BIRCH CLUSTERING ON BENCHMARK ARTIFICIAL DATASETS

Inala Vivek Vamsi
BITS Pilani
Hyderabad, India
f20160230@hyderabad.bits-pilani.ac.in

***Abstract*—**There are several clustering algorithms which can cluster various datasets based on primary mathematical norms of clustering. Such algorithms, like K Means, Agglomerative, DBSCAN, Divisive, EM using GMM to name a few, are found to be effective in terms of both space and time complexity. There are several types of clustering viz., partition, density based, hierarchical, model-based, etc.. Though there are several types of clustering methods, Hierarchical clustering takes an upper hand when we do not want to specify the number of clusters required or when we need to understand data in a better way using dendrograms. Though hierarchical clustering is very flexible with data, it is not scalable, i.e, it is not very effective when the data set fed is very large. This briefly states the motivation behind adding an extension to the existing hierarchical clustering algorithms. The aim of this assignment is to build an algorithm which is as an extension to existing hierarchical clustering algorithms like Agglomerative, Divisive, etc.. This extension thereby caters to minimize such efficiency (like scalability) flaws.

***Keywords*—**Data Mining, hierarchical clustering, Agglomerative, Birch, Cluster Frequency(CF), CF Tree

I. Introduction

Data mining is a process to discover knowledge from data which the stored in data base and data warehouses responsibility. In Data mining hierarchical clustering works by grouping data objects into a tree of cluster. Hierarchical clustering methods can be further classified into agglomerative and divisive hierarchical clustering. This classification depends on whether the hierarchical decomposition is formed in a bottom-up or top-down fashion. Hierarchical techniques produce a nested sequence of partitions, with a single, all inclusive cluster at the top and singleton clusters of individual objects at the bottom. Each intermediate level can be viewed as combining two clusters from the next lower level or splitting a cluster from the next higher level. The result of a hierarchical clustering algorithm can be graphically displayed as tree, called a dendrogram. This tree graphically displays the merging process and the intermediate clusters. This graphical structure shows how points can be merged into a single cluster. It can be broadly classified into two types as Agglomerative and Divisive:

A) Agglomerative Clustering

The agglomerative clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It's also known as AGNES (Agglomerative Nesting). Agglomerative clustering works in a "bottom-up" manner. The algorithm starts by treating each object as a singleton cluster. Next, pairs of clusters are successively merged until all clusters have been merged into one big cluster containing all objects. At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (nodes). The result is a tree-based representation of the objects, named dendrogram. It is good and works best when it comes to identifying small clusters.

B) Divisive Clustering

The inverse of agglomerative clustering is divisive clustering, which is also known as DIANA (Divise Analysis) and it works in a "top-down" manner. It begins with the root, in which all objects are included in a single cluster. At each step of iteration, the most heterogeneous cluster is divided into two. The process is iterated until all objects are in their own cluster. Initially, it is trained on the whole data set. During each iteration of division, the "poorest-fit" cluster whose HMM gives the lowest likelihood to the items in this cluster would be split. This process is repeated until a stop criterion is reached or all the clusters become singletons. It is good and works best when it comes to identifying large clusters.

Where is the necessity for BIRCH? (Problem Motivation)

The main purpose of describing these clustering algorithms was to minimize disk I/O operations and consequently reducing time complexity. But unfortunately Agglomerative clustering failed to address things like efficient data scan and time complexity for large databases.

II. BIRCH

To deal with these issues Tian Zhang proposed an agglomerative hierarchical clustering extension method named BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), and verified that it was especially suitable for large databases. BIRCH incrementally and dynamically clusters incoming multi-dimensional metric data points
so that best quality clusters can be produced with available resources. BIRCH can typically produce a good cluster with a single scan of the data, and improve the quality further with a few additional scans of the data. BIRCH was also the first clustering algorithm proposed in the database area that can handle noise effectively. Tian Zhang also evaluated BIRCH's time/space efficiency, data input order sensitivity, and cluster quality through several experiments before proposing it to the Data mining world.

BIRCH process begins by partitioning objects hierarchically using tree structure and then applies other clustering algorithms to refine the clusters. BIRCH process mainly takes four phases to produce best quality clusters. In this process two concepts are introduced, clustering feature and clustering feature tree (CF tree), which are used to summarize cluster representations. A CF tree is a height-balanced tree that stores the clustering features for a hierarchical clustering. Each node of this tree is composed of several Clustering features (CF). From Fig.1, we can see what the clustering feature tree looks like. Each node including leaf nodes has several CFs, and the CFs of internal nodes have pointers to child nodes, and all leaf nodes are linked by a doubly linked list.

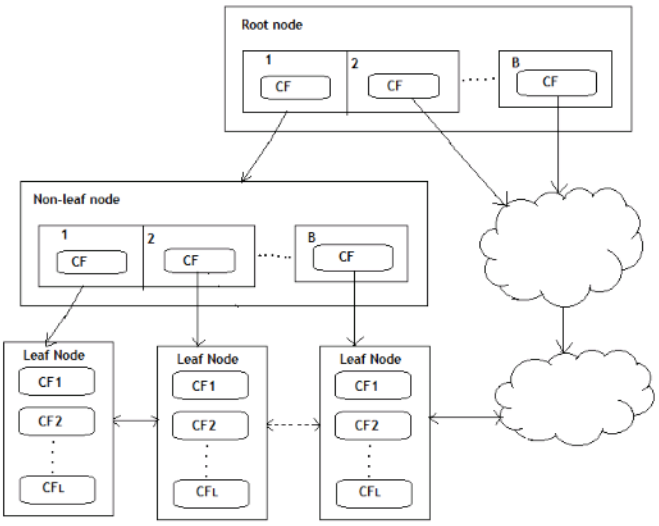

Fig. 1

Each CF is a triplet, which can be represented by (N, LS, SS).

- Where N represents the number of sample points in the CF, which is easy to understand
- LS represents the vector sum of the feature dimensions of the sample points in the CF
- SS represents the square of the feature dimensions of the sample points in the CF.

CF has a very good property. It satisfies the linear relationship, that is:

$$CF_1+CF_2 = (N_1+N_2, LS_1+LS_2, SS_1+SS_2)$$

This property is also well understood by definition. If you put this property on the CF Tree, that is to say, in the CF Tree, for each CF node in the parent node, its (N, LS, SS) triplet value is equal to the CF node pointed to the sum of the triples of all child nodes. As can be seen from the Fig.2, the value of the triplet of CF1 of the root node can be obtained by adding the values of the 6 child nodes (CF7-CF12) that it points to. In this way, we can be very efficient when updating the CF Tree.

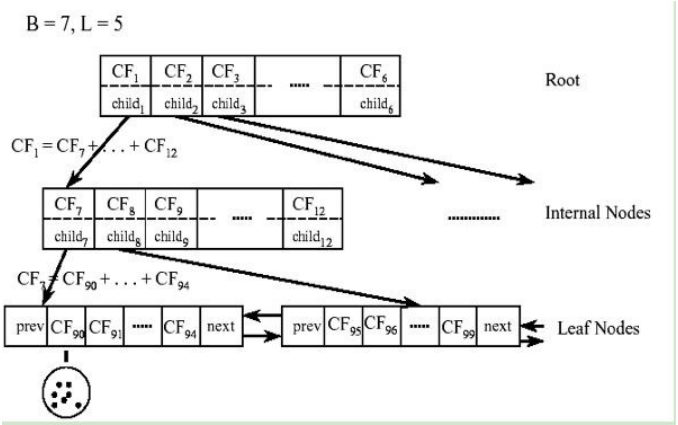

Fig. 2

Algorithm 1 BIRCH algorithm

Input: M data points
Output: N subclusters
repeat
Calculate the cluster feature $\vec{CF}$ of data points
Insert record to construct CF tree
until Insert all records into the CF tree

Fig. 3

To elaborate on what is done in the repeat section of Fig.3, at any stage of iteration of the algorithm as jotted in Fig.3, we need to do the following:

- Find the leaf node closest to the new sample and the closest CF node in the leaf node from the root node
- After the new sample is added, if the radius of the hyper-sphere corresponding to this CF node still satisfies the threshold T, then all the CF triplets on the path are updated, and the insertion ends. Otherwise, go to 3.
- If the number of CF nodes of the current leaf node is less than the threshold L, create a new CF node, put in a new sample and the new CF node into this leaf node, update all CF triplets on the path, and insertion Ends. Otherwise, go to 4
- Divide the current leaf node into two new leaf nodes, select the two CF tuples with the farthest hyper-sphere among all CF tuples in the old leaf node, and distribute as the first CF node of the two new leaf nodes. Put other tuples and new sample tuples into corresponding leaf nodes according to the principle of distance.
- In turn, check whether the parent node is also to be split. If it needs to be split in the same way as the leaf node.

After the completion of all these steps all the training set samples are built into a CF Tree, and a basic BIRCH algorithm is completed. The corresponding output is several CF nodes, and the sample point in each node is a cluster.

In the process till here we will have had achieved some key advantages like:

- Save memory, all samples are on disk, CF Tree only stores CF nodes and corresponding pointers.

- The clustering speed is fast, and it only takes one scan of the training set to build the CF Tree, and the addition, deletion, and modification of the CF Tree are very fast.

- Noise points can be identified, and preliminary classification preprocessing can be performed on the data set.

III. Data Preprocessing

I have been provided with the artificial benchmark data set which contains several .arff or ARFF(Attribute-Relation File Format) files. An ARFF file is an ASCII text file that describes a list of instances sharing a set of attributes. The artificial benchmark data set provided is comprehensive as well as straightforward for application of the implemented algorithm. But some files of the data set lack an attribute called “class”. So we ignore it all together before feeding the data to the algorithm. In doing so we also avoid doing any kind of supervised clustering, as the class attribute’s column gives the label(class) of every data point or data vector. This is not desired as we intend that the algorithm takes all the data points and performs necessary tasks as explained above with efficient scanning as well as with minimal time complexity possible. After importing and applying this on the data set wherever applicable, we can plot the data points just as received before passing it to the algorithm(which returns labels of sub clusters and its centroids). These scatter plots for each file of the artificial data set is clubbed into a sub folder named “plots_before_Birch” of “plots” folder. For the sake of visualization, a few of those plots have been illustrated in the figures below (Fig. 4, 5, 6).

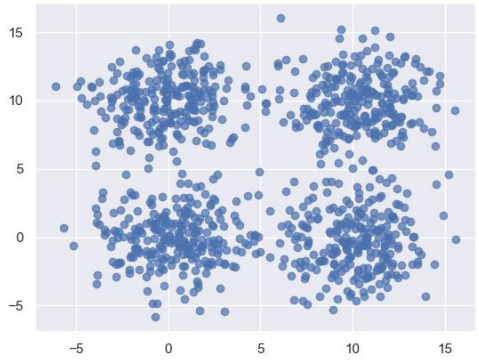

Fig. 4

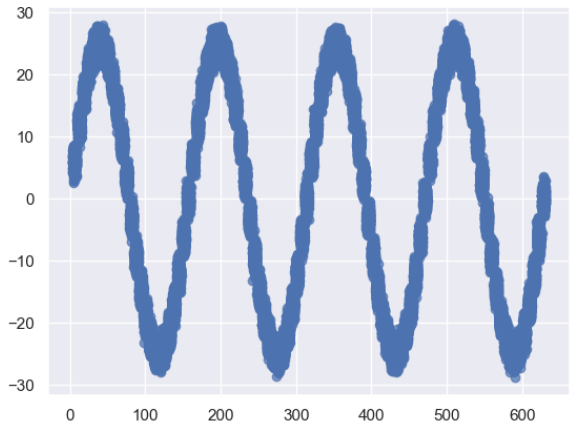

Fig. 5

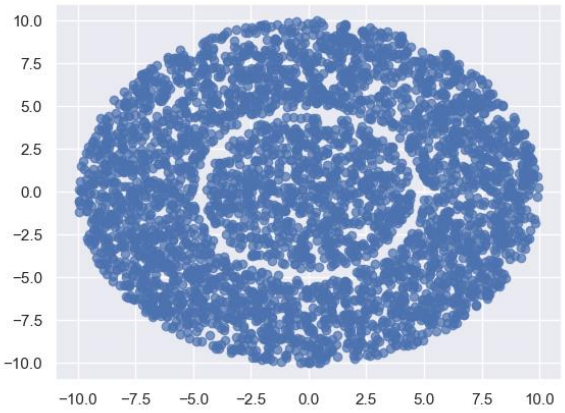

Fig. 6

IV. Techniques for obtaining results

There are very good number of techniques in data mining to obtain the required results depending on the nature and characteristics of data. The BIRCH algorithm which is implemented here is itself a very efficient technique to boost the performance of Agglomerative clustering.

Implementation of the algorithm from scratch requires extensive knowledge of Object oriented Python as we need to implement classes for CF node, CF sub cluster, to maintain a DLL(Doubly Linked List) for final leaves of process etc.. and call for creation of objects of these classes wherever required. It is also required to declare, initialize and implement several functions on these objects so as to carry out various tasks as explained above in the process of algorithm.

For calculating pairwise distance between each pair of data vectors of the data set, there are several distance metrics which can be implemented to function. But for this case **Euclidean distance** is implemented after looking at majority of the data set and its data types. On the basis of this distance metric we generate a distance matrix chunk by chunk, stacking all the pairwise distance metrics calculated. These can be seen in the "euclidean_distances" and "pairwise_distances_chunked" functions of the implementation file "BIRCH_algo.ipynb" in "code" folder.

Regarding the node to be to be split if there is no place for a new sub cluster in the node, a "splitNode" function has been implemented which creates two new sub clusters and two new nodes of Cluster Frequency (CF), setting these two CF nodes as children to their corresponding SCs (Sub Clusters). The pair of distant sub clusters are then found and the DLL (Doubly Linked List) of leaf nodes is arranged in accordance with the split. The properties of the empty sub clusters and nodes created at start are also updated according to the nearest distance between the sub clusters to the pair of distant sub clusters. The two nodes are set as children to the two sub clusters finally again so as to allow for updates (if any) and are returned.

Classes for each node in a CFTree, i.e, CFNode and each sub cluster in a CFNode, i.e, CFSubcluster are implemented which contains several variables and methods which will be utilized by the main class called Algorithm. Algorithm is the main class which implements the BIRCH algorithm. It contains a "buildCFTree" function to build Cluster Frequency(CF) Tree based on the data fed. As explained above in the algorithm, the final leaf nodes are each a node of a DLL. So each node will have a next_node and prev_node pointers apart from containing CF s of sub clusters as its node data. Leveraging the same we implement an "obtain_leafnodes" function to obtain all the leaf nodes(DLL of nodes) with help of their next pointers.

Once we have these classes with functions implemented we use them in main class – "Algorithm" of the implementation file "BIRCH_algo.ipynb" residing in the "code" folder. This algorithm takes only **2(two)** parameters which_are:

1. **Threshold (T) -** The radius of the sub cluster obtained by merging a new sample and the closest sub cluster should be lesser than the threshold. Otherwise a new sub cluster is started. Setting this value to be very low promotes splitting and vice-versa
2. **Branching Factor (B) -** Maximum number of CF sub clusters in each node. If a new samples enters such that the number of sub clusters exceed the branching factor

then that node is split into two nodes with the sub clusters redistributed in each. The parent sub cluster of that node is removed and two new sub clusters are added as parents of the 2 split nodes.

We create an instance of this algorithm class and initialize the parameters with the values we send as arguments during instantiation. We then call the buildCFTree function on this instance and pass our preprocessed data set as an argument. After fitting the CF tree we now call a function to get labels for sub clusters (after clustering) and their CF s from which their centroids can also be obtained. These labels are eventually passed to matplotlib's scatter function to visualize the output(set of sub clusters) of the implemented algorithm.

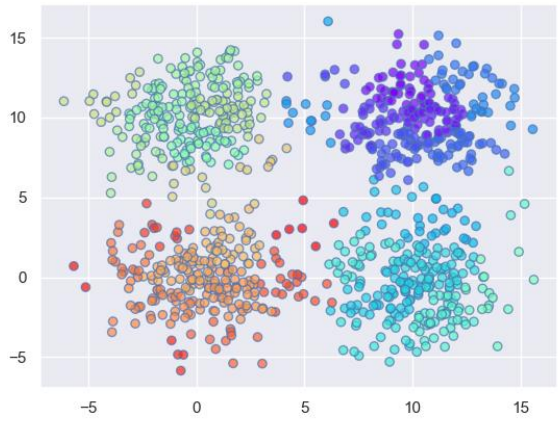

Fig. 7

V. Results and their Visualization

After calling the "getLabels" function of our algorithm class, we get the labels of sub clusters and store them in a list data structure. Intuitively these labels are nothing but centroids of sub clusters present in the leaf nodes of initially constructed CFTree. The results of application of the algorithm on each data set file is captured and saved in the "plots" folder. This folder contains 2 sub folders. One of those named "plots_before_Birch" contains the plots of data just as is imported from the data set folder whereas the other folder named "plots_after_Birch" contains the plots after application of the algorithm on the data set. The naming is followed for these plots to conveniently match its corresponding data set file from the " data " folder.

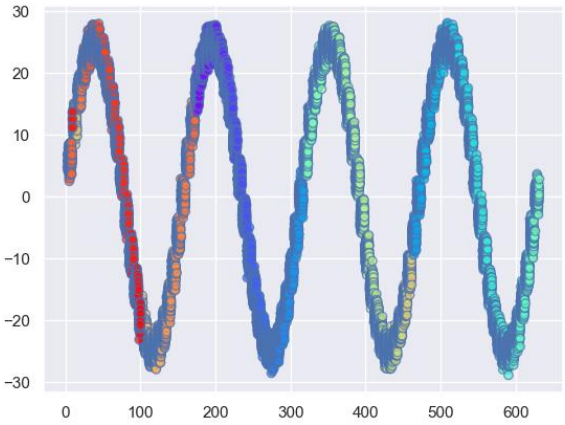

Fig. 8

Corresponding to the scatter plots shown in the Data Preprocessing section above, the results of application of algorithm on the same data set files is shown in Fig. (7, 8, 9).

Though,note that complete results for each and every data file of the artificial benchmark data set is clubbed and consolidated at a place in "plots" folder. These three figures are just to give an illustration of how these visualization scatter plots look.

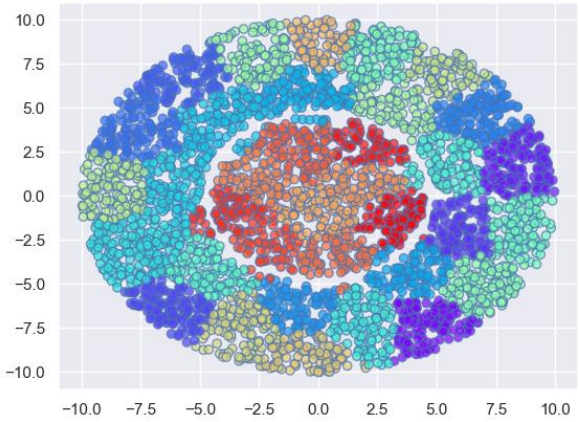

Fig. 9

VI. Conclusion and Notes

The results as can be seen from the above figures portray how different sub clusters are obtained after the application of implemented algorithm. Each sub cluster is given different color(from the list of rainbow colors) so that it is identifiable.

An important thing to be considered and noted here is that, these results are **NOT** the output of a hierarchical clustering algorithm like Agglomerative or Divisive. This is in fact an extension to such hierarchical clustering algorithms so as to pre compute, obtain and present all the possible sub clusters from a given huge data set in accordance to the parameters passed. **This serves as reduction in a lot of data points as the output is now just pertained to centroids of the sub clusters obtained from algorithm**. Hierarchical clustering algorithm – Agglomerative clustering can be now applied on this reduced data set (just centroids of sub clusters) and thereby its flaw of not being able to cater for large or huge data sets is now addressed.

VII. References

[1] https://towardsdatascience.com/birch-clustering-clearly-explained-ffd75f07e5ed

[2] *Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996.* ***BIRCH: an efficient data clustering method for very large databases****. In Proceedings of the 1996 ACM SIGMOD international conference on Management of data (SIGMOD '96). Association for Computing Machinery, New York, NY, USA, 103–114.*

[3] Techniques taught in class.

[4] https://github.com/grabcollab18/BIRCH-implementation-from-scratch-on-artificial-datasets