Global Density Clustering

Scott Cohen* SCOTTCOHEN9999@GMAIL.COM
36 Calvin Road, Newton Massachusetts

Abstract

This paper presents Global Density Clustering, (GDC), an algorithm that has several major advantages over the most popular existing clustering algorithms: (1) No parameters are chosen at the outset of the function; rather, the user can control the desired resolution as clustering proceeds. (2) GDC runtime is O(MN log N) where M is the dimension and N is the number of data points, that is, the dataset size. It is suitable for big data. (3) GDC defines clusters as: points within a cluster are closer than distance dist to their nearest neighbor in the cluster (dist is not picked at the outset but rather it is chosen as the algorithm is progressing) and all points outside the cluster are further than dist from any point in the cluster. (4) GDC supports variable density without the many special data structures such as HDBSCAN needs. One reason that GDC has these advantages is it searches for and considers points whose nearest neighbor are furthest apart before searching for those that are closer together. Other novel approaches to the main problems of clustering such as noisy backgrounds are described.

Keywords: unsupervised learning, clustering, density clustering, hierarchical clustering, nearest neighbor algorithms.

1. Introduction

There is no perfect clustering algorithm. According to the Impossibility Theorem for clustering, no clustering algorithm has all the desirable properties of clustering.(Kleinberg, 2002). Multiple valid meaningful clusters can be constructed from the same data. See section 3.2 in Jain (2010). This paper, however, proposes an algorithm that has many advantages compared to the most popular existing choices.

D. Xu (2015) is a comprehensive survey of clustering algorithms with a table that tabulates the strengths of many clustering algorithm on 7 aspects: time complexity; scalability; handles large-scale data; handles high dimensional data; suitable shape of cluster; sensitive to sequence of inputting data; sensitive to noise and outliers. Large-scale data includes the issues of the 4 Vs: volume, velocity, variety, and incomplete veracity.

There are also, however, other issues that arise for some but not all clustering algorithms. Some of these are: whether parameters need to be chosen before the algorithm runs especially if the parameters are difficult to choose and the results are sensitive to that choice; whether the algorithm is easily explainable especially in regulated fields since a person oversees the result; how well does the algorithm handle streaming of a large update of data; does it handle categorical data; does it have unusual data structures that take time to construct, interpret, and explain.

The following comparisons touch on the above issues and are organized by discussing them in the context of the most well-known and popular current clustering algorithms.

* Thanks to my family for their support

1.1. GDC and k-means clustering

There are many disadvantages of k-means clustering that do not occur with GDC.

GDC has no equivalent of k, let alone an arbitrary choice of k. In fact, no parameters are chosen at the outset of the algorithm; as the algorithm progresses, the user has the option to stop it when user likes the clusters found thus far. The user decides when the resolution is right. A suggestion to stop is made when nearest neighbors get much closer together than before. This is the GDC solution to one of the most difficult problems in clustering. See section 3.3 in Jain (2010).

GDC captures many kinds of clusters that k-means misses even in the simplest of situations such as in 2-dim let alone in higher dimensions. See section 3.1, figure 5 in Jain (2010) for one example.

GDC need not worry whether the clusters are roughly spherical as with k-means. See table 22 in D. Xu (2015).

GDC supports all this is efficiently and in an arbitrary number of dimensions.

GDC is suitable for big data with or without any sampling. The entire (big) database can be efficiently clustered.

GDC has no concern about sensitivity to an initial choice of centroids. Perhaps the biggest advantage of k-means is that it is fast: $O(tMN)$ where M is the number of features, N is the dataset size, and t is the number of iterations of the k-means algorithm. Algorithmically, global density clustering is not far behind if t is at least slightly bigger than $\log N$ which is not unreasonable. Global density clustering is $O(MN \log N)$ so the comparison depends on t versus $\log N$. Sort routines even on big data are so highly optimized and $\log N$ even for big data is probably somewhat smaller than 40. So, the comparison of runtimes is reasonable.

1.2. GDC and other density algorithms

Although global density clustering is, as its name suggests, a density algorithm, it does not suffer from some of the disadvantages of other density algorithms such as DBSCAN, OPTICS, and HDBSCAN.

GDC has no need or use for parameters *minpts* or *epsilon* let alone any arbitrary choice of them as with DBSCAN and OPTICS.

GDC catches clusters that would be missed because of the arbitrary choice of epsilon as with DBSCAN and OPTICS, especially in higher dimensions.

GDC's worst-case runtime is better than DBSCAN, OPTICS, HDBSCAN which are worse than $O(N \log N)$. See E. Schubert and Kriegel (2017). It is $O(MN \log N)$ where M is the number of features (i.e. dimensions) and N is the number of points in M-dim (i.e. the dataset size). This is the difference between practical and for the worst-case situation prohibitively intolerable as with DBSCAN, OPTICS and HDBSCAN.

GDC has no need or use for a dendrogram as with OPTICS.

GDC has no need or use for a minimum spanning tree, cluster hierarchy, condensed tree or any other special purpose structures that take time to construct, interpret, and explain as with HDBSCAN.

GDC has an easy explanations of what the algorithm is doing unlike HDBSCAN. Density algorithms all have a major benefit in their intuitive definition of the clusters that are

uncovered, and the arbitrary cluster shapes supported. Likewise, GDC has those benefits. GDC definition of a cluster is that points are within a cluster if they are closer than distance *dist* to their nearest neighbor in the cluster and that all points outside the cluster are further than dist from any point in the cluster. (The distance *dist* is not picked at the outset but rather that is chosen as the algorithm is progressing.) In addition, GDC has the significant advantages described above.

1.3. GDC and hierarchical clustering

Hierarchical clustering has some serious performance disadvantages.

GDC is exponentially faster. It is $O(MN \log N)$ where M is the number of features (that is, dimensions) and N is the number of points in M-dim (that is, the dataset size).

GDC only needs two calls to sort the data per feature and one of them nearly always exists anyway because it'll be a primary or secondary index. Until GDC, the benefits of hierarchical clustering were more theoretical than practical because of its runtime performance.

1.4. Concluding remarks on comparisons

In general, choosing such parameters such as k, *epsilon*, *minpts*, and others has too much trial and error. At best they indicate some a priori knowledge such as there are 10 digits for the MNIST dataset, so we say for kmeans that there are 10 clusters at the outset of the algorithm. But choosing these parameters affect the resulting clustering which defeats the exploratory advantage of clustering algorithms. We are affecting the result before we know what the best choice for clusters is.

In general, it would be useful to avoid all these a priori choices which can affect the outcome before we start. It is valuable to be easy to explain. It is also very valuable to be usable for big data.

2. Strategy of GDC algorithm

The approach taken in this paper is, first, completely solve the problem when each data point is a scalar, that is, just a single real number and then, second, repeatedly apply the solution in 1-dim to solve when each data point is a M dimensional point where M is the number of features of the data.

The complete solution of the 1-dimensional data point used here is based on the observation that the clusters along a line with the N points are simply bounded by the largest of the $N-1$ intervals that are in descending order. This view occurs as a result of looking at the global picture and drilling top down. Although this is a density algorithm in that we look for pockets of points close to each other (usually a local view) we use top down processing (usually a global view) to find the dense pocket of data points. Hence, the name of this algorithm.

Let's say that *largest_interval_remaining* is the largest interval in one dimension. Then all, say, $k < N$ data points to the left of *largest_interval_remaining* are in one group of clusters. Likewise, all $N-k$ data points to the right of *largest_interval_remaining* are in a different group of clusters. Every point in the left group of clusters is further

than $|largest\_interval\_remaining|$ from every point in the right group of clusters. This procedure then may be, but not necessarily, repeated for one or both the left group and the right group for $second\_largest\_interval\_remaining$, and so on.

Now we discuss building up the solution from 1-dim to where each data point is an M dimensional point. The approach relies on a few observations:

First, if two points are in the same cluster then they should be in the same cluster in each projection onto an axis (even though the reverse is not necessarily true);

Second, in M-dim space, if the distance is $dist$ between two points then the projection onto at least one axis is $\geq dist/M$; otherwise the M lengths would add up to less than $dist$ when in L1 norm which is a contradiction.

Third, the minimum distance between points of different clusters is, in L1 norm,
$\sum |largest\_interval\_remaining[axis_n dx]|$
whereas the maximum distance between two nearest neighbors is
$\sum |second\_largest\_interval\_remaining[axis_n dx]|$
which can be much smaller.

A careful reading of the above description indicates that GDC is a divisive hierarchical algorithm as well as a density algorithm. As a result of the top-down consideration of clusters, variable density of the clusters is handled easily: clusters defined earlier in the top-down processing have lower density than those clusters that are defined later.

An essential reason that GDC has these advantages is it searches for and considers points whose nearest neighbor is furthest apart before searching for those that are closer together. It is a top-down or "global" consideration of distances that other density algorithms do from a bottom-up or "local" view.

2.1. Choice of norm

Distance is in L1 norm for convenience of analysis; similar situations occur for L2 norm as well as other norms. Also, the axes could be any M independent vectors although we simply choose the orthogonal coordinate axes. This choice is so that their projection onto that axis is just selecting its coordinate of the M coordinates.

2.2. Order of runtime speed

The only step that is not O(MN) in all the above for GDC is a sort of the dataset on each feature and possibly a FFT on one or more 1-dim axes (see section 7.1 on Noisy Data); even for the huge datasets of big data this is not prohibitive. Therefore, it is not an order O(N^2) algorithm. In the code for this paper, mergesort is used because of its excellent performance on big data. All other processing of data is done with sequential passes over the data so only sequential disk IO is necessary and that results in the best performance for runtime especially for big data.

3. What is a complete clustering solution?

We define a complete clustering solution in any dimensional space to be a partition of data points into clusters of 1 or more data points that satisfy two properties:

Definition 1 (Complete Clustering Solution: CCS) *A CCS is when for some number dist, where* $\boldsymbol{v1}$, $\boldsymbol{v2}$ *are two data points (that is, vectors), for all* $\boldsymbol{v1}$ *in one cluster we can find a partition into one or more sets of data points where two properties hold.*

1. *If there is more than one data point* $\boldsymbol{v1}$, $\boldsymbol{vk}$ *in the cluster then there exists a sequence of points* $\boldsymbol{v_i}$ *starting at* $\boldsymbol{v1}$ *and ending at* $\boldsymbol{vk}$ *all in that cluster,* $|\boldsymbol{v_{i+1}} - \boldsymbol{v_i}| < dist$*;*
2. *For all* $\boldsymbol{v2}$ *in a different cluster,* $|\boldsymbol{v1} - \boldsymbol{v2}| \geq dist$

Note that this paper uses the L1 norm because it is easier to analyze in this situation than the L2 norm. So if dist is the distance between $\boldsymbol{v1}$ and $\boldsymbol{v2}$,

$dist = \sum |\boldsymbol{v1}[axis\_ndx] - \boldsymbol{v2}[axis\_ndx]|$

There is always some choice of clusters that satisfies CCS properties (1) and (2). The argument is the following. For each point $\boldsymbol{v1}$: add all points to the cluster that are within distance *dist* of $\boldsymbol{v1}$; keep repeating the previous step for each point in the cluster until either all points in the entire dataset are in the cluster or not. This satisfies CCS property (1). If not all points are in the cluster containing point $\boldsymbol{v1}$ then all the remaining points are distance $\geq dist$ from all points in the cluster which is CCS property (2) Now we can repeat the above steps by finding a cluster that is distinct from the cluster containing point $\boldsymbol{v1}$.

In order to get a little more understanding, consider that as *dist* increases, clusters get larger and there are fewer clusters. The largest *dist* can be is the diameter of the dataset in M-dim. As *dist* decreases the clusters get smaller and eventually there are N clusters where each is an isolated point like an outlier. The smallest *dist* can be is the smallest interval which can be close to zero.

So, letting *dist* start out as the diameter or larger of the dataset in M-dim, we start top-down from a global view with one cluster and as *dist* gets smaller when we set it to the largest remaining distance between two points. We iteratively apply the first step to either the left group of clusters or to the right group of clusters.

We continue decreasing *dist* until the next interval is a dramatic drop in size (the ratio is tiny) or until the user is satisfied with the clusters found thus far. A dramatic drop in size might indicate that any more clusters will already be dense enough to form a single cluster. But the user can override this and continue clustering.

One could think of CCS property (1) as a local statement about nearest neighbors in just one cluster and just about its insides. On the other hand, CCS property (2) is basically a global statement about points that are not nearby but far apart in two clusters, that is, global objects.

Picking the largest interval on an axis may in practice initially pick out the outliers and then will probably start to divide the data points roughly in half. If so then there will be roughly the number of outliers plus $\log(N)$ repetitions which helps make this an even faster algorithm. Regardless, the worst case runtime is only $O(MN \log N)$.

4. Pseudo-code for 1-dimensional data points

Algorithm 1 is the python pseudo-code for GDC in 1-dimension. It is a CCS.

Algorithm 1: The complete solution in 1-dimension

1. $sorted\_N\_points$ = sort the N 1-dim data points that are on the line.
2. $largest\_interval\_array$ = sortInDescendingOrder(diff2Consecutive($sorted\_N\_points$))
3. Start with a $dist >$ diameter of a cluster containing the entire dataset.
4. for $interval\_ndx$ in range(N):
   (a) $largest\_interval\_remaining = largest\_interval\_array[interval\_ndx]$
   Say $largest\_interval\_remaining$ is bounded by data points I_L on the left and I_R on the right. Then I_L and I_R must belong to the same cluster because that interval is chosen in descending order of size.
   (b) Since I_L and I_R both belong to the same cluster and that cluster is bounded by data points C_L on the left and C_R on the right: then replace that cluster with the two clusters;
      i. data points in the interval $[C_L, I_L]$ is a cluster
      ii. data points in the interval $[I_R, C_R]$ is another cluster
   (c) If $largest\_interval\_array[interval\_ndx + 1]$ is much smaller than $largest\_interval\_remaining$: then ask the user whether to quit
   (d) reduce $dist$ to $dist = largest\_interval\_array[interval\_ndx]$
5. # when done the clusters satisfy CCS properties (1) and (2)

5. The approach for 2 or more dimensional data points

For simplicity, let's start with 2-dim points, so M is 2. Let the 1-dim point A_x be the 2-dim point A projected onto the x-axis and 1-dim A_y be the point A projected onto the y-axis. Likewise, for 2-dim point B and 1-dim points B_x and B_y. Then either (remember that $M = 2$ for now)

$|B_x – A_x| > |B – A|/M$

where the left side of the inequality is distance along a 1-dimensional interval bounded by 1-dim points A_x and B_x and the right side of the inequality is an M-dim distance along a M-dimensional line segment bounded by M-dim points A and B; or

$|B_y – A_y| > |B – A|/M$

for the 1-dim points A_y and B_y. This either/or inequalities is because of the L1 norm where

$|B – A| = |B_x – A_x| + |B_y – A_y|$

Define K_{dist} as some cluster in M-dim containing A for distance $dist$, and define $K_{dist(x)}$ is the cluster of the projection of K_{dist} onto the x-axis, in 1-dim containing A_x for distance $dist$.

By the definition of $K_{dist(x)}$, for every axis x, $K_{dist(x)}$ is a cluster in a CCS and therefore it satisfies

CCS property (2) if $A_x \in K_{dist(x)}$ and B_x is not then $|A_x - B_x| > dist$.

CCS property (1) Let A_x and $B_x \in K_{dist(x)}$ and A_x and B_x be nearest neighbors.

Define r_x by

$r_x = (second\_largest\_interval\_remaining/largest\_interval\_remaining)$.

Notice that, for each axis, $r_x \leq 1$ because $second\_largest$ is always smaller or equal to largest. Therefore,

$|A_x - B_x| < r_x dist$.

Also define inverse() to take a set of 1-dim points $K_{dist(x)}$ and map them to the set of M-dim points that project onto $K_{dist(x)}$. inverse() is possibly not a function but rather just a one-to-many mapping.

Theorem 2 (Finding a cluster in a CCS in multi-dim) *If K' is the set of dataset points in M-dim space defined by*

$K' = \{\cap inverse(K_{dist(x)})\}$

Then K' a cluster in a CCS if $\sum r_x < 1$.

Proof *K' satisfies CCS property (2) because of the following.*

Say $A \in \{\cap inverse(K_{dist(x)})\}$ where the intersection is over every axis and B is not in K', that is $\cap inverse(K_{dist(x)})$. Then

$B \in \{\cup\{inverse(K_{dist(x)})^{complement}\}\}$

so for at least one axis x, $B \in \{inverse(K_{dist(x)})\}^{complement}$.

$B \in \{set\ of\ all\ M\text{-}dim\ points\ Z\ s.t.\ |Z_x - A_x| < dist\}^{complement}$.

For at least one axis x, $|B_x - A_x| > dist$ which implies $|B - A| > dist$.

Also, K' satisfies CCS property (1) because

for every axis x, $|B_x - A_x| < dist$ where $dist$ will be some $largest\_interval\_remaining$; so, $|B_x - A_x| < r_x dist$ where $r_x dist$ will be the $second\_largest\_interval\_remaining$. Summing

$\sum |B_x - A_x| < \sum r_x dist$

$|B - A| < dist$ ■

But the inequality $\sum r_x < 1$ is reasonable to expect at some time as we reduce distance *dist* in descending order of *largest_interval_remaining*. In words, it says that if the density decreases enough on all the axes (even if, most likely, by different amounts) then we can make a cluster for that distance that is a complete solution.

Note that when we include a new M-dim point the maximum distance between nearest neighbors in the same cluster might decrease along one or more axes and thus in M-dim space as well. Yet the cluster that contains point A will still obey property (1). Similarly, the minimum distance between two clusters may increase along one or more axes and thus increase in M-dim space. So the clusters will still obey CCS property (2).

Algorithm 2 is in the pseudo-code. This approach is a complete solution that always satisfies CCS properties (1) and (2). The algorithm is $O(MN \log N)$.

6. Pseudo-code for multi-dimensional data points

Now, we show the python pseudo-code for global density clustering in M-dimensions and M axes axes[0],... axes[$M-1$]. Note how it builds on the solution in 1-dimension axis for each axis. See Algorithm 2.

When implemented efficiently, this algorithm is $O(MN \log N)$.

7. Future investigations

7.1. Noisy data

Noisy data is particularly problematic for many algorithms including GDC. Distinguishing outliers from noise is very difficult. Another example of a problem is where noise causes two clusters that should be separate to be confused with a single cluster. DBMAC and DBMAC II Zhang and Yuan (2018) approaches the problem of a noisy background (even 80%) with multi-scale analysis. GDC considers an alternative to multi-scale analysis.

When GDC projects the dataset onto each 1-dimensional axis, GDC can insert code to deal with noise in 1-dim space. The code to deal with noise removal in 1-dim space is Fast Fourier Transforms (FFT). In particular, Algorithm 3.

FFT is $O(N \log N)$ time complexity so the impact on runtime depends on how many axes, one or more, the above steps are done.

7.2. Streaming data

GDC handles streaming data updates by collecting as much of the new data being streamed into memory of a single computer where it can sort it fast. Then GDC merges the sorted updates into the existing data which is also sorted. This is simply another merge iteration of the mergesort used by GDC. During this merge, nearest neighbor distances are updated so that the top-down processing of defining clusters can be repeated. But since the sorting is the reason that time complexity is $O(MN \log N)$ and it is done as just mentioned, it will be practical for GDC to repeat defining clusters based on the new information of nearest neighbors. In this way, streaming data fits naturally into the GDC algorithm.

Algorithm 2: The CCS in M-dimensions

1. Initialize the set of clusters to be just a single cluster C_1 that contains all points in the dataset. Set $num\_clusters$=1; $sum\_all\_second\_largest$=0; $dist$=infinity.
2. For $axis\_ndx$ in range(M):
   (a) Say the current set of all clusters is C_1, C_2,...C_i, ...$C_{num\_clusters}$
   (b) Get all clusters on the line axes [$axis\_ndx$] for distance $dist$ that is the largest interval remaining using the 1-dimension solution of the current axis.
   Say the clusters are $D_1, D_2, \ldots D_{n'}$ where n' is defined by
   $largest\_interval\_remaining[axis\_ndx][n'–1] = dist$
   $dist > largest\_interval\_remaining[axis\_ndx][n']$
   (c) $sum\_all\_second\_largest+ = largest\_interval\_remaining[axis\_ndx][n']$
   which happens to be the second largest interval remaining on the current axis.
   (d) Get the set of M-dim points $D'_i = inverse(D_i)$
   Note that $\cup D'_i$ is the entire dataset.
   (e) Replace each cluster C_i with the group of clusters $C_i \cap D'_1 \ldots, C_i \cap D'_{n'}$
   (f) $num\_clusters+ = (n' - 1)$
   (g) reduce $dist$ to $dist = largest\_interval\_array[axis\_ndx]][n']$
3. if $sum\_all\_second\_largest < dist$:
   (a) # Since the clusters satisfy CCS properties (1) and (2), ask the user whether they want to quit or refine the resolution by reducing $dist$.
4. Reduce $dist$ to $largest\_interval\_remaining[axis\_ndx][n']$ and repeat the above loop until either this inequality holds or we decide there is no clear-cut further division into clusters.

Algorithm 3: FFT handling noisy data

1. FFT the 1-dim projections of the dataset,
2. Remove higher frequencies just as if denoising an audio signal
3. Do the inverse FFT back from frequency space to the 1-dim projection
4. Remove the data points in the 1-dim projection that are not in the result of the inverse FFT
5. Remove the M-dimensional points that projected onto those points in step 4.
6. Continue with GDC processing.

7.3. Different datasets

We will investigate how well global density clustering does with MNIST images of digits: (1) suggesting the correct number of clusters, ten for the digits; (2) accurately predicting the label of a test digit by seeing which cluster we assign the "test" digit to and seeing the label that we assigned to that cluster during the "training" phase. The label assigned to that cluster was the plurality of the labels of all the digits in that cluster during the "training" phase. We then do the same with k-means and DBSCAN and compare the results. We can even compare the results to the accuracy of supervised learning models such as kNN, XGBOOST, etc.

An important action is to try different datasets. MNIST is the "hello world" dataset for machine learning algorithms but clustering has more practical uses such as in market segmentation. Such datasets would provide real-world use and comparison of the global density algorithm.

7.4. Compare with all nearest neighbor algorithms

Another important comparison is against other "all nearest neighbor" algorithms in addition to the above comparison with k-means and DBSCAN. For example, Sankaranarayanan and Samet (2007) gives an all nearest neighbor algorithm that is orders of magnitude faster than alternatives. It is important to see how well global density does in comparison especially on big data.

7.5. Final notes

Of course, there are many other directions to explore. For example, exploring norms other than L1 would be nice. More important is exploring Hamming distance for categorical clustering. An important study will be the impact of GDC ideas on other cluster algorithms and cluster methods (a distinction from Zhang and Yuan (2018) that is only concerned with all nearest neighbor problems). It will be informative to see how it fits in with the clustering of cluster algorithms by their clustering of 12 datasets. See section 3.5 in Jain (2010).

But most important is that future investigation must try clustering big data to analyze its performance. We would learn the limits of M and N for not being prohibitively costly in runtime performance. Large scale clustering is where we cluster millions of data points with thousands of features. Gene clustering, clustering of earth science data and other extremely valuable applications have an M of order 100 and an N of order 100,000 and requires support for large scale clustering. See section 4.3 in Jain (2010). This is where GDC may have a particularly significant impact.

8. Citations and Bibliography

Acknowledgments

I wish to thank my family for their support.

References

Y. Tian D. Xu. A comprehensive survey of clustering algorithms. *Ann. Data. Sci*, 2:165–193, 2015.

M. Ester E. Schubert, J. Sander and H. Kriegel. Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Transactions on Database Systemsi*, 42:3, 2017.

A. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31:8: 651–666, June 2010.

J. Kleinberg. An impossibility theorem for clustering. *NIPS MIT Press*, 15:463–470, 2002.

J. Sankaranarayanan and H. Samet. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics*, 2:157–174, 2007.

T. Zhang and B. Yuan. Density-based multiscale analysis for clustering in strong noise settings with varying densities. *IEEE Access*, 6:25861–25873, March 2018.