ENHANCED DBSCAN ALGORITHM

Priyamvada Paliwal¹, Meghna Sharma*²

¹Software Engineering, ITM University
Sector 23-A, Gurgaon, India
²Asst. Prof. Dept. of CS, ITM University
Sector 23-A, Gurgaon, India

Abstract— The DBSCAN algorithm can identify clusters in large spatial data sets by looking at the local density of database elements, using only one input parameter. This paper presents a comprehensive study of DBSCAN algorithm and the enhanced version of DBSCAN algorithm. The salient of this paper to present enhanced DBSCAN algorithm with its implementation with the complexity and the difference between the older version of DBSCAN algorithm. And there are also additional features described with this algorithm for finding outliers.

Keywords— Density-based spatial clustering of applications with noise, Threshold Distance, Minimum Points, Cluster, Density.

I. INTRODUCTION

DBSCAN (for density-based spatial clustering of applications with noise) is a data clustering algorithm. It is a density-based clustering algorithm because it finds a number of clusters starting from the estimated density distribution of corresponding nodes. DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

II. DBSCAN ALGORITHM

Clustering algorithm that can do everything that DBSCAN can do is not yet available. Various new clustering algorithms appear occasionally. DBSCAN has been modified to great extent recently and used to derive a new procedure to calculate EPS (threshold distance) which are most important parameters from the section [5]. The density-based clustering algorithm presented is different from the classical Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and has the following advantages: first, Greedy algorithm substitutes for R*-tree in DBSCAN (Density based spatial clustering of application with noise) to index the clustering space so that the clustering time cost is decreased to great extent and I/O memory load is reduced as well; second, the merging condition to approach to arbitrary-shaped clusters from [12] is designed carefully so that a single threshold can distinguish correctly all clusters in a large spatial dataset though some density-skewed clusters live in it from the outliers in [4]. DBSCAN can find clusters of arbitrary shape, as can be seen in fig 1 [6]. However, clusters that lie close to each other tend to belong to the same class.

A. Complexity

DBSCAN visits each point of the database, possibly multiple times (e.g., as candidates to different clusters) from the section [8]. For practical considerations, however, the time complexity is mostly governed by the number of region Query invocations [6]. DBSCAN executes exactly one such query for each point, and if an indexing structure is used from [12] that executes such a neighbourhood query in O(logn), an overall runtime complexity of O(n.logn) is obtained.

B. Implementation

Follow the below given path to run the code (i.e. netbeans):
1) Create a project name "DBSCAN1"
2) Under which create a package name as "dbscan",
3) It will show now path DBscan1-->src-->dbscan & now create a class as same name as above given java files.
4) Under dbscan package create 5 classes --
dbscan.java, Gui.java, Point.java Utility.java, DB.java
5) After saving all & try to run GUI.java where main() class is defined.

This section will focus on three primary components
- CODE
- SCREENSHOTS OF DBSCAN ALGORITHM

B.1 Code

```java
for(Point f : hset){
if (Utility.equalPoints(f, np)) {
```
Y = true;
break;
} else
{Y = false;}
}
if (!Y) {
hset.add(np);
status.setText("Point " + x1 + "," + y1 + " Added");
status.setForeground(Color.BLUE);
counter.setText("Number of Points - " + Integer.toString(hset.size()));
tfx.setText(null);
tfy.setText(null);
} else {
if (Y) {
status.setText("Point " + x1 + "," + y1 + " Already Exists");
tfx.setText(null);
tfy.setText(null);
status.setForeground(Color.BLACK);
} else {
status.setText("Wrong Input");
status.setForeground(Color.RED);
tfx.setText(null);
tfy.setText(null);

**III. ENHANCED DBSCAN ALGORITHM**

The key idea of the DBSCAN algorithm is that, for each point of a cluster, the neighbourhood of a given radius has to contain at least a minimum number of points, that is, the density in the neighbourhood has to exceed some predefined threshold from [6]. This algorithm needs three input parameters:
- k, the neighbour list size;
- Eps, the radius that delimitate the neighbourhood area of a point (Epsneighbourhood);
- MinPts, the minimum number of points that must exist in the Eps-neighbourhood.

This method is highly dependent on parameter provided by the users and computationally expensive when applied unbounded data set. With the development of information technologies, the number of databases, as well as their dimensions and complexity grow rapidly. With high dimensional dataset calculate distance with each instances will increase the computational cost.

Pairwise distance computes the Euclidean distance between pairs of objects in n-by-p data matrix X. Rows of X correspond to observations; columns correspond to variables.
y is a row vector of length n(n–1)/2, corresponding to pairs of observations in X. The distances are arranged in the order (2,1), (3,1), ..., (n,1), (3,2), ..., (n,2), ..., (n,n–1)). y is commonly used as a dissimilarity matrix in clustering or multidimensional scaling.

Euclidean distance.
\[
dist((x, y), (a, b)) = \sqrt{(x - a)^2 + (y - b)^2}
\]

1) Take minimum points and threshold distance from user.
2) Calculate pairwise distance that is computing the Euclidean distance between pairs of object.
3) Take square distance. Calculate maximum values from square distance values.
4) According from the values of dataset, the different clusters with varied density is formed.

The parameters of the algorithm form of data points from the RFID dataset i.e. Tag id, Data id, Time in, Time out:
This section will focus on three main primary components
- IMPLEMENTATION
- CODE
- COMPLEXITY
- SCREENSHOTS

A. Implementation
This implementation was adapted to deal with datasets consisting of lists of POIs. However, as the original algorithm, our implementation needs the same two input parameters: Eps and MinPts.

Follow the below given path to run the code (i.e. netbeans):

1) Create a project name "DBSCAN1"

2) Under which create a package name as "dbscan",

3) It will show now path DBscan1-->src-->dbscan & now create a class as same name as below given java files.

4) Under dbscan package create 6 classes –

dbscan.java, Gui.java, Point.java, Utility.java, DB.java, ReadExcel.java

5) After saving all & try to run GUI.java where main() class is defined.

B. Code

Point np = new Point(x1, y1)
if (a){for(Point f : hset){
  if (Utility.equalPoints(f, np)) {
    Y = true;
    break;
  } else
  {
    Y = false;
  }
if (!Y) {
  hset.add(np);
  status.setText("Point " + x1 + "," + y1 + ": Added");
  status.setForeground(Color.BLUE);
  counter.setText("Number of Points - "+
  Integer.toString(hset.size()));
  tfx.setText(null);
  tfy.setText(null);
} else {
  status.setText("Point " + x1 + "," + y1 + ": Already Exists");
  tfx.setText(null);
  tfy.setText(null);
  status.setForeground(Color.BLACK);
} }} else {
  status.setText("Wrong Input");
  status.setForeground(Color.RED);
  tfx.setText(null);
  tfy.setText(null);
}

The code which shows the index structure in this algorithm:

corrdinatMap= (HashMap)test.read();
  System.out.println(">>>1");
  if(corrdinatMap.containsKey("xCordinat"))
    xAxis=(ArrayList)corrdinatMap.get("xCordinat");
  if(corrdinatMap.containsKey("yCordinat"))
    yAxis=(ArrayList)corrdinatMap.get("yCordinat");
  System.out.println(">>>2");
}
C. Complexity

Big(O) Notation complexity analysis.

1) In this for loop says loop count as N

2) So, initialization int loop 0 , will execute only once , +1

3) Comparison i.e. loop< loop count , will execute N+1 times, N times it will result true and iteration of for loop and the last time it will result false , so total +N+1.

4) The increment statement i.e. loop ++ will execute N times i.e. : for every true output of comparison statement.

5) So the number of operations required by this loop are

5.1) \(1+(N+1)+N\) = 2N+2

5.2) So, Complexity is O(n).

5.3) /

Algorithm can be titled as linear algorithm because complexity will remain same for any value of loop count.

D. Screenshots

IV. COMPARISON OF BOTH ALGORITHMS

<table>
<thead>
<tr>
<th>DBSCAN ALGORITHM</th>
<th>ENHANCED DBSCAN ALGORITHM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) The detected clusters are only separated by the sparse region.</td>
<td>1) The detected clusters are not only separated by the sparse regions but also separated by the region having the density variation.</td>
</tr>
<tr>
<td>2) In this , only points are taken.</td>
<td>2) In this algorithm, pixels and points both are taken.</td>
</tr>
<tr>
<td>3) Without the use of an index structure , the runtime complexity is O(n2).</td>
<td>3) In this algorithm, we used an index structure and the runtime complexity is O(n) and the code has shown above in III. B.</td>
</tr>
<tr>
<td>4) The complexity of DBSCAN algorithm is high which is O(n.log n).</td>
<td>4) The complexity of DBSCAN algorithm is reduced, which is O(n).</td>
</tr>
</tbody>
</table>

TABLE 2. ALGORITHM COMPARISON

![Fig. 3 The classification of the DBSCAN algorithm.](image)

As seen in the figure 3, DBSCAN manage to classify all the clusters correctly for the three different databases.

V. ADDITIONAL FEATURES ADDED

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
Minimum Points 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.
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.)

VI. CONCLUSIONS

DBSCAN algorithm here considers only point objects but it could be extended for other spatial objects like polygons. Applications of Enhanced DBSCAN to high dimensional feature spaces should be investigated and radius generation this high dimensional data also has to be explored. It also gains to detect clusters with varied density. DBSCAN algorithm here considers only point objects but it could be extended for other spatial objects like polygons. Time complexity is high so it could be reduced. The input parameter can be determined automatically for better clustering.

ACKNOWLEDGMENT

We would like to give our sincere gratitude to our guide Ms. Meghna Sharma who guided us to pursue this topic and helped us to complete this topic. We would also like to thank her for providing us remarkable suggestions and constant encouragement. I deem it my privilege to have carried out my research work under her guidance.

REFERENCES