Evolution of Clustering Algorithms and HDBSCAN

Introduction

The goal of this paper is to discuss different clustering techniques and the problem on implementing this in practice and how Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) can help us to provide near perfect clustering techniques. This paper will also discuss at high level internal algorithms followed in HDBSCAN clustering

What is Clustering?

In machine learning, Clustering is an unsupervised task. It automatically discovers natural grouping in data. In short, a cluster is a group of similar data points. In vector space, similarity means how close one data point is with each other. Similarity of two data points can be calculated through different techniques cosine distance or Euclidean distance

Different Types of Clustering

The following three clustering technique are most popular techniques

K-Means

K-means clustering is one of the popular and simple unsupervised machine learning algorithms. This is the way this algorithm works. On this algorithm, we first choose Hyperparameter i.e the number of cluster you want (K) . Start with K random datapoints allocating each of them to one cluster each and initially they are the centroid of the cluster. We try to allocate rest of the datapoints to one of the K clusters and in the process of doing so, we move the Centroids based on the data allocated. The algorithm ends when the movement of the centroid is minimal. Main demerits of this clustering is that K-value is constant and in some use cases this is a big bottleneck. Time complexity of this algorithm is $O(N^2)$ [1]

Hierarchical Clustering

This is clustering method which builds a hierarchy of clusters. There are two strategies followed mainly [4]

- **Agglomerative** : This is a bottom up approach where each data points starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. This type of clustering takes decisions by considering the local patterns without initially considering the global distribution of data. Here is the high level steps followed:
    1. Consider Each data points as a cluster
    2. Calculate the distance/similarity of one cluster with all the other clusters
    3. Merge the clusters which are highly similar or close to each other.
    4. Recalculate the proximity matrix for each cluster
    5. Repeat Step 3 and 4 until only a single cluster remains.
    6. Cut the tree at a certain level gives a set of clusters

- **Divisive**: This is a top-down approach where all data points start in one cluster, and splits are performed recursively as one moves down the hierarchy. This type of clustering initially takes decisions by global distribution of data. Here, we separate the data points from the clusters which aren't comparable. In the end, we are left with N clusters i.e this is completely opposite way on how Agglomerative clustering works

Time complexity for this type of clustering is $O(N^3)$. This kind of cluster analysis is essentially an exploratory approach; the interpretation of the resulting hierarchical structure is context-dependent and often several solutions are equally good from a theoretical point of view. Cluster composition completely depends on at

what level you are cutting the tree. This also fails to cluster separate local density based datapoints into a single separate cluster.

DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is one of the most popular clustering technique. This is a non-parametric algorithm i.e given a set of points in some space, it groups together points that are closely packed together and form a cluster and the points which are from non-dense region are considered as Noise or outlier points. [2]

- Assume the number of minimum points (minPts, a hyperparameter) to consider for defining the dense region
- Find the points in the ε (eps, another hyperparameter) neighbourhood of every point, and identify the core points with more than minPts neighbours.
- Find the connected components of core points on the neighbour graph, ignoring all non-core points.
- Assign each non-core point to a nearby cluster if the cluster is an ε (eps) neighbour, otherwise assign it to noise.

Time complexity of this Algorithm is O(n log n)

Cons:

- DBSCAN not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster
- It cannot cluster data sets well with large differences in densities, since the minPts-ε combination cannot then be chosen appropriately for all clusters[3]

Why HDBSCAN?

While DBSCAN needs a minimum cluster size and a distance threshold epsilon as user-defined input parameters, HDBSCAN is basically a DBSCAN implementation for varying epsilon values and therefore only needs the minimum cluster size as single input parameter.[5]
In particular, let us consider a large data set distributed such that there is a high number of very dense objects in some areas, and only few objects in other areas. If we were only interested in the highly populated areas, HDBSCAN would give us good results for a minPts value large enough to declare sparse regions as noise and dense regions as clusters. In some cases, however, we do not want all observations in sparse environments to be marked as noise: these areas might naturally contain fewer objects, but small yet dense groups of objects that do exist might be just as relevant as the ones in regions with lots of data.[6]

How HDBSCAN Algorithm works?

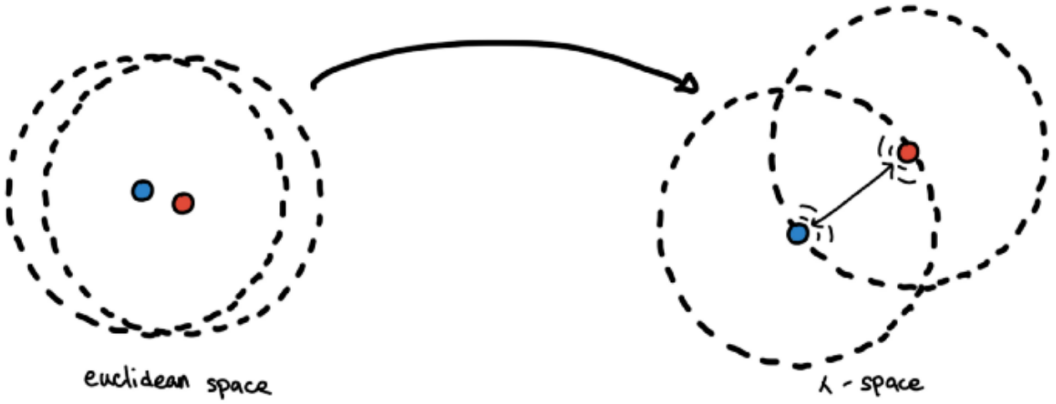
A rough sketch of the HDBSCAN's implementation goes as follows: [8]

1. Given a dataset X, compute the **core distances** of all the data objects in X.
2. Use the **mutual_reachability(a, b)** as a distance metric for each a, b
3. Compute a **Minimum Spanning Tree** (MST) of the Mutual Reachability Graph G_{mreach}
4. Extend the MST to obtain MST_{ext}, by adding a "self loop edge" for each vertex with weight equal to that of its core distance
5. Extract the HDBSCAN hierarchy as a dendrogram from MST_{ext}.
    a. All the objects are assigned to the same label, thus forming the root of the tree.
    b. Iteratively remove all edges from MST_{ext} in decreasing order of weights.
        i. Edges with the same weight are removed simultaneously.
        ii. After removal of an edge, labels are assigned to the connected components that contain one vertex of the removed edge. A new cluster label is assigned if the component has at least one edge in it, else the objects are assigned a null label, indicating it to be a noise object.

Unlike DBSCAN , here instead of setting ε then counting the neighbours, we determine the number of neighbours we want and find the smallest value of ε that would contain these K neighbours. These ε is called the core distance of a point. Based on this core distance mutual reachability distance is calculated as follows

```
mutual_reachability_distance(a, b) = max(
   core_distance(a),
   core_distance(b),
   distance(a, b)
)
```

This is a major difference on HDBSCAN on how the distance is calculated between 2 datapoints as compared to other clustering algorithm where we mainly use Euclidian or cosine distance to measure the distance between the two. The following diagram depicts it as below [7]



Due to the randomness of a random sample, two points can be close to each other in a very sparse region. However, we expect points in sparse regions to be far apart from each other. By using the mutual reachability distance, points in sparse regions "repel other points" if they are too close to it, while points in very dense regions are unaffected.
Time complexity of this clustering algorithm is $O(N^2/2)$

Conclusion

Which clustering technique is well suited is completely depend on the use case you want to solve. In case you need fixed number of clusters , **K-means** clustering might be a good choice since this is easy to implement. In case you would like to cluster based on certain distance threshold, hierarchy clustering is suitable for you although this has significant CPU overhead. DBSCAN is suitable on most of the clustering use cases since the clustering is based on the density distribution of the datapoints but it might create a single BIG cluster (*single-linkage* effect) in case the most of the data is densed. HDBSCAN is an extension of DBSCAN only which addresses the limitation of DBSCAN and can create multiple small clusters from a densed distribution although this is a bit difficult to implement following its algorithm. Python has support for HDBSCAN Clustering through sklearn library [9]

Reference

1. Garbade, M. J. (2018, September 13). Understanding K-means Clustering in Machine Learning. Medium. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1
2. Maimon, Oded; Rokach, Lior (2006). "Clustering methods". Data Mining and Knowledge Discovery Handbook. Springer. pp. 321–352. ISBN 978-0-387-25465-4.
3. Schubert, Erich; Sander, Jörg; Ester, Martin; Kriegel, Hans Peter; Xu, Xiaowei (July 2017). "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN". ACM Trans. Database Syst. 42 (3): 19:1–19:21. doi:10.1145/3068335. ISSN 0362-5915. S2CID 5156876.
4. Kriegel, Hans-Peter; Kröger, Peer; Sander, Jörg; Zimek, Arthur (2011). "Density-based Clustering". WIREs Data Mining and Knowledge Discovery. 1 (3): 231–240. doi:10.1002/widm.30.
5. "Combining HDBSCAN* with DBSCAN¶." *Combining HDBSCAN* with DBSCAN - Hdbscan 0.8.1 Documentation*, https://hdbscan.readthedocs.io/en/latest/how_to_use_epsilon.html.

6. Malzer, Claudia, and Marcus Baum. “A Hybrid Approach to Hierarchical Density-Based Cluster Selection.” ArXiv.org, 21 Jan. 2021, https://arxiv.org/abs/1911.02282.
7. Berba, P. (2020, January 17). Understanding HDBSCAN and Density-Based Clustering. Pepe Berba. https://pberba.github.io/stats/2020/01/17/hdbscan/
8. Syed, T. I. (2015). Parallelization of Hierarchical. . . ERA. https://era.library.ualberta.ca/items/ae1d254b-c62b-4dda-a178-b1e284e64dfd
9. Basic Usage of HDBSCAN* for Clustering — hdbscan 0.8.1 documentation. Basic Usage of HDBSCAN* for Clustering. Retrieved 30 October 2021, from https://hdbscan.readthedocs.io/en/latest/basic_hdbscan.html