Lab-5

May 25, 2021

Hands-on Lab: Perform K-Means Clustering Analysis with R

0.0.1 Welcome!

In this hands-on lab, we will perform the following tasks to illustrate the use of in-database anayltics in Db2 on Cloud using ibmdbR.

Tasks

Pre-requisites

Load the ibmdbR package

Connect to your Db2 on Cloud database

Create a table with data

Analyze the data

Dis-connect

Estimated Time Needed: 30 min

a. Pre-requisites

In this lab we will use Jupyter Notebooks within SN Labs to access data in a Db2 on Cloud database using RODBC. Information about Jupyter notebooks, SN Labs, and Db2 services is provided in the previous labs.

b. Load the ibmdbR package
   Let's start by loading the ibmdbR package. ibmdbR utilizes RODBC to connect to interact with the database so ensure that the database is accessable using RODBC. Let's load the ibmdbR package by clicking on the following cell and executing it (Shift+Enter):

[ ]:
```
library(ibmdbR);
```

c. Connect to your Db2 on Cloud database

Enter the other connection details for your instance of Db2 on Cloud:

[ ]:
```

```

Click here to view/hide hint

```
# Fill in the ...
driver.name <- "..."    # this matches entry in odbc.ini for DB2 driver
db.name <- "..."
host.name <- "host..."
port <- "..."
protocol="..."
user.name <- "..."
pwd <- "..."
dsnstr <- paste(driver....,
                  ";Data...=",....name,
                  ";Host...=",....name,
                  ";Port...=",...,
                  ";PROTOCOL=",..., sep="...")
```

Click here to view/hide solution

```
driver.name <- "DB2"    # this matches entry in odbc.ini for DB2 driver
db.name <- "BLUDB"      # e.g. BLUDB
host.name <- "hostname" # e.g.  "dashdb-txn-sbox-yp-dal09-11.services.dal.bluemix.net"
port <- "50000"         # e.g.  50000
protocol="TCPIP"        # i.e. TCPIP
user.name <- "username" # e.g.  "zjh17769"
pwd <- "password"       # e.g. "zcwd4+8gbq9bm5k4"
dsnstr <- paste(driver.name,
                  ";Database=",db.name,
                  ";Hostname=",host.name,
                  ";Port=",port,
                  ";PROTOCOL=",protocol, sep="")
```

Now create a connection string and connect to the database using idaConnect(). Syntax and arguments for this function follow:

Usage idaConnect(dsn, uid = ”“, pwd =”“, conType =”odbc”)

Arguments dsn - The DSN of the data base. uid - The user name. pwd - The password. conType - The connection type.

Add code to connect to the database (example: con <- idaConnect(…))

[ ]:

Click here to view/hide hint

```
# Fill in the ...
con <- ida...(dsn..., uid=....name, p...=...)
```

Click here to view/hide solution

```
con <- idaConnect(dsnstr, uid=user.name, pwd=pwd)
```

Now let’s initialize the In Database Analytics functions using idaInit()

Usage idaInit(con,jobDescription=NULL) Arguments con - An open RODBC connection. jobDescription - Optional argument that allows to assign a description to the jobs submitted from the R session

[ ]:

Click here to view/hide hint

```
# Fill in the ...
ida...(...)
```

Click here to view/hide solution

```
idaInit(con)
```

Let's test to check our connection was established successfully by using idaShowTables() to get a list of tables.

Tip: The list might be long … single or double click on the left side just below Out [linenumber] to minimize the output result

[ ]:

Click here to view/hide hint

```
# Fill in the ...
ida...(show...=...)
```

Click here to view/hide Solution

```
idaShowTables(showAll=TRUE)
```

d. Create table with data
   Our R environment contains a sample dataset called mtcars that contains certain attributes (like mileage, cylinders, horsepower, etc.) about 32 automobiles. If you are not familiar with this dataset, lets first take a look at it.

Load the mtcars library.

[ ]:

Click here to view/hide hint

```
# Fill in the ...
mt...
```

Click here to view/hide Solution

```
mtcars
```

We will load this data into a table in our database. Note in the output above that mpg, cyl, etc. are column names but the that the car name doesn't have a column label. That's because it isn't actually a real column in the mtcars dataframe, rather its a special column that indictes the rownames.

Now, use the as.ida.data.frame() function to create a new table in the database that inputs the data from the newmtcars dataframe. This function creates an IDA data frame from a local R dataframe by creating a table in the database.

Usage as.ida.data.frame(x, table=NULL, clear.existing=FALSE, case.sensitive=TRUE, rownames=NULL, dbname=NULL, asAOT=FALSE) Arguments x - The name of the input object. table - The name of the table to be created. clear.existing - Indicates whether the existing table is to be dropped (TRUE). case.sensitive - Specifies whether the column names to be treated as case-sensitive (TRUE). rownames - The name of the column for the unique row id. If NULL, not added. dbname, asAOT - Parameters for DB2 on z/OS

Replace the code below with functional parameters.

[ ]:

Click here to view/hide hint

```
# Fill in the ...
tableName <- "..."
idf <- as.ida.dataframe(...)
idf
```

Click here to view/hide solution

```
tableName <- "MTCARS"
as.ida.data.frame(mtcars, table=tableName, clear.existing=TRUE, rownames="carname")
```

Now let's use idaQuery() to check if a table was created in the database with the mtcars data. This function can run any SQL query on the database and put the results into a dataframe.

Usage idaQuery(..., as.is=TRUE, na.strings = "NA") Arguments as.is - Specifies whether the result columns are to be converted using RODBC type conversions (as.is=FALSE) or left unconverted (as.is=TRUE). na.strings - character vector of strings to be mapped to NA when reading character data.

Replace the … with your query to fetch the contents of the MTCARS table.

[ ]:

Click here to view/hide hint

```
# Fill in the ...
mtcarsdb <- idaQuery(...)
mtcarsdb
```

Click here to view/hide solution

```
mtcarsdb <- idaQuery("SELECT * FROM ",tableName)
mtcarsdb
```

e. Analyze the data
As you have seen above you can use ibmdbR to put data into the database and run queries against the database. The key benefit of ibmdbR, however, lies in being able to perform in-database analytics. That is, "pushing down many basic and complex R operations into the

database, which removes the main memory boundary of R and allows you to make full use of parallel processing in the underlying database".

Let's first create a dataframe referencing the MTCARS table in the database. NOTE: We could use the 'idf' dataframe that we created above to store the mtcars data in the database but this shows another way.

[ ]:

Click here to view/hide hint

```
# Fill in the ...
id... <- ida.data....(...)
```

Click here to view/hide solution

```
idf2 <- ida.data.frame(tableName)
```

Note that an in database dataframe like idf2 is just a pointer to the object(s) in database. Therefore the following will not display the data.

[ ]:

Click here to view/hide hint

```
# Fill in the ...
i...2
```

Click here to view/hide solution

```
idf2
```

If you want to see the data referenced by the dataframe you can use a function like head() to examine the first few rows:

[ ]:

Click here to view/hide hint

```
# Fill in the ...
head(...)
```

Click here to view/hide solution

```
head(idf2)
```

Now let's use one of the built-in analytic functions such as idaKMeans() on this in-database dataframe created above on MTCARS table. Just like kmeans(), idaKMeans() is used for analyzing clustering on the data.

PS: Although we will utilize the sample mtcars dataset in the database, appreciate the advantages of in-database analytics if this was a very large dataset.

idaKMeans This function generates a k-means clustering model based on the contents of a IDA data frame. Usage idaKMeans(data, id, k=3, maxiter=5, distance="euclidean", outtable=NULL, randseed=12345, statistics=NULL, modelname=NULL) Arguments data - An IDA data frame

that contains the input data for the function. id - The name of the column that contains a unique ID for each row of the input data. k - The number of clusters to be calculated. maxiter - The maximum number of iterations to be used to calculate the k-means clusters. distance - The distance function that is to be used. e.g.: “euclidean” or “norm_euclidean” outtable - The name of the output table that is to contain the results of the operation. randseed - The seed for the random number generator. statistics - Denotes which statistics to calculate. e.g. “none”, “columns” and “all” or NULL modelname - The name under which the model is stored in the database. This is the name that is specified when using functions such as idaRetrieveModel or idaDropModel. object - An object of the class idaKMeans to be used for prediction, i.e. for applying it to new data. x - An object of the class idaKMeans to be printed. newdata - A IDA data frame that contains the data to which to apply the model. … - Additional parameters to pass to the print or predict method.

Now run K-Means with k=4 clusters on the mtcars dataset in the database; replace the … with your own parameters and then print the model.

[ ]:

Click here to view/hide hint

```
# Fill in the ...
#now run kmeans with k=4 clusters on the mtcars dataset in the database
kclusters=...
#replace ... with your parameters
km <- idaKMeans (idf2, ...)
print(...) #Print ...
```

Click here to view/hide solution

```
# drop this model in case it already exists e.g. re-running the cell
idaDropModel("KMEANSMODEL")
kclusters=4
#Create a kmeans model stored in the database as KMEANSMODEL:
km <- idaKMeans(idf2, id="carname", k=kclusters, modelname="KMEANSMODEL")
print(km) #Print the model
```

Predict the model. Replace the … with the code to predict the model.

[ ]:

Click here to view/hide hint

```
# Fill in the ...
pred <- pred...(km,...2,id="carname")
....(pred,...)
```

Click here to view/hide solution

```
pred <- predict(km,idf2,id="carname")
head(pred,10)
```

Now let’s visualize the clustering analysis using ggplot2. You need to fix the code as indicated.

[ ]:

Click here to view/hide hint

```
# Fill in the ...
#FIX THIS CELL
#Not working with idf2 and km
#data(idf2)
library(gg...2)
x <- t...(idf2$...,km$cluster,...)
y <- t...(idf2$...,km$cluster,...)
kcenters <- data....(x,...)
ggplot(id...,aes(...,hp))+geom_...(col=km$...,size=...) +
    geom_point(...=kcenters,aes(...,...),pch=...,size=...,colour=...:k...)
```

Click here to view/hide solution

```
#FIX THIS CELL
#Not working with idf2 and km
#data(idf2)
library(ggplot2)
x <- tapply(idf2$mpg,km$cluster,mean)
y <- tapply(idf2$hp,km$cluster,mean)
kcenters <- data.frame(x,y)
ggplot(idf2,aes(mpg,hp))+geom_point(col=km$cluster,size=4) +
    geom_point(data=kcenters,aes(x,y),pch=8,size=10,colour=1:kclusters)
```

f. Dis-connect

Add the code to disconnect from the databse.

[ ]:

Click here to view/hide hint

```
# Fill in the ...
ida...(...)
```

Click here to view/hide solution

```
idaClose(con)
```

Summary

In this lab you performed K-Means clustering analysis on a sample dataset using the ibmdbR library with an R notebook in Jupyter.

Thank-you for completing this lab on K-Means Clustering.

0.1 Authors

- Rav Ahuja
- Agatha Colangelo
- Sandip Saha Joy

0.2 Changelog

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-01-22	2.0	Sandip Saha Joy	Created revised version of the lab
2017	1.0	Rav Ahuja & Agatha Colangelo	Created initial version of the lab