Density-Based Spatial Clustering Of Applications With Noise(DBSCAN) Clustering

-By Suman Chatterjee.

Indian Institute of Technology(ISM) Dhanbad.

UNDER THE GUIDANCE OF

Dr. CHANDRASHEKHAR RAO

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY(INDIAN SCHOOL OF MINES),

DHANBAD.

DHANBAD-826004.

ACKNOWLEDGEMENT

I take this opportunity to express my deep sense of gratitude and respect towards my project guide,

Dr. ACS Rao, Assistant Professor, Department of Computer Science And Engineering, IIT(ISM) DHANBAD.

I am very much indebted to him for the generosity, expertise and guidance I have received from him while working on the project.

I wish to express my profound gratitude to Dr. P.K Jana, Head of Department Of Computer Science And Engineering , IIT(ISM) DHANBAD for his guidance and inspiration.

DATE-10/04/2018

NAME-SUMAN CHATTERJEE.

ADMISSION NO.-15JE001400.

Abstract

In 1972, Robert F. Ling had already published a paper titled "The Theory and Construction of k-Clusters" in The Computer Journal with a very closely related, but with an estimated runtime complexity of $O(n^3)$, whereas DBSCAN has a worst-case of $O(n^2)$, and the database-oriented range-query formulation of DBSCAN allows for index acceleration. The algorithms differ in their handling of border points.

Density-based spatial clustering of applications with noise (**DBSCAN**) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996. It is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

In 2014, the algorithm was awarded the test of time award (an award given to algorithms which have received substantial attention in theory and practice) at the leading data mining conference, KDD.

INTRODUCTION

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular clustering algorithm used as an alternative to K-means in predictive analytics. It doesn't require that you input the number of clusters in order to run. But in exchange, you have to tune two other parameters.

The scikit-learn implementation provides a default for the eps and min_samples parameters, but you're generally expected to tune those. The eps parameter is the maximum distance between two data points to be considered in the same neighborhood. The min_samples parameter is the minimum amount of data points in a neighborhood to be considered a cluster.

One advantage that DBSCAN has over K-means is that DBSCAN is not restricted to a set number of clusters during initialization. The algorithm will determine a number of clusters based on the density of a region.

Keep in mind, however, that the algorithm depends on the eps and min_samples parameters to figure out what the density of each cluster should be. The thinking is that these two parameters are much easier to choose for some clustering problems.

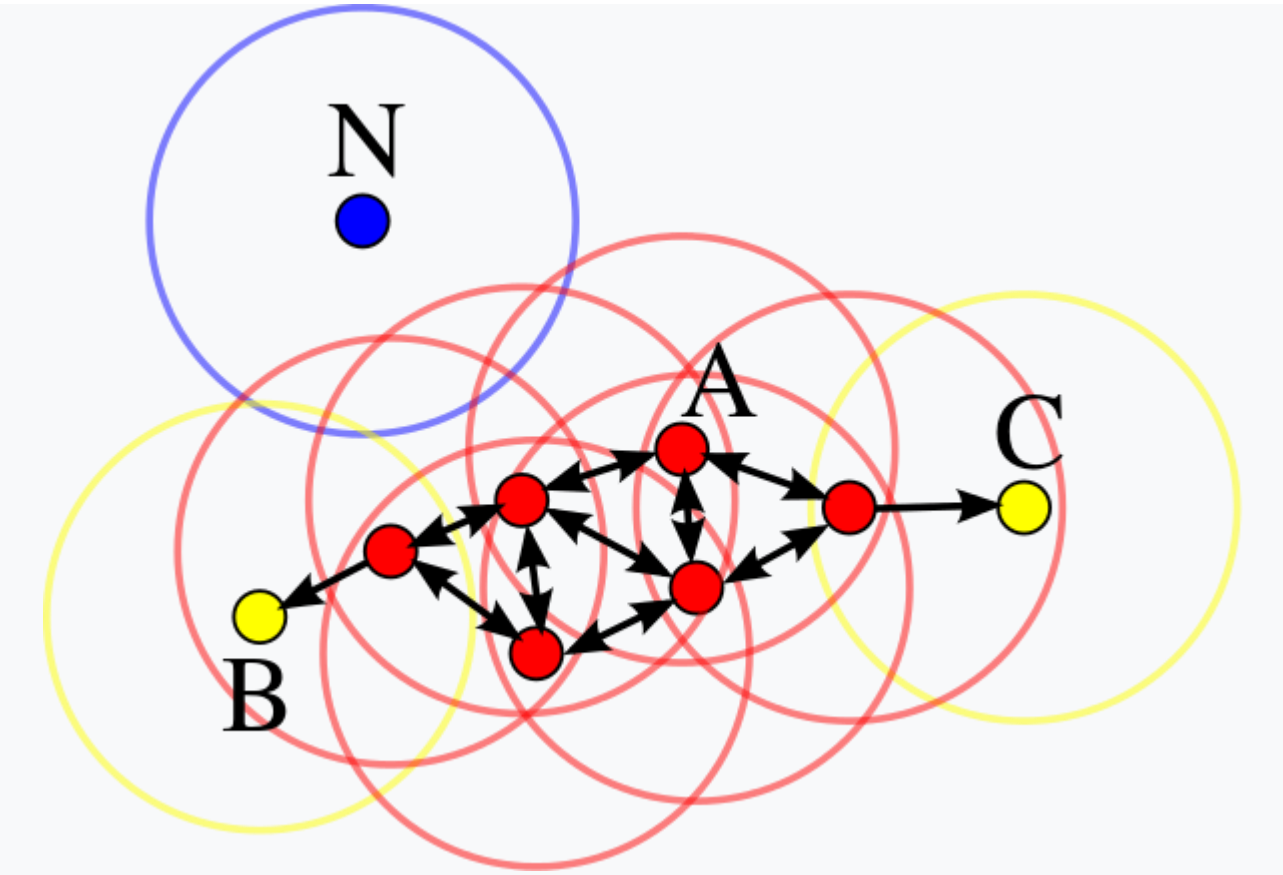
Because the DBSCAN algorithm has a built-in concept of noise, it's commonly used to detect outliers in the data — for example, fraudulent activity in credit cards, e-commerce, or insurance claims.

2.Preliminary

Consider a set of points in some space to be clustered. For the purpose of DBSCAN clustering, the points are classified as *core points*, (*density-*)*reachable points* and *outliers*, as follows:

- A point p is a core point if at least minPts points are within distance ε (ε is the maximum radius of the neighborhood from p) of it (including p). Those points are said to be *directly reachable* from p.
- A point q is directly reachable from p if point q is within distance ε from point p and p must be a core point.
- A point q is reachable from p if there is a path p_1, ..., p_n with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i (all the points on the path must be core points, with the possible exception of q).
- All points not reachable from any other point are outliers.

Now if p is a core point, then it forms a *cluster* together with all points (core or non-core) that are reachable from it. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge", since they cannot be used to reach more points.



In this diagram, $\text{minPts} = 4$. Point A and the other red points are core points, because the area surrounding these points in an ε radius contain at least 4 points (including the point itself). Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core

points) and thus belong to the cluster as well. Point N is a noise point that is neither a core point nor directly-reachable.

Reachability is not a symmetric relation since, by definition, no point may be reachable from a non-core point, regardless of distance (so a non-core point may be reachable, but nothing can be reached from it). Therefore, a further notion of *connectedness* is needed to formally define the extent of the clusters found by DBSCAN. Two points p and q are density-connected if there is a point o such that both p and q are reachable from o. Density-connectedness *is* symmetric.

A cluster then satisfies two properties:

1. All points within the cluster are mutually density-connected.
2. If a point is density-reachable from any point of the cluster, it is part of the cluster as well.

3. Algorithm:

Original Query-based Algorithm

DBSCAN requires two parameters: ε (eps) and the minimum number of points required to form a dense region (minPts). It starts with an arbitrary starting point that has not been visited. This point's ε-neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. Note that this point might later be found in a sufficiently sized ε-environment of a different point and hence be made part of a cluster.

If a point is found to be a dense part of a cluster, its ε-neighborhood is also part of that cluster. Hence, all points that are found within the ε-neighborhood are added, as is their own ε-neighborhood when they are also dense. This process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.

DBSCAN can be used with any distance function (as well as similarity functions or other predicates). The distance function (dist) can therefore be seen as an additional parameter.

The algorithm can be expressed in pseudocode as follows:

```
DBSCAN(DB, dist, eps, minPts) {
   C = 0                                                  /* Cluster counter */
   for each point P in database DB {
      if label(P) ≠ undefined then continue               /* Previously
processed in inner loop */
      Neighbors N = RangeQuery(DB, dist, P, eps)          /* Find neighbors */
      if |N| < minPts then {                              /* Density check */
         label(P) = Noise                                 /* Label as Noise */
         continue
      }
      C = C + 1                                           /* next cluster label
*/
      label(P) = C                                        /* Label initial point
*/
```

```
      Seed set S = N \ {P}                              /* Neighbors to expand
*/
      for each point Q in S {                           /* Process every seed
point */
         if label(Q) = Noise then label(Q) = C          /* Change Noise to
border point */
         if label(Q) ≠ undefined then continue          /* Previously
processed */
         label(Q) = C                                   /* Label neighbor */
         Neighbors N = RangeQuery(DB, dist, Q, eps)     /* Find neighbors */
         if |N| ≥ minPts then {                         /* Density check */
            S = S ∪ N                                   /* Add new neighbors
to seed set */
         }
      }
   }
}
```

where RangeQuery can be implemented using a database index for better performance, or using a slow linear scan:

```
RangeQuery(DB, dist, Q, eps) {
   Neighbors = empty list
   for each point P in database DB {                 /* Scan all points in
the database */
      if dist(Q, P) ≤ eps then {                     /* Compute distance
and check epsilon */
         Neighbors = Neighbors ∪ {P}                 /* Add to result */
      }
   }
   return Neighbors
}
```

Abstract Algorithm

The DBSCAN algorithm can be abstracted into the following steps:

1. Find the ε (eps) neighbors of every point, and identify the core points with more than minPts neighbors.
2. Find the connected components of *core* points on the neighbor graph, ignoring all non-core points.
3. Assign each non-core point to a nearby cluster if the cluster is an ε (eps) neighbor, otherwise assign it to noise.

A naive implementation of this requires storing the neighborhoods in step 1, thus requiring substantial memory. The original DBSCAN algorithm does not require this by performing these steps for one point at a time.

4. Complexity

DBSCAN visits each point of the database, possibly multiple times (e.g., as candidates to different clusters). For practical considerations, however, the time complexity is mostly governed by the number of regionQuery invocations. DBSCAN executes exactly one such query for each point, and if an indexing structure is used that executes a neighborhood query in $O(\log n)$, an overall average runtime complexity of $O(n \log n)$ is obtained (if parameter ε is chosen in a meaningful way, i.e. such that on average only $O(\log n)$ points are returned). Without the use of an accelerating index structure, or on degenerated data (e.g. all points within a distance less than ε), the worst case run time complexity remains $O(n^2)$. The distance matrix of size $(n^2-n)/2$ can be materialized to avoid distance recomputations, but this needs $O(n^2)$memory, whereas a non-matrix based implementation of DBSCAN only needs $O(n)$ memory.

4.1. Advantages

1. DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means.
2. DBSCAN can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster. Due to the MinPts parameter, the so-called single-link effect (different clusters being connected by a thin line of points) is reduced.
3. DBSCAN has a notion of noise, and is robust to outliers.
4. DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database. (However, points sitting on the edge of two different clusters might swap cluster membership if the ordering of the points is changed, and the cluster assignment is unique only up to isomorphism.)
5. DBSCAN is designed for use with databases that can accelerate region queries, e.g. using an R* tree.
6. The parameters minPts and ε can be set by a domain expert, if the data are well understood.

4.1. Disadvantages

1. DBSCAN is not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster, depending on the order the data are processed. For most data sets and domains, this situation fortunately does not arise often and has little impact on the clustering result: both on core points and noise points, DBSCAN is deterministic. DBSCAN* is a variation that treats border points as noise, and this way achieves a fully deterministic result as well as a more consistent statistical interpretation of density-connected components.
2. The quality of DBSCAN depends on the distance measure used in the function regionQuery(P,ε). The most common distance metric used is Euclidian distance. Especially for high dimensional data, this metric can be rendered almost useless due to the so-called "Curse of Dimensionality", making it difficult to find an appropriate value for ε. This effect, however, is also present in any other algorithm based on Euclidean distance.
3. DBSCAN cannot cluster data sets well with large differences in densities, since the minPts-ε combination cannot then be chosen appropriately for all clusters.[8]
4. If the data and scale are not well understood, choosing a meaningful distance threshold ε can be difficult.

5. Parameter Estimation

Every data mining task has the problem of parameters. Every parameter influences the algorithm in specific ways. For DBSCAN, the parameters ε and *minPts* are needed. The parameters must be specified by the user. Ideally, the value of ε is given by the problem to solve (e.g. a physical distance), and *minPts* is then the desired minimum cluster size.

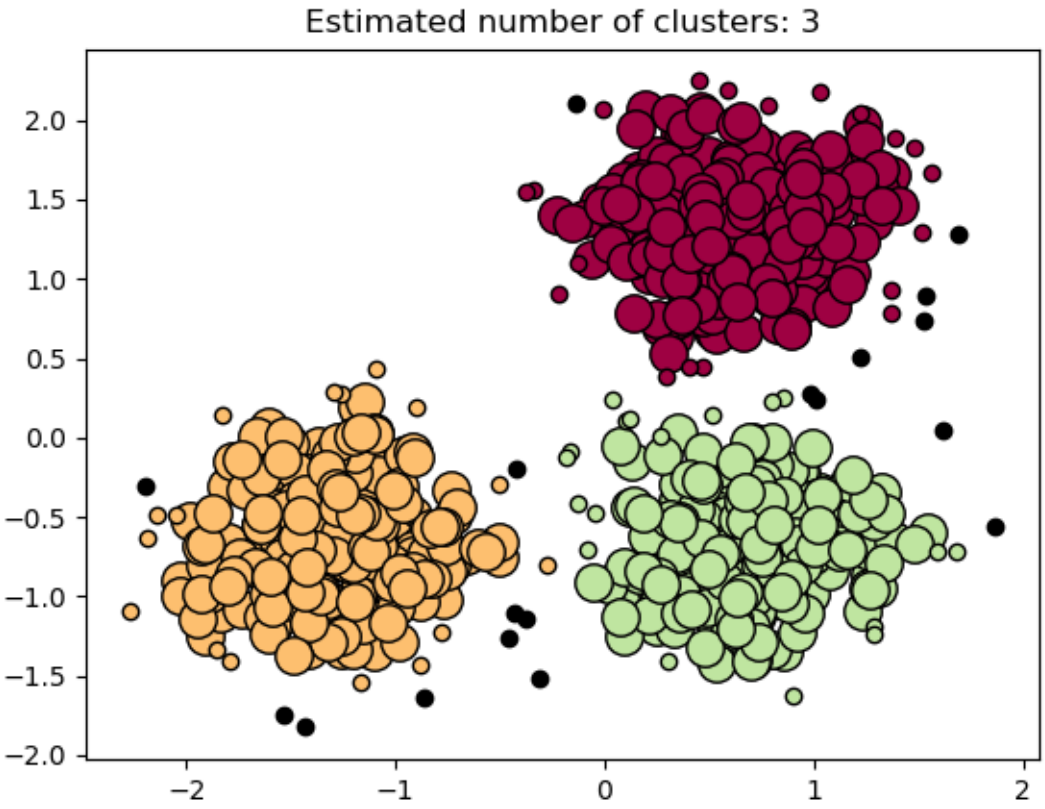
- *MinPts*: As a rule of thumb, a minimum *minPts* can be derived from the number of dimensions D in the data set, as $minPts \geq D + 1$. The low value of $minPts = 1$ does not make sense, as then every point on its own will already be a cluster. With $minPts \leq 2$, the result will be the same as of hierarchical clustering with the single link metric, with the dendrogram cut at height ε. Therefore, *minPts* must be chosen at least 3. However, larger values are usually better for data sets with noise and will yield more significant clusters. As a rule of thumb, $minPts = 2 \cdot dim$ can be used, but it may be necessary to choose larger values for very large data, for noisy data or for data that contains many duplicates.
- ε: The value for ε can then be chosen by using a k-distance graph , plotting the distance to the $k = minPts-1$ nearest neighbor ordered from the largest to the smallest value. Good values of ε are where this plot shows an "elbow": if ε is chosen much too small, a large part of the data will not be clustered; whereas for a too high value of ε, clusters will merge and the majority of objects will be in the same cluster. In general, small values of ε are preferable, and as a rule of thumb only a small fraction of points should be within this distance of each other. Alternatively, an OPTICS plot can be used to choose ε, but then the OPTICS algorithm itself can be used to cluster the data.
- Distance function: The choice of distance function is tightly coupled to the choice of ε, and has a major impact on the results. In general, it will be necessary to first identify a reasonable measure of similarity for the data set, before the parameter ε can be chosen. There is no estimation for this parameter, but the distance functions needs to be chosen appropriately for the data set. For example, on geographic data, the great circle distance is often a good choice.

OPTICS can be seen as a generalization of DBSCAN that replaces the ε parameter with a maximum value that mostly affects performance. *MinPts* then essentially becomes the minimum cluster size to find. While the algorithm is much easier to parameterize than DBSCAN, the results are a bit more difficult to use, as it will usually produce a hierarchical clustering instead of the simple data partitioning that DBSCAN produces.

Recently, one of the original authors of DBSCAN has revisited DBSCAN and OPTICS, and published a refined version of hierarchical DBSCAN (HDBSCAN*),which no longer has the notion of border points. Instead, only the core points form the cluster.

6. Results

Finds core samples of high density and expands clusters from them.




```
Estimated number of clusters: 3
Homogeneity: 0.953
Completeness: 0.883
V-measure: 0.917
Adjusted Rand Index: 0.952
Adjusted Mutual Information: 0.883
Silhouette Coefficient: 0.626
```

Parameters:

eps : float, optional

The maximum distance between two samples for them to be considered as in the same neighborhood.

min_samples : int, optional

The number of samples (or total weight) in a neighborhood for a point to be

considered as a core point. This includes the point itself.

metric : string, or callable

The metric to use when calculating distance between instances in a feature array. If metric is a string or callable, it must be one of the options allowed by metrics.pairwise.calculate_distance for its metric parameter. If metric is “precomputed”, X is assumed to be a distance matrix and must be square. X may be a sparse matrix, in which case only “nonzero” elements may be considered neighbors for DBSCAN.

New in version 0.17: metric *precomputed* to accept precomputed sparse matrix.

metric_params : dict, optional

Additional keyword arguments for the metric function.

New in version 0.19.

algorithm : {‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’}, optional

The algorithm to be used by the NearestNeighbors module to compute pointwise distances and find nearest neighbors. See NearestNeighbors module documentation for details.

leaf_size : int, optional (default = 30)

Leaf size passed to BallTree or cKDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : float, optional

The power of the Minkowski metric to be used to calculate distance between points.

n_jobs : int, optional (default = 1)

The number of parallel jobs to run. If `-1`, then the number of jobs is set to the number of CPU cores.

Attributes:

core_sample_indices_ : array, shape = [n_core_samples]

Indices of core samples.

components_ : array, shape = [n_core_samples, n_features]

Copy of each core sample found by training.

labels_ : array, shape = [n_samples]

Cluster labels for each point in the dataset given to fit(). Noisy samples are given the label -1.

Methods

fit(X[, y, sample_weight])	Perform DBSCAN clustering from features or distance matrix.
fit_predict(X[, y, sample_weight])	Performs clustering on X and returns cluster labels.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.

Perform DBSCAN clustering from features or distance matrix.

Parameters:

X : array or sparse (CSR) matrix of shape (n_samples, n_features), or array of shape (n_samples, n_samples)

A feature array, or array of distances between samples if `metric='precomputed'`.

sample_weight : array, shape (n_samples,), optional

Weight of each sample, such that a sample with a weight of at least `min_samples` is by itself a core sample; a sample with negative weight may inhibit its eps-neighbor from being core. Note that weights are absolute, and default to 1.

y : Ignored

fit_predict(*X*, *y=None*, *sample_weight=None*)

Performs clustering on X and returns cluster labels.

Parameters:	**X** : array or sparse (CSR) matrix of shape (n_samples, n_features), or array of shape (n_samples, n_samples) A feature array, or array of distances between samples if `metric='precomputed'`. **sample_weight** : array, shape (n_samples,), optional Weight of each sample, such that a sample with a weight of at least `min_samples` is by itself a core sample; a sample with negative weight may inhibit its eps-neighbor from being core. Note that weights are absolute, and default to 1. **y** : Ignored
Returns:	**y** : ndarray, shape (n_samples,) cluster labels

get_params(*deep=True)*

Get parameters for this estimator.

Parameters:	**deep** : boolean, optional If True, will return the parameters for this estimator and contained subobjects that are estimators.
Returns:	**params** : mapping of string to any Parameter names mapped to their values.

set_params(***params*)

Set the parameters of this estimator.The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Returns:	**self** :

7. Reference/Bibliography

1. *Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M., eds. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). pp. 226–231.*
2. http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html