Overview of Outlier Detection

Outliers are patterns in a database that are dissimilar with respect to the rest of the patterns in the dataset. They may be noise, or they may be highly valuable objects, but their existence can affect the quality of results and the performance of any data mining or knowledge discovery process. The proposed method for outlier detection uses a hybrid approach. The first stage is to apply a clustering algorithm that is k-means to partition the dataset into clusters. The second stage is to find outliers from the resulting clusters using distance-based methods. The main objective of the second stage is to find objects that are far away from their cluster centroids. The proposed approach efficiently identifies outliers in the dataset. The experimental results using real datasets demonstrate that the proposed method outperforms existing methods in terms of computational cost and performance.

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

I. Introduction

Outlier detection is currently a very active area of research in the data mining community. Finding outliers in a collection of patterns is a very well-known problem in the data mining field. An outlier is a pattern which is dissimilar with respect to the rest of the patterns in the dataset. Proposed Method for outlier detection uses a hybrid approach. Purpose of the approach is to apply a clustering algorithm that is k-means to partition the dataset into clusters and then find outliers from the resulting clusters using distance-based methods. The principle of outliers finding depend on the threshold. Threshold is set by user. The main objective of the second stage is to find out the objects, which are far away from their cluster centroids. In proposed approach, two techniques are combining to efficiently find the outlier from the data set. The experimental results using real datasets demonstrate that the proposed method takes less computational cost and performs better than the distance-based method. Proposed algorithm efficiently prunes the safe cells (inliers) and saves huge number of extra calculations. Outliers may introduce skew or complexity into models of the data, making it difficult, if not impossible, to fit an accurate model to the data in a computationally feasible manner.

II. Outlier Detection With The Help Of DBSCAN Algorithm

A 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 DBSCAN(Density Based Spatial Clustering of Applications with Noise) and has the following advantages: first, Greedy algorithm substitutes for R*-tree in DBSCAN to index the clustering space so that the clustering time cost is decreased to a 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]. Finally, authors investigate a robotic navigation and test two artificial datasets by the proposed algorithm to verify its effectiveness and efficiency.

DBSCAN(D, eps, MinPts(minimum points))
C = 0
for each unvisited point P in dataset D
mark P as visited
NeighborPts = regionQuery(P, eps)
if sizeof(NeighborPts) < MinPts
mark P as NOISE
else
C = next cluster
expandCluster(P, NeighborPts, C, eps, MinPts)

for each point P' in NeighborPts
    if P' is not visited
        mark P' as visited
        NeighborPts' = regionQuery(P', eps)
        if sizeof(NeighborPts') >= MinPts
            NeighborPts = NeighborPts joined with NeighborPts'
            if P' is not yet member of any cluster
                add P' to cluster C

regionQuery(P, eps)
return all points within P's eps-neighborhood

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. DBSCAN executes exactly one such query for each point, and if an indexing structure is used from [12] that executes such a neighborhood query in $O(\log n)$, an overall runtime complexity of $O(n \log n)$ is obtained. Without the use of an accelerating index structure, the run time complexity is $O(n^2)$. Often the distance matrix of size $(n^2 - n)/2$ is materialized to avoid distance recomputations. This however also needs $O(n^2)$ memory.

III. Implementation Of Dbscan Algorithm

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.

A. This section will focus on three primary components
- CODE
- SCREENSHOTS OF DBSCAN ALGORITHM
- ADDITIONAL FEATURES

A.1 CODE
public static void insertF()
{
    String x=tfx.getText();
    String y=tfy.getText();
    JOptionPane.showMessageDialog(null,x);
    double t=0,c=0;
    if(x!=null && (!x.equals(""))){
        t= Double.parseDouble(x);
    }
    if(y!=null && (!y.equals(""))){
        c=Double.parseDouble(y);
    }
    DB d=new DB();
    Connection con=d.getConnection();
    try{
        PreparedStatement ps=con.prepareStatement("insert into define values(?,?)");
        ps.setDouble(1, t);
        ps.setDouble(2, c);
        double i=ps.executeUpdate();
    } catch(Exception e){
        System.out.println(e.getMessage());
    }
}
if(i==1)
{
    JOptionPane.showMessageDialog(null,"Added Sucessfully");
}

catch(Exception e)
{
    JOptionPane.showMessageDialog(null,"Exception sorry");
}

A.2 Screenshots:
A.3 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.)

IV. CONCLUSIONS
DBSCAN algorithm here considers only point objects but it could be extended for other spatial objects like polygons. Applications of DBSCAN to high dimensional feature spaces should be investigated and radius generation for this high dimensional data also has to be explored. It also fails to detect clusters with varied density. DBSCAN algorithm here considers only point objects but it could be extended for other spatial objects like polygons. Applications of DBSCAN to high dimensional feature spaces should be investigated and radius generation for this high dimensional data also has to be explored. It also fails to detect clusters with varied density.

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
[12] Henrik Bäcklund (henba892), Anders Hedblom (andh893), Niklas Neijman (nikne866). A Density-Based Spatial Clustering of Application with Noise