Faculty of Mathematics and Computer Science

Emanuel Bîscă

Advanced Methods of Data Analysis

Birch Clustering Algorithm

2020

Department of Computer Science

Abstract

In our society, with things evolving so fast, the data sets are growing larger and larger. As a result, the tools handling those data sets and the problem generated by them must evolve as well. Nowadays, the clustering algorithms are regaining a lot of attention, together with the rise of parallelized computing architecture development. Although the progress exists and it's easy observable, most of the algorithms used intensely in clustering suffer from two major drawbacks: they do not scale well with the increasing dataset sizes and quite often they need proper parametrization which is usually difficult to provide. In my report, we will talk about the BIRCH clustering algorithm, which is only an idea of how to deal with this kind of issues. I present a theoretical insight of BIRCH concepts and a new fashion A-BIRCH algorithm that comes to resolve other arosed problems.

Contents

Chapter 1

Introduction

If we think about data mining, one important task of it, that comes first in mind to most of us, is clustering. It is useful when someone wants to group objects, which are believed to be similar in one way or another. The clustering process also means data grouped into clusters or classes, after a specific set of rules: objects with a high similarity are being disposed into the same cluster, while the ones which posses a dissimilarity it is very likely to be sent to other formed clusters (Berkhin 2006). Two of the most used methods in which clustering is done, usually, are Hierarchical methods or Partitioning methods.

In this paper, I present a clustering algorithm that works in the hierarchical way, so it uses something called by the researchers, a *tree of cluster*. Further, the hierarchical way of doing clustering, has two different approaches, agglomerative clustering and divisive clustering. Graphically speaking, these hierarchical algorithms adopt a display in a form of a tree, called in the literature a *dendrogram*.

Not everything has to be perfect, because indeed once it has performed the merge or the split step, it cannot be undone under normal circumstances, in the hierarchical approach. This can be seen as a lack of flexibility, but actually if we don't want to worry about the great amount of different choices that are involved, this way of doing things are leading to smaller costs in terms of computations (Rani and Rohil 2013).

1.1 Scope

Generally speaking, clustering algorithms encounter two different problems that required attention through time. The algorithms used in clustering, usually, do not scale well as a result of their usual complexity of $O(N^2)$ or $O(NW)$. Here N was is computed as the total number of data points and W is represents the cluster count. Second problem that has concerned scientists, is around the idea of how to identify the cluster count. My scope is to talk about an algorithm that has been present in the field for a long time and its concept still remains in use nowadays.

Zhang et al. (1997) provided us with a method for resolving our dispute: BIRCH a form of an agglomerative hierarchical clustering, proved to be a wise option for those working with larger than normal datasets or databases. The letters from BIRCH stand for *Balanced Iterative Reducing and Clustering using Hierarchies*. It produces high quality clusters with the resources at hand, by clustering, both incrementally and dinamically, the metric data points which can be multi-dimensional. BIRCH is offering a good cluster, scanning the data one single time ; however, for an improved quality it can work supplementary. Handling noise, has been a well known issue, and BIRCH was the first algorithm able to offer a solution to researchers.

1.2 Contributions and Limitations of BIRCH

As I previously mentioned, BIRCH deals only in problems with metric attributes. One attribute is a metric one, if it is possible to express or represent this specific attribute using certain coordinates inside an Euclidean space. Comparing BIRCH to older work in the field, there can be distinguished a series of new characteristics. An important one is related to the way in which the clustering problem is stated, so that BIRCH stands as an appropriate choice when one has to work with large datasets. This is achieved by explicitly making the memory constraints and time. Other distance based approaches, previously developed, have numerous disadvantages with respect to BIRCH:

- The locality of BIRCH means that for each clustering, there is a decision coming out, without knowing all the existed clusters at that time; also there exist data points that were still not scanned. In the matter of natural closeness, BIRCH has made measurements which, simultaneous, could be incrementally maintained while the process of clustering is

in place.

- Most of the times, the data space being not uniformly occupied leads to the conclusion that not each single data points matters with the same importance, having in mind the clustering scope, and BIRCH is exploiting this idea. Here the terms of outliers intervene, because they can optionally be removed if they are in spare zones of the data space, otherwise the points densely located are interpreted as one cluster.
- The accuracy and efficiency are ensured in BIRCH. First, because the memory is fully used in order to obtain subclusters that are as finest as possible, while the second is provided by a minimized I/O cost.

1.3 Background

It's important to define some concepts so that everything would be clear in the next sections. Our cluster needs a centroid, a radius and a diameter.

Let N be the number of d-dimensional data points contained in the considered cluster $\vec{X_i}$, pay attention to the fact that $i = 1, 2, ..., N$, then we consider a **centroid** with the following notation $\vec{X}0$, the **radius** R and last but not least the **diameter** D of this cluster. By definition, they adopt the following mathematical representation:

$$\vec{X}0 = \frac{\sum_{i=1}^{N} \vec{X_i}}{N} \tag{1.1}$$

$$R = (\frac{\sum_{i=1}^{N} (\vec{X_i} - \vec{X}0)^2}{N})^{\frac{1}{2}} \tag{1.2}$$

$$D = (\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (\vec{X_i} - \vec{X_j})^2}{N(N-1)})^{\frac{1}{2}} \tag{1.3}$$

R must be seen as the mean distance from the member points to centroid, while D is equivalent

with the average pairwise distance from the inside of a cluster (Vijayarani and Jothi 2013). The tightness of the cluster around the centroid can be measured alternatively by these two. Furthermore, we need to talk about alternative distances that can be used when measuring the closeness of two clusters.

Let's consider two clusters with their given centroids $X\vec{0}_1$ and $X\vec{0}_2$. Two important aspects needs discussion: $D0$ (that would be the **centroid Euclidean distance**) and $D1$ (known as the **centroid Manhattan distance**) of these considered clusters. Their definition is presented in the next lines:

$$D0 = (X\vec{0}_1 - X\vec{0}_2)^2)^{\frac{1}{2}} \tag{1.4}$$

$$D1 = |X\vec{0}_1 - X\vec{0}_2| = \sum_{i=1}^{d} |X\vec{0}_1{}^{(i)} - X\vec{0}_2{}^{(i)}| \tag{1.5}$$

Having in mind that the notation N_1 means d-dimensional data points existent in a cluster noted by $\vec{X}_i$ with the remark that $i = 1, 2, ..., N_1$. The same thing happens for N_2 data points which we considered to be part of another cluster noted $\vec{X}_j$ with $j = N_1 + 1, N_1 + 2, N_1 + N_2$ Rani and Rohil (2013). These being said, three notions must be defined:

- **average inter-cluster distance** noted for further mentions by $D2$
- **average intra-cluster distance** noted $D3$
- **variance increase distance** noted $D4$

$$D2 = (\frac{\sum_{i=1}^{N_1} \sum_{j=N_1+1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{N_1 N_2})^{\frac{1}{2}} \tag{1.6}$$

$$D3 = (\frac{\sum_{i=1}^{N_1+N_2} \sum_{j=1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{(N_1 + N_2)(N_1 + N_2 - 1)})^{\frac{1}{2}} \tag{1.7}$$

$$D4 = (\sum_{k=1}^{N_1+N_2}(\vec{X_k} - \frac{\sum_{l=1}^{N_1+N_2}\vec{X_l}}{N_1+N_2})^2 - \sum_{i=1}^{N_1}(\vec{X_i} - \frac{\sum_{l=1}^{N_1}\vec{X_l}}{N_1})^2 - \sum_{j=N_1+1}^{N_1+N_2}(\vec{X_j} - \frac{\sum_{l=N_1+1}^{N_1+N_2}\vec{X_l}}{N_2})^2)^{\frac{1}{2}} \tag{1.8}$$

1.4 The Clustering Feature

Two main concepts stay behind the incremental clustering of BIRCH: the **Clustering Feature** together with the **CF** tree. About any considered cluster, it is possible to maintain the known information in the form of triple that summarize it, which is the **Clustering Feature** (Zhang et al. 1997).

Definition 1.1. *Let there be $\vec{X_i}$ a usual cluster, with respect to $i = 1, 2, ..., N$ where there are N d-dimensional data points contained by it. A triple denoted by $(N, \vec{LS}, SS)$ is called a* ***Clustering Feature (CF)*** *vector of the cluster $\vec{X_i}$, if*

- *N is considered the number of data points naturally contained by the cluster,*
- *the vector $\vec{LS}$ is the linear sum $\sum_{i=1}^{N}\vec{X_i}$ of the studied data points,*
- *SS must be the square sum $\sum_{i=1}^{N}(\vec{X_i})^2$ of the data points that we're working with.*

Theorem 1.1 (CF Additivity Theorem)**.** *Let there be two disjoint clusters, with their* ***CF*** *vectors denoted by $CF_a = (N_a, \vec{LS_a}, SS_a)$ and $CF_b = (N_b, \vec{LS_b}, SS_b)$. If we consider merging these two clusters, the resulting CF_{ab} vector must be in the form:*

$$CF_a + CF_b = (N_a + N_b, \vec{LS_a} + \vec{LS_b}, SS_a + SS_b) = CF_{ab} \tag{1.9}$$

Proof. Straightforward algebra does a full proof of those said above. □

Indeed a set of some data points, can be seen as a cluster, but the truth is that only the **CF** vector is summarized. So the **CF** being stored as summary is enough for obtaining the measurements needed in BIRCH when there are clustering decision to be performed.

1.4.1 CF tree

Two parameters are of interest when talking about a height-balanced tree such as a **CF** tree: B being the branching factor and the threshold T. The entries of a nonleaf node which has mostly B entries, adopt the configuration $[CF_i; child_i]$, with $i = 1, 2, ..., B$. CF_i is nothing more than a CF entry of a subcluster, which constitute the representation of a pointer to the i-th child node.

On the other side, L entries are most specific for the leaf nodes, with each entry being a cluster feature. Moreover, this time, two pointers are used when each leaf nodes are chained together: *prev* and *next*. In terms of requirements for a threshold value, a leaf node with all his entries must respect some limitations. Both the diameter and the radius must have values lesser than the one of the threshold T. As it is expected, a function that depends on T gives us the tree size. The relation between these two is simply explained as the smaller we need our tree to be, the more T has to be increased.

Chapter 2

BIRCH Algorithm used for Clustering

2.1 BIRCH Explained

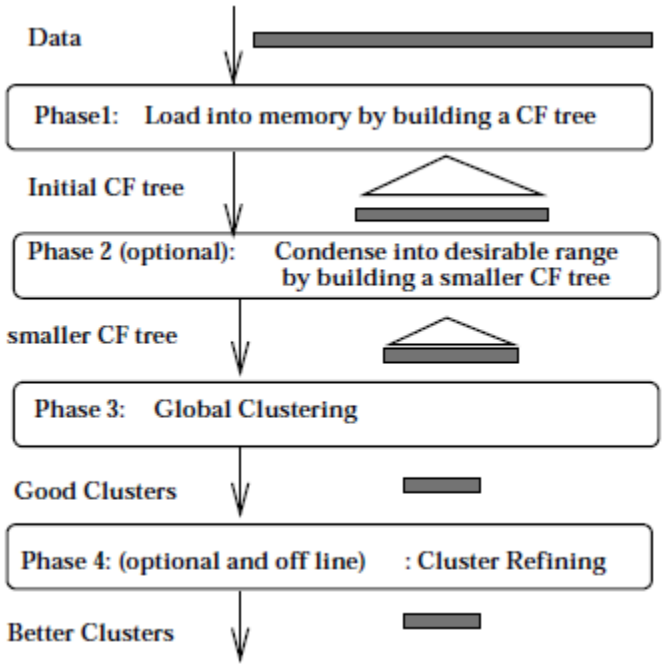

Figure 2.1: BIRCH overview

I believed it to be important, so I introduced an overview of BIRCH algorithm in the above figure. There are a total of 4 phases to be discussed. The first one consist of building an initial **CF** tree after all the data has been scanned and it uses a predetermined amount of memory.

There are two ways in which data can be interpreted: the data points with a high density are organized in fine subclusters, while those which are sparse would be removed as outliers. We remark that in Phase 1 a summary of the considered data is created in the memory. Next steps computations will be done subsequently fast and accurate.

- fast, because the initial problem of clustering is reduced to clustering the subclusters in entries of type L (so leaf ones) and as a consequence we don't need further I/O operations.
- accurate, as we get rid of many of the existing outliers, so the remaining data would be reflected with the highest granularity.

As it can be seen, the second phase is not mandatory but has its own importance as it links the results obtained in first phase with the third phase input range. Speaking in terms of quality and speed, it has been pointed out that the clustering methods (which can be global or semi-global) used in the third phase posses distinct input size ranges, so it is necessary to make the transition smooth. So we enter the fourth phase having a set of clusters that display the main distribution patterns from the considered data. The fact that the third phase is being applied on just a summary of the available data, makes it obvious that there might exist inaccuracies, that are minor ones or localized most of the time. Last phase, the fourth one, can also be optional but in this step, the inaccuracies that we just talked about can be corrected, with the cost of some further passes over our data. It is indeed remarkable, that the originally considered data was, up to this step, scanned only one time while the tree and the information regarding to the outliers may have been in the process of scanning several times. The centroids of the third phase's resulting clusters are being used as seeds for a new suite of clusters.

The next figure, is a detailed view of the Phase 1, which demands further attention.

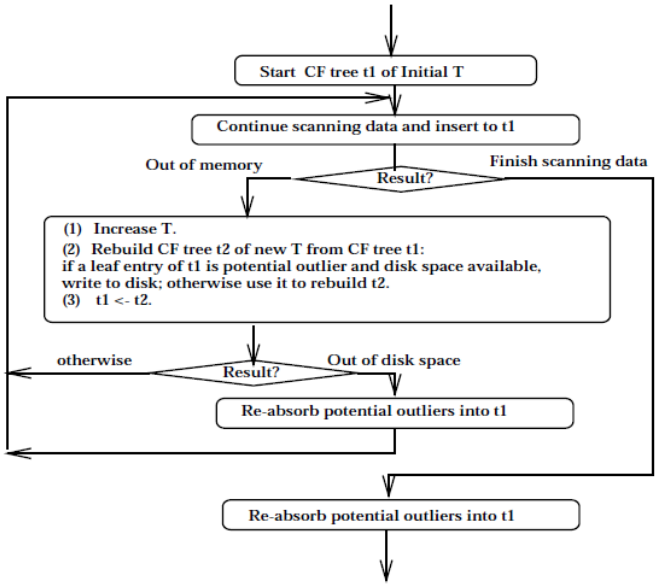

Figure 2.2: Phase 1 Control Flow

At the beginning it scans the data points, then with respect to the initial threshold value, it inserts them into the cluster feature tree. Sometimes it is possible to run out of memory before getting to read all the data. In this case, a new cluster feature tree would be rebuild. We remark here that the new tree would be a smaller one, due to the increased value of the threshold. The leaf entries need to be reinserted into the new tree, soon after that the process of data scanning would be resumed.

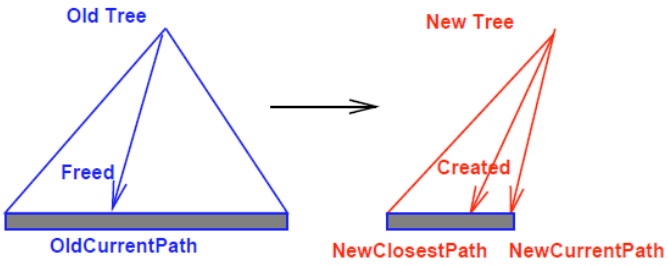

Figure 2.3: CF Tree being rebuild

2.1.1 Reducibility

Let us consider a t_i the notation for our **CF** tree being characterized by its corresponding threshold T_i. Two main concepts may need further mentions, the height of this cluster feature tree and the size of it. Naturally, these two are denoted by h, respectively S_i. The number of nodes must be seen as the size of any cluster feature tree that we've been talking about. We

define our problem as building a new **CF** tree in such a manner the size of the new **CF** tree should not exceed the number of nodes contained by the older **CF** tree. All the leaf entries of this older tree must be used. In other terms we have the relation $T_{i+1} \geq T_i$, where T_{i+1} is the threshold value of the t_{i+1} as the new cluster feature tree. Also, S_i must be greater than the number of nodes in t_{i+1}.

If we consider that for every node from t_i, the labels are from 0 to $n_k - 1$, for all to make sense, all the entries corresponding to a specific node are denoted n_k. Then, from an entry in the first level (the root) a path can be build to level h (which is the leaf node). Such a path $(i_1, i_2, ..., i_{h-1})$ can be exclusively represented once; here the label $i, j = 1, 2, ..., h-1$ means the j-th level entry on the chosen path.

Figure 2.3 render the algorithm in question. The one tree that is new, begins with NULL, while the far left side path from the older tree serves as beginning for the *OldCurrentPath*. The *OldCurrentPath* has an algorithm that states the following:

1. Creating the proper *NewCurrentPath* into the new tree;
2. It loads the entries which are leaves from *OldCurrentPath* into the new tree;
3. Space is being freed in both paths;
4. For any further purpose in which the previous steps should be repeated, *OldCurrentPath* is established as the next path in the older tree.

Theorem 2.1 (Theorem of Reducibility)**.** *Let us believe we used the algorithm just discussed to built a CF tree denoted by t_{i+1} from a previous t_i. Let T_{i+1} be the corresponding threshold of the first CF tree, while the S_{i+1} would be its number of nodes, the size. As of the initial CF tree, we had the two S_i and T_i. We consider the height of t_i to be h, then at most h pages of memory would be needed to transform t_i into t_{i+1} if the next propositions are equivalent:*

- $T_{i+1} \geq T_i$;
- $S_{i+1} \leq S_i n$;

2.1.2 Threshold values – an issue

The number of rebuilds can be dramatically reduced by a good choice of threshold value. It is possible to adjust for the case in which the threshold value is to low, as the initial T_0 value of the

threshold is increased dynamically. But what happens if T_0 has a value too great? The result of our algorithm will be a **CF** tree with lesser details than it must have in order to be feasible. All that because of the memory available. A conservative way of setting the initial threshold value must be adopted. By default, BIRCH has the T_0 set as zero. Having a T_i too small leads us to run out of memory after scanning only N_i data points and building C_i leaf entries (satisfying the threshold value T_i). The algorithm must choose a next T_{i+1} as a threshold value relying its decision only on the amount of data scanned already. The estimation problem has been proved to be a difficult one.

2.2 Threshold Automatically Estimated

As we talked in Section 1.1 one of the most difficult parameter to be appreciated is the cluster count. This have been a hot topic for other researchers that have developed a new approach in terms of clustering algorithms. I believe that A-BIRCH is one of the reasons that proves the progress in terms of science. A-BIRCH offers an automated way of estimation the threshold value, used extensively today.

Further mentions should be simplified by a small different notations: we shall refer as tree-BIRCH to the original BIRCH that lacks its global clustering phase. Although tree-BIRCH does not need as input the cluster count,as one result of missing the global clustering phase from the original BIRCH, a bad quality of clustering is what specifically makes tree-BIRCH undesirable for some of us.

We consider two conditions that constraint the obtaining of an optimal threshold approximation. Firstly, it is desirable that the BIRCH has a flat tree, which can be achieved by having a large enough B (branching factor). Secondly, a similar number of elements must be contained in all the clusters. The whole idea is to obtain a function with the parameters D_{min} that is the minimum cluster distance and R_{max} as the maximum radius between a centroid and other data points contained in a cluster. We presumably believe that both are known. Lorbeer et al. (2016) came with one idea of how to find D_{min} and R_{max}, if they're still not explicit. We presume that D_{min} and R_{max} are specific for a tiny subset of the data, so that this subset is a representative one for the entire data. On this small dataset, we apply Gap Statistic, with the wish to secure k, the cluster count. Then k-means uses this cluster count k, to compute a clustering of the data set. In the end, the D_{min} and R_{max} are returned from this process. There exist many scenarios

of interests that can be run by BIRCH with respect to the same unchanged conditions, still, each time a different sample of data is used. From cluster combining and cluster splitting are resulting the desired probabilities of D_{min} and R_{max}.

2.3 Concluding Remarks

The recent years, have brought advances in technology and science. This continuous process of progress is allowing us to automatically record transactions of everyday life at a faster rate than before. It leads to large amounts of data which can and must be analyzed in one way or another. BIRCH has proven to be a data streams clustering technique that can handle those data. Its concept of ouliers detection are efficiently implemented so the performance is enhanced. Furthermore, it has been presented that other versions of BIRCH have been developed through time, so that the algorithm can be used in various issues.

References

Berkhin, P. (2006). "A survey of clustering data mining techniques". In: *Grouping multidimensional data,* pp. 25–71.

Lorbeer, B., Kosareva, A., Deva, B., Softic, D., Ruppel, P., and A., Kupper (2016). "A-BIRCH: Automatic Threshold Estimation for the BIRCH Clustering Algorithm". In: *Advances in Intelligent Systems and Computing,* 529, pp. 169–178.

Rani, Y. and Rohil, H. (2013). "A Study of Hierarchical Clustering Algorithm". In: *International Journal of Information and Computation Technology,* 3.10, pp. 1115–1122.

Vijayarani, S. and Jothi, P. (2013). "An Efficient Clustering Algorithm for Outlier Detection in Data Streams". In: *International Journal of Advanced Research in Computer and Communication Engineering,* 2.9, pp. 3657–3665.

Zhang, T., Ramakrisnan, R., and Livny, M. (1997). "BIRCH: A New Data Clustering Algorithm and Its Applications". In: *Data Mining and Knowledge Discovery,* 1, pp. 141–182.