

SQLDM – implementing k-means clustering using SQL

Jay B.Simha
Abiba Systems
Bengaluru – 560 080
jay.b.simha @abibasystems.com

ABSTRACT

Clustering is one of the important data mining techniques used in exploratory analysis. In particular k-means algorithm is a popular algorithm for clustering. The algorithms available in statistical, machine learning and database literature have limitations of scalability. In this paper an attempt has been made to implement the k-means algorithm using SQL in a database. This approach is guaranteed to provide scalability on large datasets. Further in-database analysis makes the data access and pre-processing easier. The preliminary results are encouraging. The SQLDM algorithm is showing linear scalability in both number of records and number of attributes. Work is under progress to improve the algorithm with additional features.

Introduction:

Data mining techniques can significantly boost the ability to analyze data. Despite the potential effectiveness of data mining to significantly enhance data analysis, this technology is destined to be a niche technology unless the data can be effectively accessed from traditional database systems, which may require integration with DBMS. This is because data analysis needs to be consolidated at the warehouse for data integrity and management concerns. Hence, one of the key challenges is to enable integration of data mining technology seamlessly within the framework of traditional database systems [2].

In the last decade, significant research progress has been made towards streamlining data mining algorithms. There has been an explosion of work [1] in scaling many major data mining techniques to work with large data sets, i.e., ensuring that the algorithms are “disk-aware” and more generally, conscious of memory hierarchy, instead of making the assumption that all data must reside in memory. Another direction of work that has been pursued is to consider if data mining algorithms may be implemented as traditional database applications. Efforts to implement mining algorithms on top of database systems have also led to primitives such as sampling to ease the task of data mining on relational systems [3]. Some extensions allow the data mined model to be accessed by SQL like operators [6].

However, very little work is done in using existing DBMS capabilities and SQL programming language to implement data mining algorithms [7]. In this paper we propose a simple implementation of k-means clustering algorithm using SQL. Commonly available DDL and DML commands are used to build the system. The resulting system is elegant (in terms of amount of code) and scalable (limited by the underlying RDBMS and its tuning).

Clustering and k-means:

Cluster analysis is a technique for grouping data and finding structures in data. The most common application of clustering methods is to partition a data set into clusters or classes, where similar data are assigned to the same cluster whereas dissimilar data should belong to different clusters [4].

K-means [5] is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one

for each cluster. These centroids should be placed in a clever way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early grouping is done. At this point k new centroids are calculated as centers of the clusters resulting from the previous step. After having these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop the k centroids change their location step by step until no more changes are done. In other words centroids do not move any more. Finally, this algorithm aims at minimizing an *objective function*, in this case a squared error function. The objective function

$$J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2,$$

where $\|x_i^{(j)} - c_j\|^2$ is a chosen distance measure between a data point $x_i^{(j)}$ and the cluster centre c_j , i is an indicator of the distance of the n data points from their respective cluster centers.

The algorithm is composed of the following steps:

1. *Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.*
2. *Assign each object to the group that has the closest centroid.*
3. *When all objects have been assigned, recalculate the positions of the K centroids.*
4. *Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.*

Fig 1. K-means algorithm

Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centers. The k-means algorithm can be run multiple times to reduce this effect.

Implementing SQL K-means:

There are three reasons for implementing k-means algorithm in SQL. First, most of the implementations for k-means in a procedural language like C, C++ or Java. Using SQL makes writing data mining algorithm easier, since SQL is declarative and there is no need to handle data at the lowest level.

Second, these algorithms assume that all the data can fit into memory. This creates a problem, when the data to be explored is larger than the available memory, which in reality is always true. Even though some clever algorithms use sampling or incremental learning, the ability to handle large data is still a problem with these algorithms.

Third, building the cluster model is only one part of the problem. Accessing the data from the sources like a data warehouse and transforming them outside the database is another major problem. Using SQL this problem can be minimized, since the data will be analyzed within the database, without the need for additional overheads.

In this research an attempt has been made to implement k-means algorithm using SQL for in-database analysis of the large datasets. The strategy for implementing SQL k-means is as follows.

- Select the number of clusters randomly
 - Top K records
 - Filter with Order by
 - Filter with MOD operator
- Compute distance matrix
 - Join records and clusters table to compute distances
 - Select the one cluster per record
- Check the movement of the points
 - re compute new cluster centers
 - check the difference between old and new centers
- Repeat the steps 2 and 3 till the difference is less than threshold

The different data structures (tables) used in implementing the SQL k-means are given in table 1 and the process flow is shown in figure 2.

Table 1. Data structures used for SQL k-means

Table	Primary Key	Columns	Number of rows	Contents
Y	RID	y1, y2, ..., yp	N	Records
C	CID	y1, y2, ..., yp	K	Cluster centers
YD	RID	d1, d2, ..., dp	KN	Distances
MD	RID	D	N	Minimum distance
C2	None	cid, rid	N	Classified records
CN	CID	y1, y2, ..., yp	K	New cluster centers

The process starts with selection of cluster centers from the data table (Y) into table C. Next the distance of each of the data points (records) in data table (Y) with each of the cluster center from table C, are inserted into the distances table (YD). Next the records are assigned to the clusters based on minimum distance and attached with a class label of the cluster they belong to in table MD. Subsequently the new cluster centers are computed by joining tables C2 and Y and inserted into table CN. If the CN and C are different (based on chosen criteria), then the process is repeated till convergence.

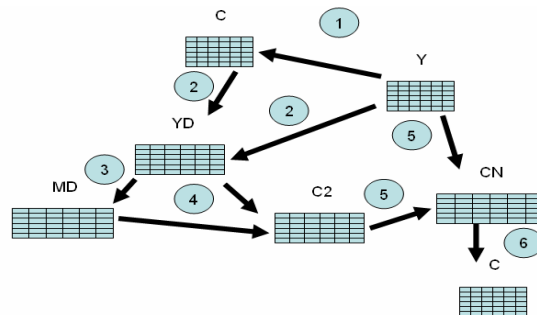


Fig. 2 SQL k-means process

The queries used for implementing the k-means are given in fig 3. The code shown is only for one attribute to demonstrate the compactness and readability of the code. However, the concept can be generalized to accommodate any number of attributes as restricted by the underlying RDBMS.

```

create table Y (rid int, y1 float); //populate data
create table C (cid int, c1 float); //Populate data
create table YD(rid int, cid int, dist float);
insert into YD (rid, cid, dist) (select Y.rid, C.cid, abs(C.c1-Y.y1) from Y,C);
create table MD (rid int, dist float);
insert into MD (rid, dist) (select rid, min(dist) from YD group by rid);
create table C2(rid int, cid float);
insert into C2 (rid, cid) (select YD.rid, min(YD.cid)
from YD, MD where MD.rid = YD.rid and MD.dist = YD.dist group by MD.rid);
create table CN (cid int, c1 float);
insert into CN (cid, c1) (select C2.cid, avg(Y.y1) from Y, C2 where Y.rid = C2.rid group by C2.cid);
select max(abs(C.c1-CN.c1)) from C, CN where C.cid = CN.cid;
delete from C;
insert into C (cid, c1) (select cid, c1 from CN);
delete C2;
delete from CN;
delete from YD;
delete from MD;

```

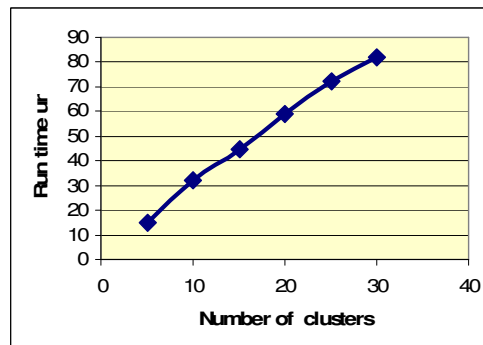
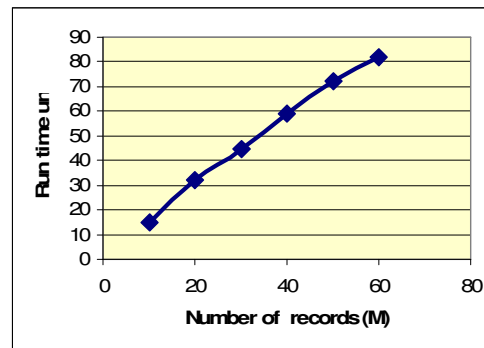
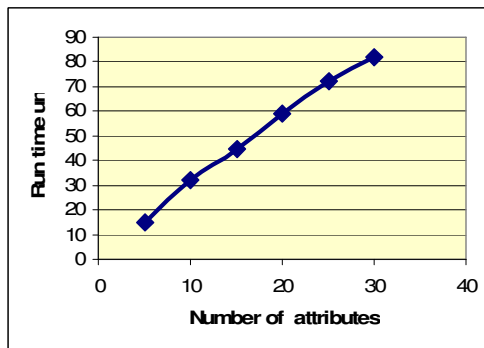
Fig 3. SQL queries for k-means

Experiments:

The experiments were conducted on a Linux based desktop system with 2 GB of RAM. The database used was MySQL 5.0. The SQL code generation was done in Jython and the connection to DBMS was done through JDBC library. Synthetic data used in the experimentation was generated using IBM data generator [8]. In the experimentation the number of attributes, number of records and number of cluster are varied to build the performance profile of the proposed algorithm.

Results and discussions:

The results of the experiments on the synthetic datasets using SQL k-means algorithm seems to be promising. It can be seen that SQL k-means scales linearly on all the three dimensions – number of attributes, number of records and number of clusters. However, the comparison of this implementation with memory based algorithms could not be carried out due to large data size, which could not fit into the memory.



Conclusions:

Clustering is one of the important functions in knowledge discovery and data mining. K-means is one of the most popular clustering algorithms used in data mining. In this research an attempt has been made to implement the k-means algorithm in SQL. The results are promising and the algorithm implemented in SQL scales linearly in all the dimensions tested. This is expected to provide scalability for large datasets and reduction in pre-processing overheads. The work is under progress to enhance the current implementation with additional features. The future research directions are – (i) to develop a method, implemented in SQL for optimum number of clusters, (ii) to evaluate the cluster quality and (iii) to test the concept in a live scenario.

References:

1. Agrawal R., et al.: Fast Discovery of Association Rules. *Advances in Knowledge Discovery and Data Mining* 1996: 307-328
2. Chaudhuri S.: Data Mining and Database Systems: Where is the Intersection? *Data Engineering Bulletin* 21(1) 1998.
3. Clear J., et al.: NonStop SQL/MX Primitives for Knowledge Discovery. *KDD 1999*: 425-429.
4. Hoppner, F., Klawonn F., Kruse, R., and Runkler, T., *Fuzzy Cluster Analysis*, John Wiley and Sons, 1999.
5. MacQueen J: "Some Methods for classification and Analysis of Multivariate Observations, *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*", Berkeley, University of California Press, 1:281-297, 1967
6. Netz A., et al., "Integration of Data Mining and Relational Databases" *Proceedings of the 26th International Conference on Very Large databases*, Cairo, Egypt, 2000
7. Ordonez C, "Programming the K-means clustering algorithm in SQL", *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, August 22-25, 2004, Seattle, WA, USA
8. http://www.cs.loyola.edu/~cgiannel/assoc_gen.html