Hannah Roach
2/13/2019

Lab Exercise 4: K-means Clustering

Purpose:	This lab is designed to investigate and practice K-means Clustering. After completing the tasks in this lab you should able to: • Use R functions to create K-means Clustering models • Use ODBC connection to the database and execute SQL statements and load datasets from the database in an R environment • Visualize the effectiveness of the K-means Clustering algorithm using graphic capabilities in R • Use MADlib functions for K-means clustering

Tasks:	Tasks you will complete in this lab include: • Use the R –Studio environment to code K-means Clustering models • Use the ODBC connection in the R environment to create the average household income from the census database as test data for K-means Clustering • Use R graphics functions to visualize the effectiveness of the K-means Clustering algorithm • Use MADlib functions for K-means clustering

References:	References used in this lab are located in your ***Student Resource Guide Appendix.*** http://www.statmethods.net/advstats/cluster.html (originally from Everitt & Hothorn).

Workflow overview

1. Set the Working Directory
2. Establish the ODBC Connection
3. Open Connections to ODBC Database
4. Get Data from the Database
5. Read in the Data for Modeling
6. Execute the Model
7. Review the Output
8. Plot the Results
9. Find the Appropriate Number of Clusters
10. Close Connections to ODBC Database
11. Perform K-means Clustering Using In-database Analytics Method

Lab Instructions

<table>
<tr><th>Ste p</th><th>Action</th></tr>
<tr><td>1</td><td>Log in with GPADMIN credentials onto R-Studio.</td></tr>
<tr><td>2</td><td><u>Set the Working Directory:</u>

1. Set working directory to ~/LAB04/, execute the command:

<code>setwd("~/LAB04")</code>

• (Or use the “Tools” option in the tool bar in the RStudio environment.)</td></tr>
<tr><td>3</td><td><u>Establish the ODBC Connection:</u>

Load the RODBC package, type in:

<code>library('RODBC')</code></td></tr>
<tr><td>4</td><td><u>Open Connections to ODBC Database:</u>

1. Ensure the username(uid) and password (pwd) are provided correctly in the following command:

<code>ch <- odbcConnect("Greenplum",uid="gpadmin",</code>
<code>   case="postgresql",pwd="changeme")</code></td></tr>
</table>

Ste p	Action
5	**<u>Get Data from the Database:</u>** 1. Before creating the table, "income_state", you must first delete the table, if it already exists. Type in: `sqlDrop(ch,"income_state")` 2. Use the sqlQuery command to create the table, "income_state" : `> sqlQuery(ch,` ` "CREATE TABLE income_state AS` ` SELECT` `   f.name AS state` ` , round(avg(h.hinc),0) AS income` ` FROM` `   housing AS h` ` JOIN` `   fips AS f` ` ON` `   h.state = f.code` ` WHERE` `   h.hinc > 0` ` GROUP BY` `   f.name` ` DISTRIBUTED BY (income);` `" )` **Note:** This code creates the table, "income_state", in database "training2". 3. Inspect this table using the following command: `sqlColumns(ch,"income_state")` 4. Review the output on the console. **Note:** The SQL Query is available for you to copy and paste into the working directory File name: mod4lab4.sql.

Ste p	Action
6	**<u>Read in the Data for Modeling</u>:** You need the data to be read in as a "matrix". 1. Execute the following statement to read in the database table "income_state". Use the sqlFetch command. The "rownames" attribute ensures the data is rendered as a matrix and the row names are taken from the column "state". `income <- as.matrix(sqlFetch(ch,"income_state",` `           rownames="state"))` `> summary(income)` 2. Review the results of "income" on the console window. 3. Ensure that in the "data" window the variable "income" is represented as a 52x1 integer matrix. ***<u>NOTE – Check in workspace windows to see this</u>!

Step	Action
7	**Execute the Model:** 1. Sort the data “income” before the modeling process. This will make it easier to understand the results and in visualizing. `income <- sort(income)` The K-Means function, provided by the *cluster* package, is used as follows: `kmeans(x, centers, iter.max = 10, nstart = 1, algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"))` where the arguments are: • **x:** A numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns). • **centers:** Either the number of clusters or a set of initial (distinct) cluster centers. If a number, a random set of (distinct) rows in x is chosen as the initial centers. • **iter.max:** The maximum number of iterations allowed. • **nstart:** If *centers* is a number, *nstart* gives the number of random sets that should be chosen. • **algorithm:** The algorithm to be used. It should be one of the following "Hartigan-Wong", "Lloyd", "Forgy" or "MacQueen". If no algorithm is specified, the algorithm of Hartigan and Wong is used by default. 2. Cluster the data into 3 groups (centers = 3) and also specify the number of random sets to start with as, 15. `> # Fit the k-means cluster with 3 initial cluster centers` `> km <- kmeans (income,3,15)`

Step	Action
8	<u>**Review the Output:**</u>

1. Use the following command to display the fitted model on the console:

```
> km
```

```
> km
K-means clustering with 3 clusters of sizes 6, 25, 21

Cluster means:
   [,1]
1 69989
2 56454
3 44297

Clustering vector:
 [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2
[30] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1

Within cluster sum of squares by cluster:
[1] 9.24e+07 3.45e+08 5.17e+08
 (between_SS / total_SS =  79.1 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"
>
```

2. What are the cluster means? 69989, 56454, 44297
3. What are the available components in the model? “clusters”, “centers”, “totss”, “withinss”, “tot.withinss”, “betweenss”, “size”
4. How many data points cluster into each group? 6, 25, 21

The output from the model provides the following:

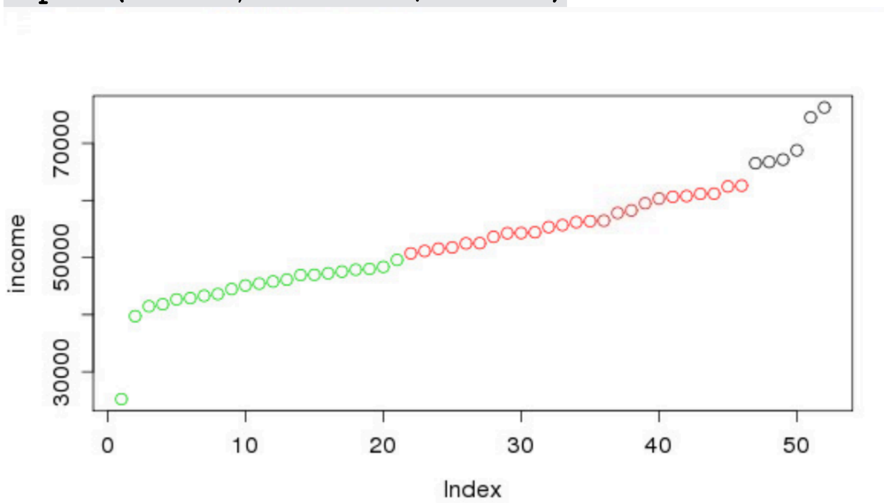
- **cluster** A vector of integers (from 1:k) indicating the cluster to which each point is allocated.
- **centers** A matrix of cluster centers.
- **withinss** The within-cluster sum of squares for each cluster.
- **totss** The total within-cluster sum of squares.
- **tot.withinss** Total within-cluster sum of squares, that is, sum(withinss).
- **betweenss** The between-cluster sum of squares.
- **size** The number of points in each cluster.

<table>
<tr><th>Ste p</th><th>Action</th></tr>
<tr><td>9</td><td>

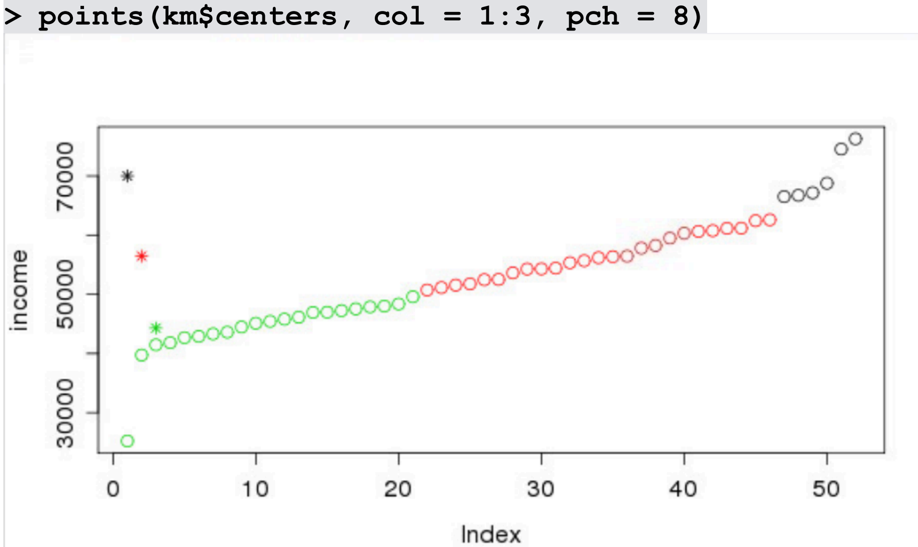
<u>Plot the Results:</u>

1. Now plot the results using the following commands:

```
> # plot clusters
> plot(income, col = km$cluster)
```




```
> # plot centers
> points(km$centers, col = 1:3, pch = 8)
```



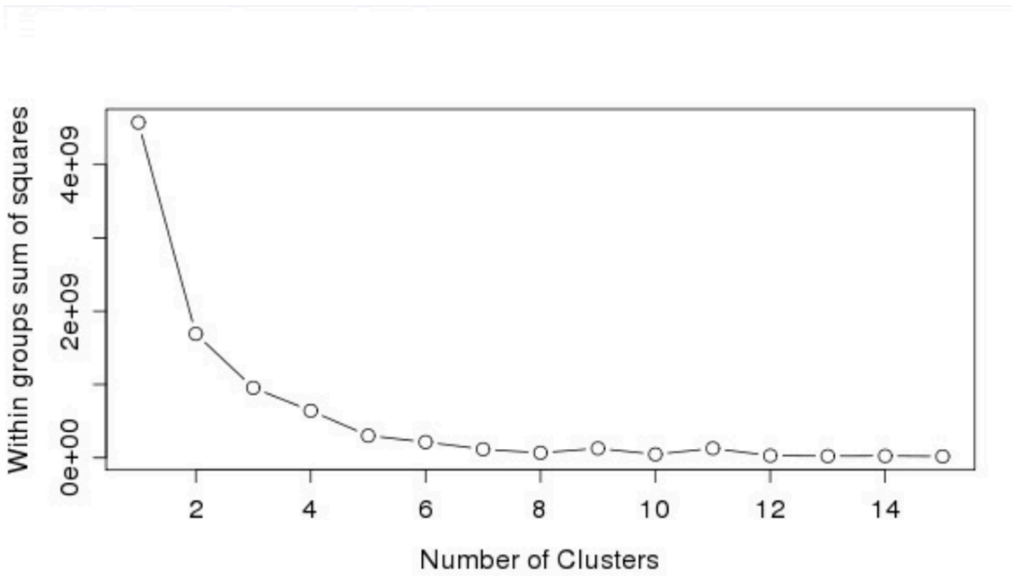
2. Review the output on the graphic window.

</td></tr>
</table>

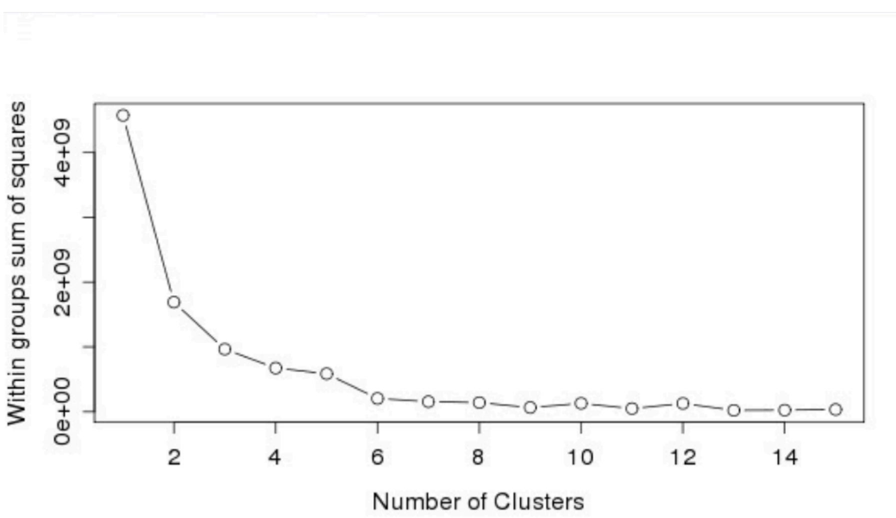
10	**Find the Appropriate Number of Clusters:**

1. Plot the within-group-sum of squares and look for an "elbow" of the plot. The elbow (if you can find one) tells you what the appropriate number of clusters probably is. (Adapted from http://www.statmethods.net/advstats/cluster.html (originally from Everitt & Hothorn)).

```
wss <- numeric(15)
> for (i in 1:15) wss[i] <- sum(kmeans(income,
      centers=i)$withinss)
> plot(1:15, wss, type="b", xlab="Number of Clusters",
   ylab="Within groups sum of squares")
```



2. Review the output on the graphic window. Is there an elbow to the plot? Yes
3. Repeat the modeling with a few values around the elbow (or 4 and 5 centers if there is no elbow) and review the results.



Ste p	Action
11	**<u>Close Connections to ODBC Database:</u>** Use the following command: `odbcClose(ch)` The R Code for this exercise is available at /home/gpadmin/LAB04/kmeans1.R

12

ACTION

Perform K-means Clustering Using In-database Analytics Method:

1. **Open putty and connect to your BE IP**
2. **Navigate to cd LAB04**
3. **Run the sql command:** `psql -d training2 -f kmeansmadlib.sql`

Below is what the command is doing

```
DROP TABLE IF EXISTS myschema.data;
CREATE  TABLE myschema.data (
  pid INT
  , position FLOAT8[])
DISTRIBUTED BY (pid);

INSERT INTO myschema.data (pid,position[1])
SELECT
  h.state
, round(avg(h.hinc),0)
FROM
  housing AS h
WHERE
  h.hinc > 0
GROUP BY
  h.state
;

SET SEARCH_PATH to madlib,public,myschema;

SELECT madlib.kmeans_random('myschema.data', 'position',
null, 'km_p', 'km_c','l2norm', 15,0.001,True, True, 6);

SELECT * FROM madlib.km_c;
SELECT * FROM madlib.km_p;
```

The first part of the code is similar to the one created in step 5 of this lab. The K-means function is called by:

using *random* centroid seeding method for a provided k :

```
SELECT * FROM kmeans_random(
  'src_relation', 'src_col_data', 'src_col_id',
  'out_points', 'out_centroids',
  'dist_metric',
  max_iter, conv_threshold,
  evaluate, verbose,
```

	`` k); ``` The centroid locations are stored in kmeans_out_centroids_(*run_id*): In the example above input_table is myshema.data, number of clusters is 6 **4. The output will be of km_c and km_p from schema myschema.data** *****provide screenshot** **This is similar to what you had dome in step 7.**

End of Lab Exercise