Package 'RODM'

July 2, 2014

Version 1.1

Date 2011-11-20

Title R interface to Oracle Data Mining

Author Pablo Tamayo <pablo.tamayo@oracle.com> and Ari Mozes <ari.mozes@oracle.com>

Maintainer Pablo Tamayo <pablo.tamayo@oracle.com>

Depends R (>= 2.10.1), RODBC

Suggests RODBC, mlbench, verification, PASWR, scatterplot3d

Description This package implements an interface to Oracle Data Mining (ODM). It provides an ideal environment for rapid development of demos and proof of concept data mining studies. It facilitates the prototyping of vertical applications and makes ODM and the RDBMS environment easily accessible to statisticians and data analysts familiar with R but not fluent in SQL or familiar with the database environment. It also facilitates the benchmarking and testing of ODM functionality including the production of summary statistics, performance metrics and graphics. It enables the scripting and control of production data mining methodologies from a high-level environment. Oracle Data Mining (ODM) is an option of Oracle Relational Database Management System (RDBMS) Enterprise Edition (EE). It contains several data mining and data analysis algorithms for classification, prediction, regression,clustering, associations, feature selection, anomaly detection,feature extraction, and specialized analytics. It provides means for the creation, management and operational deployment of data mining models inside the database environment. For more information consult the entry for ``Oracle Data Mining'' in Wikipedia (en.wikipedia.org).

License LGPL (>= 2)

URL http://www.r-project.org

Repository CRAN

Date/Publication 2012-10-29 08:57:32

NeedsCompilation no

R topics documented:

`RODM-package` *RODM: An Interface to Oracle Data Mining*

Description

Oracle Data Mining (ODM) is an option of Oracle's Relational Database Management System (RDBMS) Enterprise Edition (EE). It contains several data mining and data analysis algorithms for classification, prediction, regression, clustering, associations, feature selection, anomaly detection, feature extraction, and specialized analytics. It provides means for the creation, management and operational deployment of data mining models inside the database environment.

RODM is an interface that provides a powerful environment for prototyping data analysis and data mining methodologies. It facilitates the prototyping of vertical applications and makes ODM and the RDBMS environment easily accessible to statisticians and data analysts familiar with R but not experts in SQL. In this way it provides an ideal environment for demos and proof of concept studies. It also facilitates the benchmarking and testing of functionality including statistics and graphics of performance metrics and enables the scripting and control of production data mining methodologies from a high-level environment.

Details

Package:	RODM
Type:	Package
Version:	1.0-2
Date:	2010-05-01
License:	LGPL

RODM is a package that provides access to Oracle's in-database data mining functionality.

Requirements
RODM requires the use of an Oracle release 11g database. If you don't have an installed Oracle database in place and need to install one from scratch we stronly recommend you follow the guidelines in the Oracle Data Mining Administrator's Guide. RODM requires R release > 2.10.1.

Connecting to an Oracle database:
RODM_open_dbms_connection
RODM_close_dbms_connection

The above routines are used to establish a connection to an Oracle 11g database using ODBC. RODM uses the RODBC package as a means to manage the database connection. A data source name must be provided, as well as a username and password. The user that is connecting needs sufficient privileges for performing mining operations in the database. We have tested RODM using the Oracle ODBC driver that comes with the Oracle RDBMS. We recommend the use of this ODBC driver instead of others.

Pushing data to the database:
RODM_create_dbms_table
RODM_drop_dbms_table

Once a valid database connection has been established, in-database mining can begin. If the data to be mined exists within R (e.g., in a data frame), it first needs to be pushed to the database and placed in a table. The above routines leverage the RODBC package to push data to a database table, which can then be accessed for mining by ODM.

Auxilliary functions (for internal use):
RODM_store_settings
RODM_create_model

The above routines are used under-the-covers when building ODM models. They do not need to be invoked directly. They are present merely to improve maintainability and modularity.

Building ODM models:
RODM_create_ai_model
RODM_create_assoc_model
RODM_create_dt_model
RODM_create_glm_model
RODM_create_kmeans_model
RODM_create_nb_model
RODM_create_oc_model
RODM_create_svm_model

The above nine routines are used to build ODM models in the database. They share many of the same arguments. All of these routines require a database connection (as retrieved via RODM_open_dbms_connection) and a table/view in the database (either pre-existing or created via RODM_create_dbms_table) which will provide the training data. All routines accept a case identifier column name. This is the name of a column which can be used to uniquely identify a training record. Most routines do not need a case identifier, but some may provide extra information if one is present (e.g., cluster assignments). All supervised algorithms require a target column name. A model name can be specified (or defaults to an algorithm-specific model name). When created, the model will exist in Oracle as a database schema object. Most algorithms accept a parameter to direct ODM to enable automatic data preparation (default TRUE). This feature will request that ODM prepare data as befitting individual algorithm needs (e.g., outlier treatment, binning, normalization, missing value imputation). Many algorithms accept a number of expert settings. These expert settings will differ from algorithm to algorithm, and ODM is designed to identify values for these settings without user input, hence they do not need to be specified by the user in most situations. When the models are created in the database, information regarding the models can be retrieved and returned to the R environment. The retrieve_outputs_to_R parameter tells RODM whether or not this information should be pulled back into R for further analysis in R. As these models are database schema objects, they can be left in the database for future use. They can be applied to new data as desired. The default behavior is to leave the models in the database, but they can be automatically cleaned up by changing the leave_model_in_dbms parameter. If a model with the same name already exists in the database schema when another is being created, the previous model will be automatically dropped. Finally, the RODM package is envisioned as a quick proof of concept mechanism, with the potential of deploying the resulting methodology wholly within Oracle. As such, it is necessary to capture the SQL that would be used in the database. The sql.log.file parameter can be used to have RODM produce a file with the relevant SQL statements that comprise the work being performed.

Further operations involving ODM models:
RODM_list_dbms_models
RODM_apply_model
RODM_drop_model

Finally, there are a few routines involving ODM models once they are built. The list of accessible ODM models can be retrieved, and individual models can be dropped. Models (other than those used for Attribute Importance and Associations) can be applied to new data in the database. Regression models will produce the expected value given the input variables. Classification models will produce the probability distribution across target classes for each case, as well as a column indicating the winning class. Clustering models will produce the probability distribution across clusters for each case, as well as a column indicating the most likely cluster assignment. In all cases, additional columns from the test/apply dataset can be included in the output via the supplemental_cols parameter. It is necessary to provide some information here if there is a desire to link the results back to the original data. Either a case identifier should be provided, or the list of columns which will yield sufficient information for future analysis.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

Maintainer: Pablo Tamayo <pablo.tamayo@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Oracle Database SQL Language Reference (Data Mining functions) 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF20030

Richard O. Duda, Peter E. Hart, David G. Stork, Pattern Classification (2nd Edition). John Wiley & Sons 2001.

Wikipedia entry for Oracle Data Mining. http://en.wikipedia.org/wiki/Oracle_Data_Mining

P. Tamayo, C. Berger, M. M. Campos, J. S. Yarmus, B. L.Milenova, A. Mozes, M. Taft, M. Hornick, R. Krishnan, S.Thomas, M. Kelly, D. Mukhin, R. Haberstroh, S. Stephens and J. Myczkowski. Oracle Data Mining - Data Mining in the Database Environment. In Part VII of Data Mining and Knowledge Discovery Handbook, Maimon, O.; Rokach, L. (Eds.) 2005, p315-1329, ISBN-10: 0-387-24435-2.

Oracle Data Mining: Mining Gold from Your Warehouse, (Oracle In-Focus series), by Dr. Carolyn Hamm.

`RODM_apply_model` *Apply an Oracle Data Mining model*

Description

This function applies a previously created ODM model to score new data.

Usage

```
RODM_apply_model(database,
                 data_table_name,
                 model_name,
                 supplemental_cols,
                 sql.log.file = NULL)
```

Arguments

`database`	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
`data_table_name`	Database table/view containing the training dataset.
`model_name`	ODM Model name

`supplemental_cols`
: Columns to carry over into the output result.

`sql.log.file`
: File to append the log of all the SQL calls made by this function.

Details

This function applies a previously created ODM model to score new data. The supplemental_cols parameter should be assigned in such a way as to retain the connection between the scores and the original cases. The simplest way to do this is to include a unique case identifier in the list, which provides the ability to identify the original row information for a score. If only some of the information from the original data is needed (for example, only the actual target value is needed when computing a measure of accuracy), then it is only this information which should be identified by the supplemental columns.

Value

A list with the following components:

`model.apply_results`
: A data frame table containing: For classification: class 1 probability numeric/double ... ... class N probability numeric/double supplemental column 1 ... supplemental column M prediction

For regression: supplemental column 1 ... supplemental column M prediction numeric/double

For anomaly detection (e.g. one-class SVM): class 1 probability numeric/integer (class 1 is the typical class) class 0 probability numeric/integer (class 0 is the outlier class) supplemental column 1 ... supplemental column M prediction integer: 0 or 1

For clustering: leaf cluster 1 probability numeric/double ... ... leaf cluster N probability numeric/double supplemental column 1 ... supplemental column M cluster_id

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Oracle Database SQL Language Reference (Data Mining functions) 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF20030

See Also

RODM_create_svm_model, RODM_create_kmeans_model, RODM_create_oc_model, RODM_create_nb_model, RODM_create_glm_model, RODM_create_dt_model

Examples

```
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid= "rodm", pwd = "rodm")

### Classification

# Predicting survival in the sinking of the Titanic based on pasenger's sex, age, class, etc.


data(titanic3, package="PASWR")                                    # Load survival data from Titanic
ds <- titanic3[,c("pclass", "survived", "sex", "age", "fare", "embarked")]  # Select subset of attributes
ds[,"survived"] <- ifelse(ds[,"survived"] == 1, "Yes", "No")             # Rename target values
n.rows <- length(ds[,1])                                                  # Number of rows
random_sample <- sample(1:n.rows, ceiling(n.rows/2))   # Split dataset randomly in train/test subsets
titanic_train <- ds[random_sample,]                          # Training set
titanic_test <-  ds[setdiff(1:n.rows, random_sample),]      # Test set
RODM_create_dbms_table(DB, "titanic_train")   # Push the training table to the database
RODM_create_dbms_table(DB, "titanic_test")    # Push the testing table to the database
svm <- RODM_create_svm_model(database = DB,    # Create ODM SVM classification model
                             data_table_name = "titanic_train",
                             target_column_name = "survived",
                             model_name = "SVM_MODEL",
                             mining_function = "classification")

# Apply the SVM classification model to test data.
svm2 <- RODM_apply_model(database = DB,    # Predict test data
                         data_table_name = "titanic_test",
                         model_name = "SVM_MODEL",
                         supplemental_cols = "survived")

print(svm2$model.apply.results[1:10,])                          # Print example of prediction results
actual <- svm2$model.apply.results[, "SURVIVED"]
predicted <- svm2$model.apply.results[, "PREDICTION"]
probs <- as.real(as.character(svm2$model.apply.results[, "'Yes'"]))
table(actual, predicted, dnn = c("Actual", "Predicted"))              # Confusion matrix
library(verification)
perf.auc <- roc.area(ifelse(actual == "Yes", 1, 0), probs)          # Compute ROC and plot
auc.roc <- signif(perf.auc$A, digits=3)
auc.roc.p <- signif(perf.auc$p.value, digits=3)
roc.plot(ifelse(actual == "Yes", 1, 0), probs, binormal=T, plot="both", xlab="False Positive Rate",
         ylab="True Postive Rate", main= "Titanic survival ODM SVM model ROC Curve")
text(0.7, 0.4, labels= paste("AUC ROC:", signif(perf.auc$A, digits=3)))
```

```
text(0.7, 0.3, labels= paste("p-value:", signif(perf.auc$p.value, digits=3)))

RODM_drop_model(DB, "SVM_MODEL")            # Drop the model
RODM_drop_dbms_table(DB, "titanic_train")   # Drop the training table in the database
RODM_drop_dbms_table(DB, "titanic_test")    # Drop the testing table in the database

## End(Not run)

### Regression

# Aproximating a one-dimensional non-linear function

## Not run:
X1 <- 10 * runif(500) - 5
Y1 <- X1*cos(X1) + 2*runif(500)
ds <- data.frame(cbind(X1, Y1))
RODM_create_dbms_table(DB, "ds")   # Push the table to the database
svm <- RODM_create_svm_model(database = DB,    # Create ODM SVM regression model
                             data_table_name = "ds",
                             target_column_name = "Y1",
                             model_name = "SVM_MODEL",
                             mining_function = "regression")

# Apply the SVM regression model to test data.
svm2 <- RODM_apply_model(database = DB,    # Predict training data
                         data_table_name = "ds",
                         model_name = "SVM_MODEL",
                         supplemental_cols = "X1")

plot(X1, Y1, pch=20, col="blue")
points(x=svm2$model.apply.results[, "X1"], svm2$model.apply.results[, "PREDICTION"], pch=20, col="red")
legend(-4, -1.5, legend = c("actual", "SVM regression"), pch = c(20, 20), col = c("blue", "red"),
               pt.bg =  c("blue", "red"), cex = 1.20, pt.cex=1.5, bty="n")

RODM_drop_model(DB, "SVM_MODEL")            # Drop the model
RODM_drop_dbms_table(DB, "ds")              # Drop the database table

## End(Not run)

### Anomaly detection

# Finding outliers in a 2D-dimensional discrete distribution of points

## Not run:
X1 <- c(rnorm(200, mean = 2, sd = 1), rnorm(300, mean = 8, sd = 2))
Y1 <- c(rnorm(200, mean = 2, sd = 1.5), rnorm(300, mean = 8, sd = 1.5))
ds <- data.frame(cbind(X1, Y1))
RODM_create_dbms_table(DB, "ds")   # Push the table to the database
svm <- RODM_create_svm_model(database = DB,    # Create ODM SVM anomaly detection model
                             data_table_name = "ds",
                             target_column_name = NULL,
                             model_name = "SVM_MODEL",
                             mining_function = "anomaly_detection")
```

```
# Apply the SVM anomaly detection model to data.
svm2 <- RODM_apply_model(database = DB,    # Predict training data
                         data_table_name = "ds",
                         model_name = "SVM_MODEL",
                         supplemental_cols = c("X1","Y1"))

plot(X1, Y1, pch=20, col="white")
col <- ifelse(svm2$model.apply.results[, "PREDICTION"] == 1, "green", "red")
for (i in 1:500) points(x=svm2$model.apply.results[i, "X1"],
                        y=svm2$model.apply.results[i, "Y1"],
                        col = col[i], pch=20)
legend(8, 2, legend = c("typical", "anomaly"), pch = c(20, 20), col = c("green", "red"),
               pt.bg =  c("green", "red"), cex = 1.20, pt.cex=1.5, bty="n")

RODM_drop_model(DB, "SVM_MODEL")            # Drop the model
RODM_drop_dbms_table(DB, "ds")    # Drop the database table

## End(Not run)

### Clustering

# Clustering a 2D multi-Gaussian distribution of points into clusters

## Not run:
set.seed(seed=6218945)
X1 <- c(rnorm(100, mean = 2, sd = 1), rnorm(100, mean = 8, sd = 2), rnorm(100, mean = 5, sd = 0.6),
       rnorm(100, mean = 4, sd = 1), rnorm(100, mean = 10, sd = 1)) # Create and merge 5 Gaussian distributions
Y1 <- c(rnorm(100, mean = 1, sd = 2), rnorm(100, mean = 4, sd = 1.5), rnorm(100, mean = 6, sd = 0.5),
        rnorm(100, mean = 3, sd = 0.2), rnorm(100, mean = 2, sd = 1))
ds <- data.frame(cbind(X1, Y1))
n.rows <- length(ds[,1])                                              # Number of rows
row.id <- matrix(seq(1, n.rows), nrow=n.rows, ncol=1, dimnames= list(NULL, c("ROW_ID"))) # Row id
ds <- cbind(row.id, ds)                                            # Add row id to dataset
RODM_create_dbms_table(DB, "ds")
km <- RODM_create_kmeans_model(
   database = DB,                  # database ODBC channel identifier
   data_table_name = "ds",         # data frame containing the input dataset
   case_id_column_name = "ROW_ID", # case id to enable assignments during build
   num_clusters = 5)

# Apply the K-Means clustering model to data.
km2 <- RODM_apply_model(
   database = DB,                  # database ODBC channel identifier
   data_table_name = "ds",         # data frame containing the input dataset
   model_name = "KM_MODEL",
   supplemental_cols = c("X1","Y1"))

x1a <- km2$model.apply.results[, "X1"]
y1a <- km2$model.apply.results[, "Y1"]
clu <- km2$model.apply.results[, "CLUSTER_ID"]
c.numbers <- unique(as.numeric(clu))
c.assign <- match(clu, c.numbers)
```

```
color.map <- c("blue", "green", "red", "orange", "purple")
color <- color.map[c.assign]
nf <- layout(matrix(c(1, 2), 1, 2, byrow=T), widths = c(1, 1), heights = 1, respect = FALSE)
plot(x1a, y1a, pch=20, col=1, xlab="X1", ylab="Y1", main="Original Data Points")
plot(x1a, y1a, pch=20, type = "n", xlab="X1", ylab="Y1", main="After kmeans clustering")
for (i in 1:n.rows) {
   points(x1a[i], y1a[i], col= color[i], pch=20)
}
legend(5, -0.5, legend=c("Cluster 1", "Cluster 2", "Cluster 3", "Cluster 4", "Cluster 5"), pch = rep(20, 5),
       col = color.map, pt.bg = color.map, cex = 0.8, pt.cex=1, bty="n")

RODM_drop_model(DB, "KM_MODEL")         # Drop the model
RODM_drop_dbms_table(DB, "ds")          # Drop the database table


RODM_close_dbms_connection(DB)

## End(Not run)
```

`RODM_close_dbms_connection` *Close a connection to an Oracle Database*

Description

This function closes a connection to an Oracle Database.

Usage

```
RODM_close_dbms_connection(
    database)
```

Arguments

`database`	Database ODBC channel identifier returned from a call to `RODM_open_dbms_connection`

Details

This functions closes an Oracle Database ODBC channel by calling the RODBC function: odbcClose.

Value

None.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

See Also

RODM_open_dbms_connection

Examples

```
# Given a database user rodm, establish a connection to the orcl11g
# database.
# The database user would need privileges as described above, and could
# have been created in a fashion similar to:
# grant create session, create table, create view, create mining model,
#       unlimited tablespace to rodm identified by rodm;

## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid="rodm", pwd="rodm")

# Close the connection to the database.
RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_create_ai_model *Create an Attribute Importance (AI) model*

Description

This function creates an Oracle Data Mining Attribute Importance (AI) model.

Usage

```
RODM_create_ai_model(database,
                     data_table_name,
                     case_id_column_name = NULL,
                     target_column_name,
                     model_name = "AI_MODEL",
                     auto_data_prep = TRUE,
                     retrieve_outputs_to_R = TRUE,
                     leave_model_in_dbms = TRUE,
                     sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Database table/view containing the training dataset.
case_id_column_name	Row unique case identifier in data_table_name.
target_column_name	Target column name in data_table_name.
model_name	ODM Model name.
auto_data_prep	Setting that specifies whether or not ODM should perform automatic data preparation.
retrieve_outputs_to_R	Flag controlling if the output results are moved to the R environment.
leave_model_in_dbms	Flag controlling if the model is dropped or left in RDBMS.
sql.log.file	File where to append the log of all the SQL calls made by this function.

Details

Attribute Importance (AI) uses a Minimum Description Length (MDL) based algorithm that ranks the relative importance of attributes in their ability to contribute to the prediction of a specified target attribute. This algorithm can provide insight into the attributes relevance to a specified target attribute and can help reduce the number of attributes for model building to increase performance and model accuracy.

For more details on the algotithm implementation, parameters settings and characteristics of the ODM function itself consult the following Oracle documents: ODM Concepts, ODM Application Developer's Guide, and Oracle PL/SQL Packages: Data Mining, listed in the references below.

Value

If retrieve_outputs_to_R is TRUE, returns a list with the following elements:

model.model_settings	Table of settings used to build the model.
model.model_attributes	Table of attributes used to build the model.
ai.importance	Table of features along with their importance.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

See Also

RODM_drop_model

Examples

```
# Determine attribute importance for survival in the sinking of the Titanic
# based on pasenger's sex, age, class, etc.

## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid="rodm", pwd="rodm")

data(titanic3, package="PASWR")
db_titanic <- titanic3[,c("pclass", "survived", "sex", "age", "fare", "embarked")]
db_titanic[,"survived"] <- ifelse(db_titanic[,"survived"] == 1, "Yes", "No")
RODM_create_dbms_table(DB, "db_titanic")   # Push the table to the database

# Create the Oracle Data Mining Attribute Importance model
ai <- RODM_create_ai_model(
   database = DB,                      # Database ODBC connection
   data_table_name = "db_titanic",     # Database table containing the input dataset
   target_column_name = "survived",    # Target column name in data_table_name
   model_name = "TITANIC_AI_MODEL")    # Oracle Data Mining model name to create

attribute.importance <- ai$ai.importance
ai.vals <- as.vector(attribute.importance[,3])
names(ai.vals) <- as.vector(attribute.importance[,1])

#windows(height=8, width=12)
barplot(ai.vals, main="Relative survival importance of Titanic dataset attributes",
        ylab = "Relative Importance", xlab = "Attribute", cex.names=0.7)

ai        # look at the model details

RODM_drop_model(DB, "TITANIC_AI_MODEL")    # Drop the model
RODM_drop_dbms_table(DB, "db_titanic")     # Drop the database table

RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_create_assoc_model

Create an Association Rules model

Description

This function creates an Association Rules model.

Usage

```
RODM_create_assoc_model(database,
                        data_table_name,
                        case_id_column_name,
                        model_name = "AR_MODEL",
                        min_support = NULL,
                        min_confidence = NULL,
                        max_rule_length = NULL,
                        retrieve_outputs_to_R = TRUE,
                        leave_model_in_dbms = TRUE,
                        sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Database table/view containing the training dataset.
case_id_column_name	Row unique case identifier in data_table_name.
model_name	ODM Model name.
min_support	Setting that specifies the minimum support for assoc.
min_confidence	Setting that specifies the minimum confidence for assoc.
max_rule_length	Setting that specifies the maximum rule length for assoc.
retrieve_outputs_to_R	Flag controlling if the output results are moved to the R environment.
leave_model_in_dbms	Flag controlling if the model is deleted or left in RDBMS.
sql.log.file	File where to append the log of all the SQL calls made by this function.

Details

This function implements the apriori algorithm (Agrawal and Srikant 1994) to find frequent itemsets and generate Association Models (AM). It finds the co-occurrence of items in large volumes of "transactional" data such as in the case of market basket analysis. The rule is an implication where the appearance of a set of items in a transactional record implies another set of items. The groups

of items used to form rules must pass a minimum threshold according to how frequently they occur (support) and how often the consequent follows the antecedent (confidence). Association models generate all rules that have support and confidence greater than user-specified thresholds. The AM algorithm is efficient, and scales well with respect to the number of transactions, number of items, and number of itemsets and rules produced.

For more details on the algotithm implementation, parameters settings and characteristics of the ODM function itself consult the following Oracle documents: ODM Concepts, ODM Developer's Guide, Oracle SQL Packages: Data Mining, and Oracle Database SQL Language Reference (Data Mining functions), listed in the references below.

Value

If retrieve_outputs_to_R is TRUE, returns a list with the following elements:

`model.model_settings`
: Table of settings used to build the model.

`model.model_attributes`
: Table of attributes used to build the model.

`ar.rules`
: List of the association rules.

`ar.itemsets`
: List of the frequent itemsets.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

See Also

RODM_drop_model

Examples

```
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid= "rodm", pwd = "rodm")

data(satfruit, package="PASWR")
ards <- satfruit[,c("WH", "BA", "NAR", "COR", "SF", "VI", "PS", "ES", "AF", "CO", "AR", "AL", "OL")]  # Select sub
```

```
ards[,] <- ifelse(ards[,] == 0, NA, "YES")   # make it sparse, as required by ODM
n.rows <- length(ards[,1])           # Number of rows
row.id <- matrix(seq(1, n.rows), nrow=n.rows, ncol=1, dimnames= list(NULL, c("ROW_ID"))) # Row id
ards <- cbind(row.id, ards)          # Add row id to dataset
RODM_create_dbms_table(DB, "ards")   # Push the training table to the database

# Build the association rules model
ar <- RODM_create_assoc_model(
 database = DB,
 data_table_name = "ards",
 case_id_column_name = "ROW_ID")

# Inspect the contents of ar to find the rules and itemsets

RODM_drop_model(DB, "AR_MODEL")
RODM_drop_dbms_table(DB, "ards")

RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_create_dbms_table

Create a table in the Oracle Database

Description

Create and populate a table in the Oracle Database using a dataframe.

Usage

```
RODM_create_dbms_table(database,
                       data_table_name)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Data frame containing the input dataset. Also the name of the table to be created in the database.

Details

This functions moves a data frame to a relational table in the Oracle database using RODBC's sqlSave function. A table with the same name as the data frame will be created in the RDBMS to hold the data in the data frame. If the table already exists, it will first be dropped.

The data frame can contain attributes of numeric: double or integer, logical, character or factor type. The conversions performed by this function when transfering data to the RDBMS are as follows:

R data frame column type:	RDMS type:
numeric/double	FLOAT(126)
numeric/integer	NUMBER(38)
logical	VARCHAR2(255)
character	VARCHAR2(255)
factor	VARCHAR2(255)

Value

None.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

See Also

RODM_drop_dbms_table

Examples

```
## Not run:
data(titanic3, package="PASWR")
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid="rodm", pwd="rodm")

# Move the titanic3 data frame to a new table in the Oracle database
RODM_create_dbms_table(DB, "titanic3")

# Now drop the table (just to clean up from this example)
RODM_drop_dbms_table(DB, "titanic3")

RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_create_dt_model *Create a Decision Tree (DT) model*

Description

This function creates a Decision tree (DT).

Usage

```
RODM_create_dt_model(database,
                     data_table_name,
                     case_id_column_name = NULL,
                     target_column_name,
                     model_name = "DT_MODEL",
                     auto_data_prep = TRUE,
                     cost_matrix = NULL,
                     gini_impurity_metric = TRUE,
                     max_depth = NULL,
                     minrec_split = NULL,
                     minpct_split = NULL,
                     minrec_node = NULL,
                     minpct_node = NULL,
                     retrieve_outputs_to_R = TRUE,
                     leave_model_in_dbms = TRUE,
                     sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Database table/view containing the training dataset.
case_id_column_name	Row unique case identifier in data_table_name.
target_column_name	Target column name in data_table_name.
model_name	ODM Model name.
auto_data_prep	Whether or not ODM should invoke automatic data preparation for the build.
cost_matrix	User-specified cost matrix for the target classes.
gini_impurity_metric	Tree impurity metric: "IMPURITY_GINI" (default) or "IMPURITY_ENTROPY"
max_depth	Specifies the maximum depth of the tree, from root to leaf inclusive. The default is 7.
minrec_split	Specifies the minimum number of cases required in a node in order for a further split to be possible. Default is 20.
minpct_split	Specifies the minimum number of cases required in a node in order for a further split to be possible. Expressed as a percentage of all the rows in the training data. The default is 1 (1 per cent).
minrec_node	Specifies the minimum number of cases required in a child node. Default is 10.
minpct_node	Specifies the minimum number of cases required in a child node, expressed as a percentage of the rows in the training data. The default is 0.05 (.05 per cent).
retrieve_outputs_to_R	Flag controlling if the output results are moved to the R environment.

`leave_model_in_dbms`
: Flag controlling if the model is deleted or left in RDBMS.

`sql.log.file`
: File where to append the log of all the SQL calls made by this function.

Details

The Decision Tree algorithm produces accurate and interpretable models with relatively little user intervention and can be used for both binary and multiclass classification problems. The algorithm is fast, both at build time and apply time. The build process for Decision Tree is parallelized. Decision tree scoring is especially fast. The tree structure, created in the model build, is used for a series of simple tests. Each test is based on a single predictor. It is a membership test: either IN or NOT IN a list of values (categorical predictor); or LESS THAN or EQUAL TO some value (numeric predictor). The algorithm supports two homogeneity metrics, gini and entropy, for calculating the splits.

For more details on the algotithm implementation, parameters settings and characteristics of the ODM function itself consult the following Oracle documents: ODM Concepts, ODM Developer's Guide, Oracle SQL Packages: Data Mining, and Oracle Database SQL Language Reference (Data Mining functions), listed in the references below.

Value

If retrieve_outputs_to_R is TRUE, returns a list with the following elements:

`model.model_settings`
: Table of settings used to build the model.

`model.model_attributes`
: Table of attributes used to build the model.

`dt.distributions`
: Target class disctributions at each tree node.

`dt.nodes`
: Node summary information.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Oracle Database SQL Language Reference (Data Mining functions) 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF20030

See Also

RODM_apply_model, RODM_drop_model

Examples

```
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid= "rodm", pwd = "rodm")

# Predicting survival in the sinking of the Titanic based on pasenger's sex, age, class, etc.
data(titanic3, package="PASWR")                                    # Load survival data from Titanic
ds <- titanic3[,c("pclass", "survived", "sex", "age", "fare", "embarked")]  # Select subset of attributes
ds[,"survived"] <- ifelse(ds[,"survived"] == 1, "Yes", "No")             # Rename target values
n.rows <- length(ds[,1])                                                  # Number of rows
random_sample <- sample(1:n.rows, ceiling(n.rows/2))   # Split dataset randomly in train/test subsets
titanic_train <- ds[random_sample,]                            # Training set
titanic_test <-  ds[setdiff(1:n.rows, random_sample),]      # Test set
RODM_create_dbms_table(DB, "titanic_train")   # Push the training table to the database
RODM_create_dbms_table(DB, "titanic_test")    # Push the testing table to the database

dt <- RODM_create_dt_model(database = DB,    # Create ODM DT classification model
                           data_table_name = "titanic_train",
                           target_column_name = "survived",
                           model_name = "DT_MODEL")

dt2 <- RODM_apply_model(database = DB,    # Predict test data
                        data_table_name = "titanic_test",
                        model_name = "DT_MODEL",
                        supplemental_cols = "survived")

print(dt2$model.apply.results[1:10,])                           # Print example of prediction results
actual <- dt2$model.apply.results[, "SURVIVED"]
predicted <- dt2$model.apply.results[, "PREDICTION"]
probs <- as.real(as.character(dt2$model.apply.results[, "'Yes'"]))
table(actual, predicted, dnn = c("Actual", "Predicted"))             # Confusion matrix
library(verification)
perf.auc <- roc.area(ifelse(actual == "Yes", 1, 0), probs)          # Compute ROC and plot
auc.roc <- signif(perf.auc$A, digits=3)
auc.roc.p <- signif(perf.auc$p.value, digits=3)
roc.plot(ifelse(actual == "Yes", 1, 0), probs, binormal=T, plot="both", xlab="False Positive Rate",
         ylab="True Postive Rate", main= "Titanic survival ODM DT model ROC Curve")
text(0.7, 0.4, labels= paste("AUC ROC:", signif(perf.auc$A, digits=3)))
text(0.7, 0.3, labels= paste("p-value:", signif(perf.auc$p.value, digits=3)))

dt        # look at the model details

RODM_drop_model(DB, "DT_MODEL")             # Drop the model
RODM_drop_dbms_table(DB, "titanic_train")   # Drop the database table
RODM_drop_dbms_table(DB, "titanic_test")    # Drop the database table

RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_create_glm_model *Create an ODM Generalized Linear Model*

Description

This function creates an ODM generalized linear model.

Usage

```
RODM_create_glm_model(database,
                      data_table_name,
                      case_id_column_name = NULL,
                      target_column_name,
                      model_name = "GLM_MODEL",
                      mining_function = "classification",
                      auto_data_prep = TRUE,
                      class_weights = NULL,
                      weight_column_name = NULL,
                      conf_level = NULL,
                      reference_class_name = NULL,
                      missing_value_treatment = NULL,
                      ridge_regression = NULL,
                      ridge_value = NULL,
                      vif_for_ridge = NULL,
                      diagnostics_table_name = NULL,
                      retrieve_outputs_to_R = TRUE,
                      leave_model_in_dbms = TRUE,
                      sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Database table/view containing the training dataset.
case_id_column_name	Row unique case identifier in data_table_name.
target_column_name	Target column name in data_table_name.
model_name	ODM Model name.
mining_function	Type of mining function for GLM model: "classification" (default) or "regression".
auto_data_prep	Whether or not ODM should invoke automatic data preparation for the build.
class_weights	User-specified weights for the target classes.

`weight_column_name`
: Name of a column in data_table_name that contains a weighting factor for the rows. Row weights can be used as a compact representation of repeated rows, and can also be used to emphasize certain rows during model construction.

`conf_level`
: The confidence level for coefficient confidence intervals.

`reference_class_name`
: The target value to be used as the reference value in a logistic regression model. Probabilities will be produced for the other (non-reference) class. By default, the algorithm chooses the value with the highest prevalence (the most cases) for the reference class.

`missing_value_treatment`
: How to handle missing values. Either replace by the mean or mode by setting ODMS_MISSING_VALUE_MEAN_MODE, or delete the entire row when a missing value is present by setting ODMS_MISSING_VALUE_DELETE_ROW.

`ridge_regression`
: Whether or not ridge regression will be enabled. By default, the algorithm determines whether or not to use ridge. You can explicitly enable ridge by setting GLMS_RIDGE_REGRESSION to GLMS_RIDGE_REG_ENABLE. Ridge applies to both regression and classification mining functions. When ridge is enabled, no prediction bounds are produced by the PREDICTION_BOUNDS SQL operator.

`ridge_value`
: The value for the ridge parameter used by the algorithm. This setting is only used when you explicitly enable ridge regression by setting GLMS_RIDGE_REGRESSION to GLMS_RIDGE_REG_ENABLE. If ridge regression is enabled internally by the algorithm, the ridge parameter is determined by the algorithm.

`vif_for_ridge`
: (Linear regression only) Whether or not to produce Variance Inflation Factor (VIF) statistics when ridge is being used. By default, VIF is not produced when ridge is enabled.When you explicitly enable ridge regression by setting GLMS_RIDGE_REGRESSION to GLMS_RIDGE_REG_ENABLE, you can request VIF statistics by setting GLMS_VIF_FOR_RIDGE to GLMS_VIF_RIDGE_ENABLE; the algorithm will produce VIF if enough system resources are available.

`diagnostics_table_name`
: Non-existing database table to hold per-row diagnostic information. Requires a case_id_column_name to be specified. The table will remain in the database and must be dropped explicitly when desired.;

`retrieve_outputs_to_R`
: Flag controlling if the output results are moved to the R environment.

`leave_model_in_dbms`
: Flag controlling if the model is deleted or left in RDBMS.

`sql.log.file`
: File where to append the log of all the SQL calls made by this function.

Details

Generalized linear models (GLM) implements logistic regression for classification of binary targets and linear regression for continuous targets. GLM classification supports confidence bounds for prediction probabilities. GLM regression supports confidence bounds for predictions and supports linear and logistic regression with the logit link and binomial variance functions. Ridge regression

is a technique that compensates for multicollinearity. Oracle Data Mining supports ridge regression for both regression and classification mining functions. The algorithm automatically uses ridge if it detects singularity (exact multicollinearity) in the data.

For more details on the algotithm implementation, parameters settings and characteristics of the ODM function itself consult the following Oracle documents: ODM Concepts, ODM Developer's Guide, Oracle SQL Packages: Data Mining, and Oracle Database SQL Language Reference (Data Mining functions), listed in the references below.

Value

If retrieve_outputs_to_R is TRUE, returns a list with the following elements:

`model.model_settings`
: Table of settings used to build the model.

`model.model_attributes`
: Table of attributes used to build the model.

`glm.globals`
: Global details for the GLM model.

`glm.coefficients`
: The coefficients of the GLM model, along with more per-attribute information.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Dobson, Annette J. and Barnett, Adrian G. (2008) An Introduction to Generalized Linear Models, Third Edition. Texts in Statistical Science ,77 . Chapman & Hall/CRC Press, Boca Raton, FL.

B. L. Milenova, J. S. Yarmus, and M. M. Campos. SVM in oracle database 10g: removing the barriers to widespread adoption of support vector machines. In Proceedings of the "31st international Conference on Very Large Data Bases" (Trondheim, Norway, August 30 - September 02, 2005). pp1152-1163, ISBN:1-59593-154-6.

Milenova, B.L. Campos, M.M., Mining high-dimensional data for information fusion: a database-centric approach 8th International Conference on Information Fusion, 2005. Publication Date: 25-28 July 2005. ISBN: 0-7803-9286-8. John Shawe-Taylor & Nello Cristianini. Support Vector Machines and other kernel-based learning methods. Cambridge University Press, 2000.

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Oracle Database SQL Language Reference (Data Mining functions) 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF20030

See Also

RODM_apply_model, RODM_drop_model

Examples

```
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid= "rodm", pwd = "rodm")

### GLM Classification

# Predicting survival in the sinking of the Titanic based on pasenger's sex, age, class, etc.

data(titanic3, package="PASWR")                                    # Load survival data from Titanic
ds <- titanic3[,c("pclass", "survived", "sex", "age", "fare", "embarked")]  # Select subset of attributes
ds[,"survived"] <- ifelse(ds[,"survived"] == 1, "Yes", "No")             # Rename target values
n.rows <- length(ds[,1])                                                  # Number of rows
random_sample <- sample(1:n.rows, ceiling(n.rows/2))   # Split dataset randomly in train/test subsets
titanic_train <- ds[random_sample,]                           # Training set
train.rows <- length(titanic_train[,1])                                        # Number of rows
row.id <- matrix(seq(1, train.rows), nrow=train.rows, ncol=1, dimnames= list(NULL, c("ROW_ID"))) # Row id
titanic_train <- cbind(row.id, titanic_train)                                          # Add row id to dataset
titanic_test <-  ds[setdiff(1:n.rows, random_sample),]      # Test set
RODM_create_dbms_table(DB, "titanic_train")   # Push the training table to the database
RODM_create_dbms_table(DB, "titanic_test")    # Push the testing table to the database

# Weight one class more heavily than the other
weights <- data.frame(
            target_value = c("Yes", "No"),
            class_weight = c(1, 10))

glm <- RODM_create_glm_model(database = DB,    # Create ODM GLM classification model
                             data_table_name = "titanic_train",
                             case_id_column_name = "ROW_ID",
                             target_column_name = "survived",
                             model_name = "GLM_MODEL",
                             class_weights = weights,
                             diagnostics_table_name = "GLM_DIAG",
                             mining_function = "classification")

glm2 <- RODM_apply_model(database = DB,    # Predict test data
                         data_table_name = "titanic_test",
                         model_name = "GLM_MODEL",
                         supplemental_cols = "survived")

print(glm2$model.apply.results[1:10,])                           # Print example of prediction results
actual <- glm2$model.apply.results[, "SURVIVED"]
predicted <- glm2$model.apply.results[, "PREDICTION"]
probs <- as.real(as.character(glm2$model.apply.results[, "'Yes'"]))
table(actual, predicted, dnn = c("Actual", "Predicted"))              # Confusion matrix
library(verification)
perf.auc <- roc.area(ifelse(actual == "Yes", 1, 0), probs)          # Compute ROC and plot
auc.roc <- signif(perf.auc$A, digits=3)
```

```
auc.roc.p <- signif(perf.auc$p.value, digits=3)
roc.plot(ifelse(actual == "Yes", 1, 0), probs, binormal=T, plot="both", xlab="False Positive Rate",
         ylab="True Postive Rate", main= "Titanic survival ODM GLM model ROC Curve")
text(0.7, 0.4, labels= paste("AUC ROC:", signif(perf.auc$A, digits=3)))
text(0.7, 0.3, labels= paste("p-value:", signif(perf.auc$p.value, digits=3)))

glm       # look at the model details

# access and look at the per-row diagnostics from model training
diaginfo <- sqlQuery(DB, query = "SELECT * FROM GLM_DIAG")
diaginfo

RODM_drop_model(DB, "GLM_MODEL")            # Drop the model
RODM_drop_dbms_table(DB, "GLM_DIAG")        # Drop the diagnostics table
RODM_drop_dbms_table(DB, "titanic_train")   # Drop the database table
RODM_drop_dbms_table(DB, "titanic_test")    # Drop the database table

## End(Not run)

### GLM Regression
## Not run:
x1 <- 2 * runif(200)
noise <- 3 * runif(200) - 1.5
y1 <- 2 + 2*x1 + x1*x1 + noise
dataset <- data.frame(x1, y1)
names(dataset) <- c("X1", "Y1")
RODM_create_dbms_table(DB, "dataset")   # Push the training table to the database

glm <- RODM_create_glm_model(database = DB,    # Create ODM GLM model
                             data_table_name = "dataset",
                             target_column_name = "Y1",
                             mining_function = "regression")

glm2 <- RODM_apply_model(database = DB,    # Predict training data
                             data_table_name = "dataset",
                             model_name = "GLM_MODEL",
                             supplemental_cols = "X1")
windows(height=8, width=12)
plot(x1, y1, pch=20, col="blue")
points(x=glm2$model.apply.results[, "X1"],
       glm2$model.apply.results[, "PREDICTION"], pch=20, col="red")
legend(0.5, 9, legend = c("actual", "GLM regression"), pch = c(20, 20),
               col = c("blue", "red"),
               pt.bg =  c("blue", "red"), cex = 1.20, pt.cex=1.5, bty="n")

RODM_drop_model(DB, "GLM_MODEL")            # Drop the model
RODM_drop_dbms_table(DB, "dataset")   # Drop the database table
RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_create_kmeans_model *Create a Hierarchical k-means model*

Description

This function creates a Hierarchical k-means model.

Usage

```
RODM_create_kmeans_model(database,
                         data_table_name,
                         case_id_column_name = NULL,
                         model_name = "KM_MODEL",
                         auto_data_prep = TRUE,
                         num_clusters = NULL,
                         block_growth = NULL,
                         conv_tolerance = NULL,
                         euclidean_distance = TRUE,
                         iterations = NULL,
                         min_pct_attr_support = NULL,
                         num_bins = NULL,
                         variance_split = TRUE,
                         retrieve_outputs_to_R = TRUE,
                         leave_model_in_dbms = TRUE,
                         sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Database table/view containing the training dataset.
case_id_column_name	Row unique case identifier in data_table_name.
model_name	ODM Model name.
auto_data_prep	Whether or not ODM should invoke automatic data preparation for the build.
num_clusters	Setting that specifies the number of clusters for a clustering model.
block_growth	Setting that specifies the growth factor for memory to hold cluster data for k-Means.
conv_tolerance	Setting that specifies the convergence tolerance for k-Means.
euclidean_distance	Distance function (cosine, euclidean or fast_cosine).
iterations	Setting that specifies the number of iterations for k-Means.
min_pct_attr_support	Setting that specifies the minimum percent required for attributes in rules.

`num_bins` Setting that specifies the number of histogram bins k-Means.

`variance_split` Setting that specifies the split criterion for k-Means.

`retrieve_outputs_to_R`
Flag controlling if the output results are moved to the R environment.

`leave_model_in_dbms`
Flag controlling if the model is deleted or left in RDBMS.

`sql.log.file` File where to append the log of all the SQL calls made by this function.

Details

The algorithm k-means (kmeans) uses a distance-based similarity measure and tessellates the data space creating hierarchies. It handles large data volumes via summarization and supports sparse data. It is especially useful when the dataset has a moderate number of numerical attributes and one has a predetermined number of clusters. The main parameters settings correspond to the choice of distance function (e.g., Euclidean or cosine), number of iterations, convergence tolerance and split criterion.

For more details on the algotithm implementation, parameters settings and characteristics of the ODM function itself consult the following Oracle documents: ODM Concepts, ODM Developer's Guide and Oracle SQL Packages: Data Mining, and Oracle Database SQL Language Reference (Data Mining functions), listed in the references below.

Value

If retrieve_outputs_to_R is TRUE, returns a list with the following elements:

`model.model_settings`
Table of settings used to build the model.

`model.model_attributes`
Table of attributes used to build the model.

`km.clusters` General per-cluster information.

`km.taxonomy` Parent-child cluster relationship.

`km.centroid` Per cluster-attribute centroid information.

`km.histogram` Per cluster-attribute hitogram information.

`km.rule` Cluster rules.

`km.leaf_cluster_count`
Leaf clusters with support.

`km.assignment` Assignment of training data to clusters (with probability).

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Oracle Database SQL Language Reference (Data Mining functions) 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF20030

See Also

RODM_apply_model, RODM_drop_model

Examples

```
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid= "rodm", pwd = "rodm")

### Clustering a 2D multi-Gaussian distribution of points into clusters

set.seed(seed=6218945)
X1 <- c(rnorm(100, mean = 2, sd = 1), rnorm(100, mean = 8, sd = 2), rnorm(100, mean = 5, sd = 0.6),
      rnorm(100, mean = 4, sd = 1), rnorm(100, mean = 10, sd = 1)) # Create and merge 5 Gaussian distributions
Y1 <- c(rnorm(100, mean = 1, sd = 2), rnorm(100, mean = 4, sd = 1.5), rnorm(100, mean = 6, sd = 0.5),
        rnorm(100, mean = 3, sd = 0.2), rnorm(100, mean = 2, sd = 1))
ds <- data.frame(cbind(X1, Y1))
n.rows <- length(ds[,1])                                          # Number of rows
row.id <- matrix(seq(1, n.rows), nrow=n.rows, ncol=1, dimnames= list(NULL, c("ROW_ID"))) # Row id
ds <- cbind(row.id, ds)                                           # Add row id to dataset
RODM_create_dbms_table(DB, "ds")

km <- RODM_create_kmeans_model(
   database = DB,                  # database ODBC channel identifier
   data_table_name = "ds",         # data frame containing the input dataset
   case_id_column_name = "ROW_ID", # case id to enable assignments during build
   num_clusters = 5)

km2 <- RODM_apply_model(
   database = DB,                  # database ODBC channel identifier
   data_table_name = "ds",         # data frame containing the input dataset
   model_name = "KM_MODEL",
   supplemental_cols = c("X1","Y1"))

x1a <- km2$model.apply.results[, "X1"]
y1a <- km2$model.apply.results[, "Y1"]
clu <- km2$model.apply.results[, "CLUSTER_ID"]
c.numbers <- unique(as.numeric(clu))
```

```
c.assign <- match(clu, c.numbers)
color.map <- c("blue", "green", "red", "orange", "purple")
color <- color.map[c.assign]
nf <- layout(matrix(c(1, 2), 1, 2, byrow=T), widths = c(1, 1), heights = 1, respect = FALSE)
plot(x1a, y1a, pch=20, col=1, xlab="X1", ylab="Y1", main="Original Data Points")
plot(x1a, y1a, pch=20, type = "n", xlab="X1", ylab="Y1", main="After kmeans clustering")
for (i in 1:n.rows) {
   points(x1a[i], y1a[i], col= color[i], pch=20)
}
legend(5, -0.5, legend=c("Cluster 1", "Cluster 2", "Cluster 3", "Cluster 4", "Cluster 5"), pch = rep(20, 5),
       col = color.map, pt.bg = color.map, cex = 0.8, pt.cex=1, bty="n")

km        # look at the model details and cluster assignments

RODM_drop_model(DB, "KM_MODEL")   # Drop the model
RODM_drop_dbms_table(DB, "ds")    # Drop the database table

RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_create_model *Create an ODM model*

Description

Helper function to create an Oracle Data Mining model

Usage

```
RODM_create_model(database,
                  model_name,
                  mining_function_type,
                  data_table_name,
                  case_id_column_name = "",
                  target_column_name = "",
                  retrieve_outputs_to_R = TRUE,
                  sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
model_name	ODM Model name.
mining_function_type	Mining function to use, e.g., "dbms_data_mining.classification"
data_table_name	Database table/view containing the training dataset.

`case_id_column_name`
: Row unique case identifier in data_table_name.

`target_column_name`
: Name of the target column (for supervised models).

`retrieve_outputs_to_R`
: Flag controlling if the output results are moved to the R environment.

`sql.log.file`
: File to append the log of all the SQL calls made by this function.

Details

This is the generic function for creating ODM models.

It is not intended to be called directly by the user, but rather via the RODM_create_XX_model functions.

Value

If retrieve_outputs_to_R is TRUE, returns a list with the following elements:

`model.model_settings`
: Table of settings used to build the model.

`model.model_attributes`
: Table of attributes used to build the model.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

RODM_create_nb_model *Create a Naive Bayes model*

Description

This function creates an Oracle Data Mining Naive Bayes model.

Usage

```
RODM_create_nb_model(database,
                     data_table_name,
                     case_id_column_name = NULL,
                     target_column_name,
                     model_name = "NB_MODEL",
                     auto_data_prep = TRUE,
                     class_priors = NULL,
                     retrieve_outputs_to_R = TRUE,
                     leave_model_in_dbms = TRUE,
                     sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Database table/view containing the training dataset.
case_id_column_name	Row unique case identifier in data_table_name.
target_column_name	Target column name in data_table_name.
model_name	ODM Model name.
auto_data_prep	Whether or not ODM should invoke automatic data preparation for the build.
class_priors	User-specified priors for the target classes.
retrieve_outputs_to_R	Flag controlling if the output results are moved to the R environment.
leave_model_in_dbms	Flag controlling if the model is deleted or left in RDBMS.
sql.log.file	File where to append the log of all the SQL calls made by this function.

Details

Naive Bayes (NB) for classification makes predictions using Bayes' Theorem assuming that each attribute is conditionally independent of the others given a particular value of the target (Duda, Hart and Stork 2000). NB provides a very flexible general classifier for fast model building and scoring that can be used for both binary and multi-class classification problems.

For more details on the algotithm implementation, parameters settings and characteristics of the ODM function itself consult the following Oracle documents: ODM Concepts, ODM Application Developer's Guide, Oracle SQL Packages: Data Mining, and Oracle Database SQL Language Reference (Data Mining functions), listed in the references below.

Value

If retrieve_outputs_to_R is TRUE, returns a list with the following elements:

model.model_settings	Table of settings used to build the model.

model.model_attributes
: Table of attributes used to build the model.

nb.conditionals
: Table of conditional probabilities.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Oracle Database SQL Language Reference (Data Mining functions) 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF20030

See Also

RODM_apply_model, RODM_drop_model

Examples

```
# Predicting survival in the sinking of the Titanic based on pasenger's sex, age, class, etc.
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid= "rodm", pwd = "rodm")

data(titanic3, package="PASWR")                                    # Load survival data from Titanic
ds <- titanic3[,c("pclass", "survived", "sex", "age", "fare", "embarked")]  # Select subset of attributes
ds[,"survived"] <- ifelse(ds[,"survived"] == 1, "Yes", "No")             # Rename target values
n.rows <- length(ds[,1])                                                  # Number of rows
set.seed(seed=6218945)
random_sample <- sample(1:n.rows, ceiling(n.rows/2))   # Split dataset randomly in train/test subsets
titanic_train <- ds[random_sample,]                            # Training set
titanic_test <-  ds[setdiff(1:n.rows, random_sample),]      # Test set

RODM_create_dbms_table(DB, "titanic_train")   # Push the training table to the database
RODM_create_dbms_table(DB, "titanic_test")    # Push the testing table to the database

# If the target distribution does not reflect the actual distribution due
# to specialized sampling, specify priors for the model
priors <- data.frame(
           target_value = c("Yes", "No"),
           prior_probability = c(0.1, 0.9))
```

```
# Create an ODM Naive Bayes model
nb <- RODM_create_nb_model(
   database = DB,                      # Database ODBC channel identifier
   model_name = "titanic_nb_model",    # ODM model name
  data_table_name = "titanic_train", # (in quotes) Data frame or database table containing the input dataset
   class_priors = priors,              # user-specified priors
   target_column_name = "survived")    # Target column name in data_table_name

# Predict test data using the Naive Bayes model
nb2 <- RODM_apply_model(
   database = DB,                     # Database ODBC channel identifier
   data_table_name = "titanic_test", # Database table containing the input dataset
   model_name = "titanic_nb_model",  # ODM model name
   supplemental_cols = "survived")   # Carry the target column to the output for analysis

# Compute contingency matrix, performance statistics and ROC curve
print(nb2$model.apply.results[1:10,])                          # Print example of prediction results
actual <- nb2$model.apply.results[, "SURVIVED"]
predicted <- nb2$model.apply.results[, "PREDICTION"]
probs <- as.real(as.character(nb2$model.apply.results[, "'Yes'"]))
table(actual, predicted, dnn = c("Actual", "Predicted"))              # Confusion matrix

library(verification)
perf.auc <- roc.area(ifelse(actual == "Yes", 1, 0), probs)          # Compute ROC and plot
auc.roc <- signif(perf.auc$A, digits=3)
auc.roc.p <- signif(perf.auc$p.value, digits=3)
roc.plot(ifelse(actual == "Yes", 1, 0), probs, binormal=T, plot="both", xlab="False Positive Rate",
         ylab="True Postive Rate", main= "Titanic survival ODM NB model ROC Curve")
text(0.7, 0.4, labels= paste("AUC ROC:", signif(perf.auc$A, digits=3)))
text(0.7, 0.3, labels= paste("p-value:", signif(perf.auc$p.value, digits=3)))

nb        # look at the model details

RODM_drop_model(DB, "titanic_nb_model")     # Drop the model
RODM_drop_dbms_table(DB, "titanic_train")   # Drop the training table in the database
RODM_drop_dbms_table(DB, "titanic_test")    # Drop the testing table in the database

RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_create_oc_model *Create an O-cluster model*

Description

This function creates a O-cluster model.

Usage

```
RODM_create_oc_model(database,
                         data_table_name,
                         case_id_column_name,
                         model_name = "OC_MODEL",
                         auto_data_prep = TRUE,
                         num_clusters = NULL,
                         max_buffer = NULL,
                         sensitivity = NULL,
                         retrieve_outputs_to_R = TRUE,
                         leave_model_in_dbms = TRUE,
                         sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Database table/view containing the training dataset.
case_id_column_name	Row unique case identifier in data_table_name.
model_name	ODM Model name.
auto_data_prep	Whether or not ODM should invoke automatic data preparation for the build.
num_clusters	Setting that specifies the number of clusters for the clustering model.
max_buffer	Buffer size for O-Cluster. Default is 50,000.
sensitivity	A fraction that specifies the peak density required for separating a new cluster. The fraction is related to the global uniform density. Default is 0.5.
retrieve_outputs_to_R	Flag controlling if the output results are moved to the R environment.
leave_model_in_dbms	Flag controlling if the model is deleted or left in RDBMS.
sql.log.file	File where to append the log of all the SQL calls made by this function.

Details

The O-Cluster algorithm creates a hierarchical grid-based clustering model, that is, it creates axis-parallel (orthogonal) partitions in the input attribute space. The algorithm operates recursively. The resulting hierarchical structure represents an irregular grid that tessellates the attribute space into clusters. The resulting clusters define dense areas in the attribute space.

The clusters are described by intervals along the attribute axes and the corresponding centroids and histograms. A parameter called sensitivity defines a baseline density level. Only areas with peak density above this baseline level can be identified as clusters.

The k-means algorithm tessellates the space even when natural clusters may not exist. For example, if there is a region of uniform density, k-Means tessellates it into n clusters (where n is specified by the user). O-Cluster separates areas of high density by placing cutting planes through areas of low

density. O-Cluster needs multi-modal histograms (peaks and valleys). If an area has projections with uniform or monotonically changing density, O-Cluster does not partition it.

The clusters discovered by O-Cluster are used to generate a Bayesian probability model that is then used during scoring (model apply) for assigning data points to clusters. The generated probability model is a mixture model where the mixture components are represented by a product of independent normal distributions for numerical attributes and multinomial distributions for categorical attributes.

Keep the following in mind if you choose to prepare the data for O-Cluster: 1. O-Cluster does not necessarily use all the input data when it builds a model. It reads the data in batches (the default batch size is 50000). It will only read another batch if it believes, based on statistical tests, that there may still exist clusters that it has not yet uncovered. 2. Because O-Cluster may stop the model build before it reads all of the data, it is highly recommended that the data be randomized. 3. Binary attributes should be declared as categorical. O-Cluster maps categorical data to numerical values. 4. The use of Oracle Data Mining's equi-width binning transformation with automated estimation of the required number of bins is highly recommended. 5. The presence of outliers can significantly impact clustering algorithms. Use a clipping transformation before binning or normalizing. Outliers with equi-width binning can prevent O-Cluster from detecting clusters. As a result, the whole population appears to falls within a single cluster.

For more details on the algotithm implementation, parameters settings and characteristics of the ODM function itself consult the following Oracle documents: ODM Concepts, ODM Developer's Guide and Oracle SQL Packages: Data Mining, and Oracle Database SQL Language Reference (Data Mining functions), listed in the references below.

Value

If retrieve_outputs_to_R is TRUE, returns a list with the following elements:

`model.model_settings`
: Table of settings used to build the model.

`model.model_attributes`
: Table of attributes used to build the model.

`oc.clusters`
: General per-cluster information.

`oc.split_predicate`
: Cluster split predicates.

`oc.taxonomy`
: Parent-child cluster relationship.

`oc.centroid`
: Per cluster-attribute centroid information.

`oc.histogram`
: Per cluster-attribute hitogram information.

`oc.rule`
: Cluster rules.

`oc.leaf_cluster_count`
: Leaf clusters with support.

`oc.assignment`
: Assignment of training data to clusters (with probability).

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

B.L. Milenova and M.M. Campos, Clustering Large Databases with Numeric and Nominal Values Using Orthogonal Projection, Proceeding of the 29th VLDB Conference, Berlin, Germany (2003).

Oracle9i O-Cluster: Scalable Clustering of Large High Dimensional Data Sets http://www.oracle.com/technology/products/bi

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Oracle Database SQL Language Reference (Data Mining functions) 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF20030

See Also

RODM_apply_model, RODM_drop_model

Examples

```
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid= "rodm", pwd = "rodm")

### Clustering a 2D multi-Gaussian distribution of points into clusters

set.seed(seed=6218945)
X1 <- c(rnorm(100, mean = 2, sd = 1), rnorm(100, mean = 8, sd = 2), rnorm(100, mean = 5, sd = 0.6),
      rnorm(100, mean = 4, sd = 1), rnorm(100, mean = 10, sd = 1)) # Create and merge 5 Gaussian distributions
Y1 <- c(rnorm(100, mean = 1, sd = 2), rnorm(100, mean = 4, sd = 1.5), rnorm(100, mean = 6, sd = 0.5),
        rnorm(100, mean = 3, sd = 0.2), rnorm(100, mean = 2, sd = 1))
ds <- data.frame(cbind(X1, Y1))
n.rows <- length(ds[,1])                                          # Number of rows
row.id <- matrix(seq(1, n.rows), nrow=n.rows, ncol=1, dimnames= list(NULL, c("ROW_ID"))) # Row id
ds <- cbind(row.id, ds)                                           # Add row id to dataset
RODM_create_dbms_table(DB, "ds")

oc <- RODM_create_oc_model(
   database = DB,                  # database ODBC channel identifier
   data_table_name = "ds",         # data frame containing the input dataset
   case_id_column_name = "ROW_ID", # case id to enable assignments during build
   num_clusters = 5)

oc2 <- RODM_apply_model(
   database = DB,                  # database ODBC channel identifier
   data_table_name = "ds",         # data frame containing the input dataset
   model_name = "OC_MODEL",
   supplemental_cols = c("X1","Y1"))
```

```
x1a <- oc2$model.apply.results[, "X1"]
y1a <- oc2$model.apply.results[, "Y1"]
clu <- oc2$model.apply.results[, "CLUSTER_ID"]
c.numbers <- unique(as.numeric(clu))
c.assign <- match(clu, c.numbers)
color.map <- c("blue", "green", "red")
color <- color.map[c.assign]
nf <- layout(matrix(c(1, 2), 1, 2, byrow=T), widths = c(1, 1), heights = 1, respect = FALSE)
plot(x1a, y1a, pch=20, col=1, xlab="X1", ylab="Y1", main="Original Data Points")
plot(x1a, y1a, pch=20, type = "n", xlab="X1", ylab="Y1", main="After OC clustering")
for (i in 1:n.rows) {
   points(x1a[i], y1a[i], col= color[i], pch=20)
}
legend(5, -0.5, legend=c("Cluster 1", "Cluster 2", "Cluster 3"), pch = rep(20, 3),
       col = color.map, pt.bg = color.map, cex = 0.8, pt.cex=1, bty="n")

oc        # look at the model details and cluster assignments

RODM_drop_model(DB, "OC_MODEL")   # Drop the database table
RODM_drop_dbms_table(DB, "ds")    # Drop the database table

RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_create_svm_model *Create an ODM Support Vector Machine model*

Description

This function creates an ODM Support Vector Machine model.

Usage

```
RODM_create_svm_model(database,
                      data_table_name,
                      case_id_column_name = NULL,
                      target_column_name = NULL,
                      model_name = "SVM_MODEL",
                      mining_function = "classification",
                      auto_data_prep = TRUE,
                      class_weights = NULL,
                      active_learning = TRUE,
                      complexity_factor = NULL,
                      conv_tolerance = NULL,
                      epsilon = NULL,
                      kernel_cache_size = NULL,
                      kernel_function = NULL,
                      outlier_rate = NULL,
```

```
                    std_dev = NULL,
                    retrieve_outputs_to_R = TRUE,
                    leave_model_in_dbms = TRUE,
                    sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Database table/view containing the training dataset.
case_id_column_name	Row unique case identifier in data_table_name.
target_column_name	Target column name in data_table_name.
model_name	ODM Model name.
mining_function	Type of mining function for SVM model: "classification" (default), "regression" or "anomaly_detection".
auto_data_prep	Whether or not ODM should invoke automatic data preparation for the build.
class_weights	User-specified weights for the target classes.
active_learning	Whether or not ODM should use active learning.
complexity_factor	Setting that specifies the complexity factor for SVM. The default is NULL.
conv_tolerance	Setting that specifies tolerance for SVM. The default is 0.001.
epsilon	Regularization setting for regression, similar to complexity factor. Epsilon specifies the allowable residuals, or noise, in the data. The default is NULL.
kernel_cache_size	Setting that specifiefs the Gaussian kernel cache size (bytes) for SVM. The default is 5e+07.
kernel_function	Setting for specifying the kernel function for SVM (Gaussian or Linear). The default is to let ODM decide based on the data.
outlier_rate	A setting specifying the desired rate of outliers in the training data for anomaly detection one-class SVM. The default is NULL.
std_dev	A setting that specifies the standard deviation for the SVM Gaussian kernel. The default is NULL (algorithm generated).
retrieve_outputs_to_R	Flag controlling if the output results are moved to the R environment.
leave_model_in_dbms	Flag controlling if the model is deleted or left in RDBMS.
sql.log.file	File where to append the log of all the SQL calls made by this function.

Details

Support Vector Machines (SVMs) for classification belong to a class of algorithms known as "kernel" methods (Cristianini and Shawe-Taylor 2000). Kernel methods rely on applying predefined functions (kernels) to the input data. The boundary is a function of the predictor values. The key concept behind SVMs is that the points lying closest to the boundary, i.e., the support vectors, can be used to define the boundary. The goal of the SVM algorithm is to identify the support vectors and assign them weights that produce an optimal, largest margin, class-separating boundary.

This function enables to call Oracle Data Mining's SVM implementation (for details see Milenova et al 2005) that supports classification, regression and anomaly detection (one-class classification) with linear or Gaussian kernels and an automatic and efficient estimation of the complexity factor (C) and standard deviation (sigma). It also supports sparse data, which makes it very efficient for problems such as text mining. Support Vector Machines (SVMs) for regression utilizes an epsilon-insensitive loss function and works particularly well for high-dimensional noisy data. The scalability and usability of this function are particularly useful when deploying predictive models in a production database data mining system. The implementation also supports Active learning which forces the SVM algorithm to restrict learning to the most informative training examples and not to attempt to use the entire body of data. In most cases, the resulting models have predictive accuracy comparable to that of a standard (exact) SVM model. Active learning provides a significant improvement in both linear and Gaussian SVM models, whether for classification, regression, or anomaly detection. However, active learning is especially advantageous when using the Gaussian kernel, because nonlinear models can otherwise grow to be very large and can place considerable demands on memory and other system resources.

The SVM algorithm operates natively on numeric attributes. The function automatically "explodes" categorical data into a set of binary attributes, one per category value. For example, a character column for marital status with values married or single would be transformed to two numeric attributes: married and single. The new attributes could have the value 1 (true) or 0 (false). When there are missing values in columns with simple data types (not nested), SVM interprets them as missing at random. The algorithm automatically replaces missing categorical values with the mode and missing numerical values with the mean. SVM requires the normalization of numeric input. Normalization places the values of numeric attributes on the same scale and prevents attributes with a large original scale from biasing the solution. Normalization also minimizes the likelihood of overflows and underflows. Furthermore, normalization brings the numerical attributes to the same scale (0,1) as the exploded categorical data. The SVM algorithm automatically handles missing value treatment and the transformation of categorical data, but normalization and outlier detection must be handled manually.

For more details on the algotithm implementation see Milenova et al 2005. For details on the parameters and characteristics of the ODM function itself consult the ODM Concepts, the ODM Developer's Guide and the Oracle SQL Packages: Data Mining documents in the references below.

Value

If retrieve_outputs_to_R is TRUE, returns a list with the following elements:

`model.model_settings`
: Table of settings used to build the model.

`model.model_attributes`
: Table of attributes used to build the model.

If the model that was built uses a linear kernel, then the following is additionally returned:

`svm.coefficients`
: The coefficients of the SVM model, one for each input attribute. If auto_data_prep, then these coefficients will be in the transformed space (after automatic outlier-aware normalization is applied).

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

B. L. Milenova, J. S. Yarmus, and M. M. Campos. SVM in oracle database 10g: removing the barriers to widespread adoption of support vector machines. In Proceedings of the "31st international Conference on Very Large Data Bases" (Trondheim, Norway, August 30 - September 02, 2005). pp1152-1163, ISBN:1-59593-154-6.

Milenova, B.L. Campos, M.M., Mining high-dimensional data for information fusion: a database-centric approach 8th International Conference on Information Fusion, 2005. Publication Date: 25-28 July 2005. ISBN: 0-7803-9286-8. John Shawe-Taylor & Nello Cristianini. Support Vector Machines and other kernel-based learning methods. Cambridge University Press, 2000.

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Oracle Database SQL Language Reference (Data Mining functions) 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF20030

See Also

RODM_apply_model, RODM_drop_model

Examples

```
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid= "rodm", pwd = "rodm")

# Separating three Gaussian classes in 2D

X1 <- c(rnorm(200, mean = 2, sd = 1), rnorm(300, mean = 8, sd = 2), rnorm(300, mean = 5, sd = 0.6))
Y1 <- c(rnorm(200, mean = 1, sd = 2), rnorm(300, mean = 4, sd = 1.5), rnorm(300, mean = 6, sd = 0.5))
target <- c(rep(1, 200), rep(2, 300), rep(3, 300))
ds <- data.frame(cbind(X1, Y1, target))
```

```
n.rows <- length(ds[,1])                                          # Number of rows
set.seed(seed=6218945)
random_sample <- sample(1:n.rows, ceiling(n.rows/2))  # Split dataset randomly in train/test subsets
ds_train <- ds[random_sample,]                          # Training set
ds_test <-  ds[setdiff(1:n.rows, random_sample),]      # Test set
RODM_create_dbms_table(DB, "ds_train")   # Push the training table to the database
RODM_create_dbms_table(DB, "ds_test")    # Push the testing table to the database

svm <- RODM_create_svm_model(database = DB,    # Create ODM SVM classification model
                             data_table_name = "ds_train",
                             target_column_name = "target")

svm2 <- RODM_apply_model(database = DB,    # Predict test data
                         data_table_name = "ds_test",
                         model_name = "SVM_MODEL",
                         supplemental_cols = c("X1","Y1","TARGET"))

color.map <- c("blue", "green", "red")
color <- color.map[svm2$model.apply.results[, "TARGET"]]
plot(svm2$model.apply.results[, "X1"],
     svm2$model.apply.results[, "Y1"],
     pch=20, col=color, ylim=c(-2,10), xlab="X1", ylab="Y1",
     main="Test Set")
actual <- svm2$model.apply.results[, "TARGET"]
predicted <- svm2$model.apply.results[, "PREDICTION"]
for (i in 1:length(ds_test[,1])) {
   if (actual[i] != predicted[i])
    points(x=svm2$model.apply.results[i, "X1"],
           y=svm2$model.apply.results[i, "Y1"],
           col = "black", pch=20)
}
legend(6, 1.5, legend=c("Class 1 (correct)", "Class 2 (correct)", "Class 3 (correct)", "Error"),
      pch = rep(20, 4), col = c(color.map, "black"), pt.bg = c(color.map, "black"), cex = 1.20,
       pt.cex=1.5, bty="n")

RODM_drop_model(DB, "SVM_MODEL")       # Drop the model
RODM_drop_dbms_table(DB, "ds_train")   # Drop the database table
RODM_drop_dbms_table(DB, "ds_test")    # Drop the database table

## End(Not run)

### SVM Classification

# Predicting survival in the sinking of the Titanic based on pasenger's sex, age, class, etc.
## Not run:
data(titanic3, package="PASWR")                                    # Load survival data from Titanic
ds <- titanic3[,c("pclass", "survived", "sex", "age", "fare", "embarked")]  # Select subset of attributes
ds[,"survived"] <- ifelse(ds[,"survived"] == 1, "Yes", "No")            # Rename target values
n.rows <- length(ds[,1])                                                 # Number of rows
random_sample <- sample(1:n.rows, ceiling(n.rows/2))  # Split dataset randomly in train/test subsets
titanic_train <- ds[random_sample,]                          # Training set
titanic_test <-  ds[setdiff(1:n.rows, random_sample),]      # Test set
RODM_create_dbms_table(DB, "titanic_train")   # Push the training table to the database
```

```
RODM_create_dbms_table(DB, "titanic_test")    # Push the testing table to the database

svm <- RODM_create_svm_model(database = DB,    # Create ODM SVM classification model
                             data_table_name = "titanic_train",
                             target_column_name = "survived",
                             model_name = "SVM_MODEL",
                             mining_function = "classification")

svm2 <- RODM_apply_model(database = DB,    # Predict test data
                         data_table_name = "titanic_test",
                         model_name = "SVM_MODEL",
                         supplemental_cols = "survived")

print(svm2$model.apply.results[1:10,])                          # Print example of prediction results
actual <- svm2$model.apply.results[, "SURVIVED"]
predicted <- svm2$model.apply.results[, "PREDICTION"]
probs <- as.real(as.character(svm2$model.apply.results[, "'Yes'"]))
table(actual, predicted, dnn = c("Actual", "Predicted"))              # Confusion matrix
library(verification)
perf.auc <- roc.area(ifelse(actual == "Yes", 1, 0), probs)          # Compute ROC and plot
auc.roc <- signif(perf.auc$A, digits=3)
auc.roc.p <- signif(perf.auc$p.value, digits=3)
roc.plot(ifelse(actual == "Yes", 1, 0), probs, binormal=T, plot="both", xlab="False Positive Rate",
         ylab="True Postive Rate", main= "Titanic survival ODM SVM model ROC Curve")
text(0.7, 0.4, labels= paste("AUC ROC:", signif(perf.auc$A, digits=3)))
text(0.7, 0.3, labels= paste("p-value:", signif(perf.auc$p.value, digits=3)))

RODM_drop_model(DB, "SVM_MODEL")            # Drop the model
RODM_drop_dbms_table(DB, "titanic_train")   # Drop the database table
RODM_drop_dbms_table(DB, "titanic_test")    # Drop the database table

## End(Not run)

### SVM Regression

# Aproximating a one-dimensional non-linear function
## Not run:
X1 <- 10 * runif(500) - 5
Y1 <- X1*cos(X1) + 2*runif(500)
ds <- data.frame(cbind(X1, Y1))
RODM_create_dbms_table(DB, "ds")   # Push the training table to the database

svm <- RODM_create_svm_model(database = DB,    # Create ODM SVM regression model
                             data_table_name = "ds",
                             target_column_name = "Y1",
                             mining_function = "regression")

svm2 <- RODM_apply_model(database = DB,    # Predict training data
                         data_table_name = "ds",
                         model_name = "SVM_MODEL",
                         supplemental_cols = "X1")

plot(X1, Y1, pch=20, col="blue")
```

```
points(x=svm2$model.apply.results[, "X1"],
       svm2$model.apply.results[, "PREDICTION"], pch=20, col="red")
legend(-4, -1.5, legend = c("actual", "SVM regression"), pch = c(20, 20), col = c("blue", "red"),
               pt.bg =  c("blue", "red"), cex = 1.20, pt.cex=1.5, bty="n")

RODM_drop_model(DB, "SVM_MODEL")       # Drop the model
RODM_drop_dbms_table(DB, "ds")         # Drop the database table

## End(Not run)

### Anomaly detection
# Finding outliers in a 2D-dimensional discrete distribution of points
## Not run:
X1 <- c(rnorm(200, mean = 2, sd = 1), rnorm(300, mean = 8, sd = 2))
Y1 <- c(rnorm(200, mean = 2, sd = 1.5), rnorm(300, mean = 8, sd = 1.5))
ds <- data.frame(cbind(X1, Y1))
RODM_create_dbms_table(DB, "ds")   # Push the table to the database

svm <- RODM_create_svm_model(database = DB,    # Create ODM SVM anomaly detection model
                             data_table_name = "ds",
                             target_column_name = NULL,
                             model_name = "SVM_MODEL",
                             mining_function = "anomaly_detection")

svm2 <- RODM_apply_model(database = DB,    # Predict training data
                         data_table_name = "ds",
                         model_name = "SVM_MODEL",
                         supplemental_cols = c("X1","Y1"))

plot(X1, Y1, pch=20, col="white")
col <- ifelse(svm2$model.apply.results[, "PREDICTION"] == 1, "green", "red")
for (i in 1:500) points(x=svm2$model.apply.results[i, "X1"],
                        y=svm2$model.apply.results[i, "Y1"],
                        col = col[i], pch=20)
legend(8, 2, legend = c("typical", "anomaly"), pch = c(20, 20), col = c("green", "red"),
               pt.bg =  c("green", "red"), cex = 1.20, pt.cex=1.5, bty="n")

RODM_drop_model(DB, "SVM_MODEL")       # Drop the model
RODM_drop_dbms_table(DB, "ds")         # Drop the database table

RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_drop_dbms_table *Drops a table in the Oracle Database*

Description

This function drops a table in the Oracle Database.

Usage

```
RODM_drop_dbms_table(database, data_table_name)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
data_table_name	Oracle database table to be dropped.

Details

This function drops a table in the Oracle Database.

Value

None.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

See Also

RODM_create_dbms_table

Examples

```
## Not run:
data(titanic3, package="PASWR")
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid="rodm", pwd="rodm")
RODM_create_dbms_table(DB, "titanic3")

# Drop the titanic3 database table
RODM_drop_dbms_table(DB, "titanic3")

RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_drop_model *Drop an Oracle Data Mining model*

Description

This function drops an Oracle Data Mining model in the Oracle database.

Usage

```
RODM_drop_model(database,
                model_name,
                sql.log.file = NULL)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
model_name	ODM Model name to be dropped.
sql.log.file	File to append the log of all the SQL calls made by this function.

Details

This function drops an Oracle Data Mining model in the Oracle database.

Value

None.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

See Also

RODM_create_svm_model, RODM_create_kmeans_model, RODM_create_oc_model, RODM_create_ai_model, RODM_create_nb_model, RODM_create_glm_model, RODM_create_assoc_model, RODM_create_dt_model

Examples

```
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid="rodm", pwd="rodm")
data(titanic3, package="PASWR")
db_titanic <- titanic3[,c("pclass", "survived", "sex", "age", "fare", "embarked")]
db_titanic[,"survived"] <- ifelse(db_titanic[,"survived"] == 1, "Yes", "No")
RODM_create_dbms_table(DB, "db_titanic")   # Push the table to the database
ai <- RODM_create_ai_model(
   database = DB,                      # Database ODBC connection
   data_table_name = "db_titanic",     # Database table containing the input dataset
   target_column_name = "survived",    # Target column name in data_table_name
   model_name = "TITANIC_AI_MODEL")    # Oracle Data Mining model name to create

# Drop the model that was just created
RODM_drop_model(DB, "TITANIC_AI_MODEL")

RODM_drop_dbms_table(DB, "db_titanic")     # Drop the database table
RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_list_dbms_models *List Oracle Data Mining models*

Description

This function list all of the Oracle Data Mining models in the user's schema in the Oracle database.

Usage

```
RODM_list_dbms_models(database)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection

Details

This function list all of the Oracle Data Mining models in the user's schema in the database. For each model, this function returns the model name, mining function (type of operation), algorithm, date the model was created, time it took to build the model (in seconds), size of the model (in megabytes), and comments associated with the model (if any).

Value

List of the following information:

MODEL_NAME	Name of the model.

MINING_FUNCTION	Mining function used when building the model.
ALGORITHM	Algorithm used when building the model.
CREATION_DATE	Date the model was created.
BUILD_DURATION	Duration to build the model in seconds.
MODEL_SIZE	Size of the model in MB.
COMMENTS	Comments associated with the model, if any.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Concepts 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/toc.htm

Oracle Data Mining Application Developer's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28131/toc.htm

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Examples

```
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid= "rodm", pwd = "rodm")

data(titanic3, package="PASWR")                                    # Load survival data from Titanic
ds <- titanic3[,c("pclass", "survived", "sex", "age", "fare", "embarked")]  # Select subset of attributes
ds[,"survived"] <- ifelse(ds[,"survived"] == 1, "Yes", "No")             # Rename target values
n.rows <- length(ds[,1])                                                  # Number of rows
set.seed(seed=6218945)
random_sample <- sample(1:n.rows, ceiling(n.rows/2))  # Split dataset randomly in train/test subsets
titanic_train <- ds[random_sample,]                          # Training set
titanic_test <-  ds[setdiff(1:n.rows, random_sample),]      # Test set
RODM_create_dbms_table(DB, "titanic_train")   # Push the training table to the database
RODM_create_dbms_table(DB, "titanic_test")    # Push the testing table to the database

# Create an ODM Naive Bayes model
nb <- RODM_create_nb_model(
   database = DB,                     # Database ODBC channel identifier
   model_name = "titanic_nb_model",   # ODM model name
  data_table_name = "titanic_train", # (in quotes) Data frame or database table containing the input dataset
   target_column_name = "survived")   # Target column name in data_table_name

# Create an ODM Attribute Importance model
```

```
ai <- RODM_create_ai_model(
   database = DB,                      # Database ODBC channel identifier
   model_name = "titanic_ai_model",    # ODM model name
  data_table_name = "titanic_train", # (in quotes) Data frame or database table containing the input dataset
   target_column_name = "survived")   # Target column name in data_table_name

# List the models
mlist <- RODM_list_dbms_models(DB)
mlist

RODM_drop_model(DB, "titanic_nb_model")
RODM_drop_model(DB, "titanic_ai_model")
RODM_drop_dbms_table(DB, "titanic_train")
RODM_drop_dbms_table(DB, "titanic_test")

RODM_close_dbms_connection(DB)

## End(Not run)
```

`RODM_open_dbms_connection`

Open a connection to an Oracle Database

Description

This function opens a connection to an Oracle Database.

Usage

```
RODM_open_dbms_connection(
   dsn,
   uid = "",
   pwd = "")
```

Arguments

`dsn`	ODBC Data Source Name.
`uid`	Database user id.
`pwd`	Password for the database user.

Details

This functions opens an ODBC channel to an Oracle database by calling the RODBC function: odbcConnect(dsn=dsn, uid=uid, pwd=pwd, case="oracle").

It validates that the database is Oracle version 11, that Oracle Data Mining is installed, and that the connecting user has appropriate privileges for mining in the database.

The necessary privileges for connecting and mining are as follows: CREATE SESSION, CREATE TABLE, CREATE VIEW, and CREATE MINING MODEL. The connecting user will also need quota in a tablespace.

The first time RODM_open_dbms_connection is invoked for a given database user, this function will also create a temporary table to hold settings that are used to build ODM models (table name RODM_SETTINGS_TABLE).

Value

An Oracle database ODBC channel.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Data Mining Administrator's Guide 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28130/toc.htm

See Also

RODM_close_dbms_connection

Examples

```
# Given a database user rodm, establish a connection to the orcl11g
# database.
# The database user would need privileges as described above, and could
# have been created in a fashion similar to:
# grant create session, create table, create view, create mining model,
#       unlimited tablespace to rodm identified by rodm;
## Not run:
DB <- RODM_open_dbms_connection(dsn="orcl11g", uid="rodm", pwd="rodm")

# Close the connection to the database.
RODM_close_dbms_connection(DB)

## End(Not run)
```

RODM_store_settings *Store Mining Model settings*

Description

This function stores model build settings in the the database table RODM_SETTINGS_TABLE.

Usage

```
RODM_store_settings(database,
                    rodmset,
                    auto_data_prep,
                    sql.log.file = NULL,
                    bias_frame = NULL,
                    bias_weights = FALSE)
```

Arguments

database	Database ODBC channel identifier returned from a call to RODM_open_dbms_connection
rodmset	Data frame containing the settings to be used for building a model.
auto_data_prep	Whether or not ODM should invoke automatic data preparation for the build.
sql.log.file	File to append the log of all the SQL calls made by this function.
bias_frame	Data frame containint bias information (priors/costs/weights).
bias_weights	Priors or Weights for 2-column bias frame.

Details

This is a generic function for persisting settings that drive model build.

It is not intended to be called directly by the user, but rather via the RODM_create_XX_model functions.

Value

None.

Author(s)

Pablo Tamayo <pablo.tamayo@oracle.com>

Ari Mozes <ari.mozes@oracle.com>

References

Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1) http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419/d_datmin.htm#ARPLS192

Index