the pre-sampled data might vary from that of the complete data, and this would result in a biased sample. It can even happen that entire strata are excluded from the final sample.

When `fetchFirst` is set to TRUE, the sample values of each stratum are taken in the order in which they are returned from the database rather than randomly. This is usually much faster than random sampling, but can introduce bias.

Value

An object of class `data.frame` that contains the sample.

Examples

```
## Not run:
idf<-ida.data.frame('IRIS')

#Simple random sampling
df <- idaSample(idf,10)

#Stratified sample
df <- idaSample(idf,10,'Species')


## End(Not run)
```

---

idaShowTables *Return a list of tables*

Description

Returns a data frame that contains the names of the tables contained in the current database.

Usage

```
idaShowTables(showAll=FALSE, matchStr=NULL, schema=NULL, accelerated=FALSE)
```

Arguments

`showAll`	List all tables that are listed in the catalog of the current database (TRUE) or only those tables that are in the current schema (FALSE).
`matchStr`	If not NULL, only tables that contain the character string in this argument will be returned.

schema	If not NULL, only tables with this schema will be returned. This parameter is ignored if showAll=FALSE.
accelerated	Valid for DB2 for z/OS connections only. If TRUE, only accelerated tables will be returned.

Value

A data frame with the columns Schema, Name, Owner, and Type. For DB2 for z/OS connections the columns Acceleratorname and Enable are included as well.

Examples

```
## Not run:

#Get a list of all tables in the current schema
tabs <- idaShowTables()


## End(Not run)
```

---

idaTable *In-Database Cross Tabulation and Table Creation*

Description

Function used to build a contingency table of the counts at each combination of factor levels based on the contents of a IDA data frame (ida.data.frame).

Usage

```
idaTable(idadf,max.entries=1000)
```

Arguments

idadf	A IDA data frame that contains the input data for the function.
max.entries	The maximum number of entries. If the cross product of all columns exceeds this number, an error will be thrown.

Details

idaTable uses the cross-classifying factors to build a contingency table of the counts at each combination of categorical values in all categorical columns of the ida.data.frame passed as input.

Value

The idaTable function returns a contingency table, an object of class "table".

Examples

```
## Not run:

#Create a pointer to the table IRIS
idf<-ida.data.frame('IRIS')

#Add a column
idf$SepalLengthg4 <- ifelse(idf$SepalLength>4,'t','f')

#Calculate the cross-table between Species and SepalLengthg4
idaTable(idf[,c('Species','SepalLengthg4')])


## End(Not run)
```

`idaTApply` *Apply R-function to subsets of IDA data frame*

Description

This function applies a R function to each subset (group of rows) of a given IDA data frame (`ida.data.frame`).

Usage

```
idaTApply(X, INDEX, FUN = NULL, output.name=NULL, output.signature=NULL,
          clear.existing=FALSE, debugger.mode=FALSE,
          num.tasks = 0, working.dir=NULL, apply.function="default", ...)
```

Arguments

`X`	A IDA data frame that contains the input data for the function.
`INDEX`	The name or the position of the column of the input IDA data frame X used to partition the input data into subsets.
`FUN`	The R function to be applied to the subsets of the input data.
`output.name`	The name of the output table where the results are written to.
`output.signature`	The Db2 data types of the output table. It is a named list with the column names as the names and the data types as the values. Supported data types are CHAR, VARCHAR, SMALLINT, INTEGER, BIGINT, FLOAT, REAL, DOUBLE, DECFLOAT, DECIMAL, NUMERIC, DATE
`clear.existing`	If TRUE the ouput table is dropped before recreating it.
`debugger.mode`	If TRUE intermediate results written into the working directory will not be removed.

num.tasks	The number of parallel tasks, i.e. R processes, which execute the R function on the subsets of the input data. If not specified or if the value is less than 1 it is calculated based on the number of available CPUs.
working.dir	The name of the directory where the directory is created into which intermediate results are written to. This directory is removed if debugger.mode is FALSE. The default value for working.directory is the value of the extbl_location Db2 database configuration variable or, if this variable has not been set, the home directory.
apply.function	The name of the R function to be used for parallelizing the execution of the calls of the function FUN. Possible values are "default", "spark.lapply" and "mclapply". If the value is "default" "spark-lapply" is used in a multi-node and "mclapply" in a single node environment. Please note that using the "spark.lapply" function requires Db2 Warehouse with integrated Spark.
...	Additional parameters that can be passed to the function FUN to be called by idaTApply.

Details

idaTApply applies a user-provided R function to each subset (group of rows) of a given ida.data.frame. The subsets are determined by a specified index column. The results of applying the function are written into a Db2 table which is referenced by the returned ida.data.frame.

Value

The idaTApply function returns a ida.data.frame .

Examples

```
## Not run:
#Create an ida data frame from the iris data
idf <- as.ida.data.frame(iris)

#Define a function that computes the mean value for every column of a data frame x
#except the index column.
#It returns a data frame with the value of the index column and the mean values.
columnMeans<- function(x, index) {
       cbind(index=x[1,match(index, names(x))],
              as.data.frame(as.list(apply(x[,names(x) != index],2,mean))))}


#Apply the columnMeans function to the subsets of the iris data identified by the Species column
resSig <- list(Species="VARCHAR(12)", MSepalLength="DOUBLE", MSepalWidth="DOUBLE",
                                      MPetalLength="DOUBLE", MPetalWidth="DOUBLE")
resDf <-
  idaTApply(idf, "Species", FUN=columnMeans, output.name="IRIS_MEANS", output.signature=resSig)

#It is possible as well to apply an anonymous function.
#The value "5" of the second parameter designates the position of the "Species" column
#in the idf ida.data.frame.
#The output table of the previous call is recreated because of the "clear.existing=T" parameter.
```

```
resDf <- idaTApply(idf, 5,
                   FUN=function(x, index) {
                             cbind(index=x[1,match(index, names(x))],
                             as.data.frame(as.list(apply(x[,names(x) != index],2,mean))))},
                   output.name="IRIS_MEANS", output.signature=resSig, clear.existing=T)

#Apply the columnMeans2 function which has an additional parameter "columns"
#to specify the columns for which the mean values are computed
columnMeans2 <- function(x, index, columns) {
       cbind(index=x[1,match(index, names(x))],
           as.data.frame(as.list(apply(x[,names(x) != index & names(x) %in% columns],2,mean))))}
petalColumns <- c("PetalLength", "PetalWidth")
resSig2 <- list(Species="VARCHAR(12)", MPetalLength="DOUBLE", MPetalWidth="DOUBLE")
resDf2 <- idaTApply(idf, "Species", FUN=columnMeans2, output.name="IRIS_MEANS2",
                    output.signature=resSig2, clear.existing=T, columns=petalColumns)

## End(Not run)
```

---

idaTree *Decision and Regression tree*

Description

This function generates a tree model based on the contents of an IDA data frame (ida.data.frame).

Usage

```
idaTree( form, data, id, minsplit=50, maxdepth=10, qmeasure=NULL,
         minimprove=0.01, eval=NULL, valtable=NULL, modelname=NULL)


## S3 method for class 'idaTree'
plot(x,...)
## S3 method for class 'idaTree'
predict(object, newdata, id, ...)
```

Arguments

form	A formula object that specifies both the name of the column that contains the categorical target variable and either a list of columns separated by plus symbols (each column corresponds to one predictor variable) or a single period (to specify that all other columns in the IDA data frame are to be used as predictors.
data	An IDA data frame that contains the input data for the function. The input IDA data frame must include a column that contains a unique ID for each row.
id	The name of the column that contains a unique ID for each row of the input data.
minsplit	The minimum number of rows a node must contain to be split further.

maxdepth	The maximum depth (that is, the number of hierarchical levels) of the generated tree.
qmeasure	The measure that is to be used to prune the tree. For a decision tree, allowed values are "Acc" (this is the default) and "wAcc". For a regression tree, allowed values are "mse" (this is the default), "r2", "pearson", and "spearman".
minimprove	The minimum improvement. A node is not split further unless the split improves the class impurity by at least the amount specified for this parameter.
eval	The criterion that is to be used to calculate each split. For a decision tree, allowed values are "entropy" (this is the default) and "gini". For a regression tree, the only allowed value is "variance" (this is the default).
valtable	When the output tree is to be pruned using external data, use this parameter to specify the fully-qualified name of the table that contains that data. Otherwise, specify NULL.
modelname	The name under which the model is stored in the database. This is the name that is specified when using functions such as idaRetrieveModel or idaDropModel.
object	An object of the class idaTree.
x	An object of the class idaTree.
newdata	A IDA data frame that contains the data to which to apply the model.
...	additional arguments to be passed to plot or predict.

Details

The idaTree function uses a top-down, iterative procedure to generate a decision-tree or regression-tree model, depending on the type of the target variable. The resulting model comprises a network of nodes and connectors, and each subnode is the endpoint of a binary split.

A node is not split further when any of the following are true:

- The node has a uniform class (and therefore cannot be split further).
- Additional splits do not improve the class impurity by at least the amount specified by minimprove.
- The number of rows contained by the node is less than the value specified by minsplit.
- The tree depth reaches the value specified by maxdepth.

If variable that is used to determine a split does not have a value, the corresponding row remains in the node that is being split.

The output of the print function for a idaTree object is a textual description of the corresponding model.

The output of the plot function for a idaTree object is a graphical representation of the corresponding model.

Models are stored persistently in the database under the name modelname. Model names cannot have more than 64 characters and cannot contain white spaces. They need to be quoted like table names, otherwise they will be treated upper case by default. Only one model with a given name is allowed in the database at a time. If a model with modelname already exists, you need to drop it with idaDropModel first before you can create another one with the same name. The model name can be used to retrieve the model later (idaRetrieveModel).

The predict.idaTree method applies the model to the data in a table and returns a IDA data frame that contains a list of tuples, each of which comprises one row ID and one prediction.

Value

The idaTree function returns an object of classes `idaTree` and `rpart`.

See Also

idaRetrieveModel, idaDropModel, idaListModels

Examples

```
## Not run:

#Create a pointer to the table IRIS
idf <- ida.data.frame('IRIS')

#Create a tree model
tr <- idaTree(Species~.,idf,"ID",modelname="MYTREEMODEL")

#Print the model
print(tr)

#Plot the model
plot(tr)

#Apply the model to data
pred <- predict(tr,idf,id="ID")

#Inspect the predictions
head(pred)


## End(Not run)
```

`idaTwoStep` *two step clustering*

Description

This function generates a two step clustering model based on the contents of an IDA data frame (ida.data.frame).

Usage

```
idaTwoStep( data, id, k = 3, maxleaves = 1000, distance = "euclidean", outtable = NULL,
            randseed = 12345, statistics = NULL, maxk = 20, nodecapacity = 6,
            leafcapacity = 8, outlierfraction = 0, modelname = NULL)

## S3 method for class 'idaTwoStep'
print(x,...)
```

```
## S3 method for class 'idaTwoStep'
predict(object, newdata, id,...)
```

Arguments

data	A IDA data frame that contains the input data for the function. The input IDA data frame must include a column that contains a unique ID for each row.
id	The name of the column that contains a unique ID for each row of the input data.
k	The number of clusters to be calculated.
maxleaves	The maximum number of leaf nodes in the initial clustering tree. When the tree contains maxleaves leaf nodes, the following data records are aggregated into clusters associated with the existing leaf nodes. This parameter is available for Db2 for z/OS only and ignored for Db2 Warehouse with integrated Spark.
maxk	The maximum number of clusters that can be determined automatically.
nodecapacity	The branching factor of the internal tree that is used in pass 1. Each node can have up to <nodecapacity> subnodes. This parameter is available for Db2 Warehouse with integrated Spark only and ignored for Db2 for z/OS.
leafcapacity	The number of clusters per leaf node in the internal tree that is used in pass 1. This parameter is available for Db2 Warehouse with integrated Spark only and ignored for Db2 for z/OS.
outlierfraction	The fraction of the records that is to be considered as outlier in the internal tree that is used in pass 1. Clusters that contain less than <outlierfraction> times the mean number of data records per cluster are removed. This parameter is available for Db2 Warehouse with integrated Spark only and ignored for Db2 for z/OS.
distance	The distance function that is to be used. This can be set to "euclidean", which causes the squared Euclidean distance to be used, or "norm_euclidean", which causes normalized euclidean distance to be used.
outtable	The name of the output table that is to contain the results of the operation. When NULL is specified, a table name is generated automatically.
randseed	The seed for the random number generator.
statistics	Denotes which statistics to calculate. Allowed values are "none","columns" and "all". If NULL, the default of the database system will be used.
modelname	The name under which the model is stored in the database. This is the name that is specified when using functions such as idaRetrieveModel or idaDropModel.
object	An object of the class idaTwoStep to be used for prediction, i.e. for applying it to new data.
x	An object of the class idaTwoStep to be printed.
newdata	A IDA data frame that contains the data to which to apply the model.
...	Additional parameters to pass to the print or predict method.

Details

The idaTwoStep clustering function distributes first the input data into a hierarchical tree structure according to the distance between the data records where each leaf node corresponds to a (small) cluster. Then idaTwoStep reduces the tree by aggregating the leaf nodes according to the distance function until k clusters remain.

Models are stored persistently in database under the name `modelname`. Model names cannot have more than 64 characters and cannot contain white spaces. They need to be quoted like table names, otherwise they will be treated upper case by default. Only one model with a given name is allowed in the database at a time. If a model with `modelname` already exists, you need to drop it with `idaDropModel` first before you can create another one with the same name. The model name can be used to retrieve the model later (`idaRetrieveModel`).

The output of the print function for a idaTwoStep object is:

- A vector containing a list of centers
- A vector containing a list of cluster sizes
- A vector containing a list of the number of elements in each cluster
- A data frame or the name of the table containing the calculated cluster assignments
- The within-cluster sum of squares (which indicates cluster density)
- The names of the slots that are available in the idaTwoStep object

Value

The idaTwoStep function returns an object of class `idaTwoStep` and `TwoStep`.

See Also

`idaRetrieveModel`, `idaDropModel`, `idaListModels`

Examples

```
## Not run:

#Create ida data frame
idf <- ida.data.frame("IRIS")

#Create a TwoStep model stored in the database as TwoStepMODEL
tsm <- idaTwoStep(idf, id="ID",modelname="TwoStepMODEL")

#Print the model
print(tsm)

#Predict the model
pred <- predict(tsm,idf,id="ID")

#Inspect the predictions
head(pred)


## End(Not run)
```

Index