NAME: Meith Navlakha
SAP: 60004190068
DWM Practical

EXPERIMENT 10

AIM: Write and Explain one algorithm each on:

1. Spatial Association Rules
2. Spatial Classification
3. Spatial Clustering – DBScan

THEORY:

1.Spatial Association Rules

Input :
D // Data , including spatial and nonspatial attributes
c // Concept hierarchies
s // Minimum support for levels
Ci // Confidence
q // Query to retrieve interested objects
p // Topological predicate (s) of interest
Output :
R //Spatial association rules

SPATIAL association rule algorithm:

```
D'= q(D) ;

CP is built by applying the coarse predicate version of P to D';

        // CP consists of the set of coarse predicates satisfied by pairs of objects in D'.

determine the set of frequent coarse predicates FCP by finding the coarse predicates
that satisfy s;

find the set of frequent fine predicates FFP from FCP;

find R by finding all frequent fine predicates and then generating rules;
```

The algorithm works in a similar manner to the Apriori algorithm in that large "predicate sets" are determined. Here a predicate set is a set of predicates of interest. A 1-predicate might be {(close_to, park)}, so all spatial objects that are close_to a park will be counted as satisfying this predicate. A 2-predicate could be {(close_to, park), (south_of, Plano)}. Counts of 1-predicate sets are counted, then those that are large are used to generate 2-predicate sets, and these are then counted. In actuality, the algorithm can be used to generate multilevel association rules if desired or rules at a coarse level rather than a fine level.

2.Spatial Classification

Spatial Decision Tree:

One spatial classification technique builds decision trees using a two-step process similar to that used for association rules [KHS98]. The basis of the approach is that spatial objects can be described based on objects close to them. A description of the classes is then assumed to be based on an aggregation of the most relevant predicates for objects nearby.

To construct the decision tree, the most relevant predicates (spatial and nonspatial) are first determined. These relevant predicates are the ones that will be used to build the decision tree. It is assumed that a training sample is used to perform this step and that weights are assigned to attributes and predicates. Initial weights are 0. Two corresponding objects are examined for each object. The nearest miss is the spatial object closest to the target object that is in a different class. The nearest hit is the closest target in the same class. For each predicate value in the target object, if the nearest hit object has the same value, then the weight of that predicate is increased. If it has a different value, then the weight is decreased. Likewise, the weight is decreased (increased) if the nearest miss has the same (different) value. Only predicates with positive weights above a predefined threshold are then used to construct the tree. It is proposed that, because of the complexity of finding the relevant predicates, relevant predicates be found first at a coarse level and then at a finer leveL MBRs, instead of actual objects, and a generalized coarse close_to relationship are first used to find the relevant predicates. Then these relevant predicates and the true objects are used during the second pass.

For each object in the sample, the area around it, called its buffer, is examined. A description of this buffer is created by aggregating the values of the most relevant predicates of the items in the buffer. Obviously, the size and shape of the buffer impact the resulting classification algorithm. It is possible, although unrealistic, to perform an exhaustive search around all possible buffer sizes and shapes. The objective would be to choose the one that results in the best discrimination between classes in the training set. This would be calculated using the information gain. Other approaches based on picking a particular shape were examined, and the authors finally used circles (equidistance buffers).

To construct the tree, it is assumed that each sample object has associated with it a set of generalized predicates that it satisfies. Counts of the number of objects that satisfy (do not satisfy) each predicate can then be determined. This is then used to calculate information gain as is done in ID3. Instead of creating a multiway branching tree, a binary decision tree is created. The resulting algorithm to construct the decision tree is

Input parameters:

Target_table: the analysed objects (i.e. the analysed thematic layer),
Neighbor_table: thematic layer objects (neighbors of analysed objects),
Spatial_join_index: the join index table,
Target_attribute: the attribute to predict (i.e. class labels),
Predictive_attributes: attributes from a target table or neighbour table that could be used to predict the target attribute,
Saturation_condition: condition under which the split is considered invalid..

Output:

A binary decision tree

Algorithm

Step 1:

Initially, assign all target objects to the root (i.e. to node number 1)

Step 2:

```
Best_gain = 0
For each predictive_attribute
For each attribute_value
If the predictive_attribute belongs to the target_table
Info_gain = compute information gain -- as in CART
Else If the predictive_attribute belongs to neighbor_table
For each spatial relationship spatRel
Info_Gain = Compute information gain for the split criterion

"exists neighbours (by mean of spatRel) having such
attribute_value".
If Info_gain > Best_gain
Save the split criterion
-- The retained split criterion is the one maximising the information gain for all
predictive attributes.
```

Step 3:

```
If the current leaf is not saturated
Perform the node split
Assign its objects to the left son or to the right son accordingly.
```

Step 4:

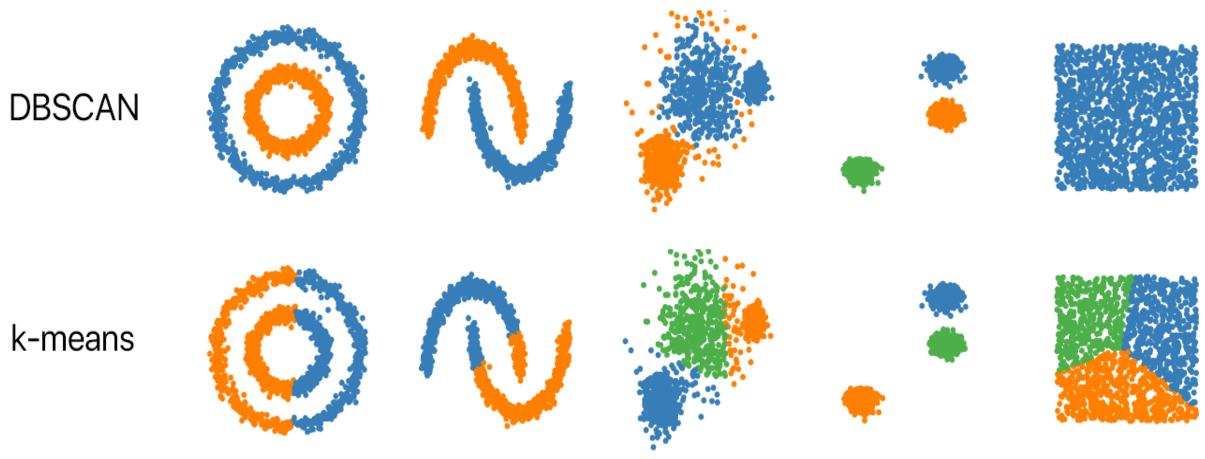
Replace the current node by the following according to the code number.
Iterate step 2 and step 3.
--The algorithm will stop when all the leaves of the lower level are saturated.

NAME: Meith Navlakha
SAP: 60004190068
DWM Practical

3. Spatial Clustering – DBScan

Density-Based Clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.



Parameters:

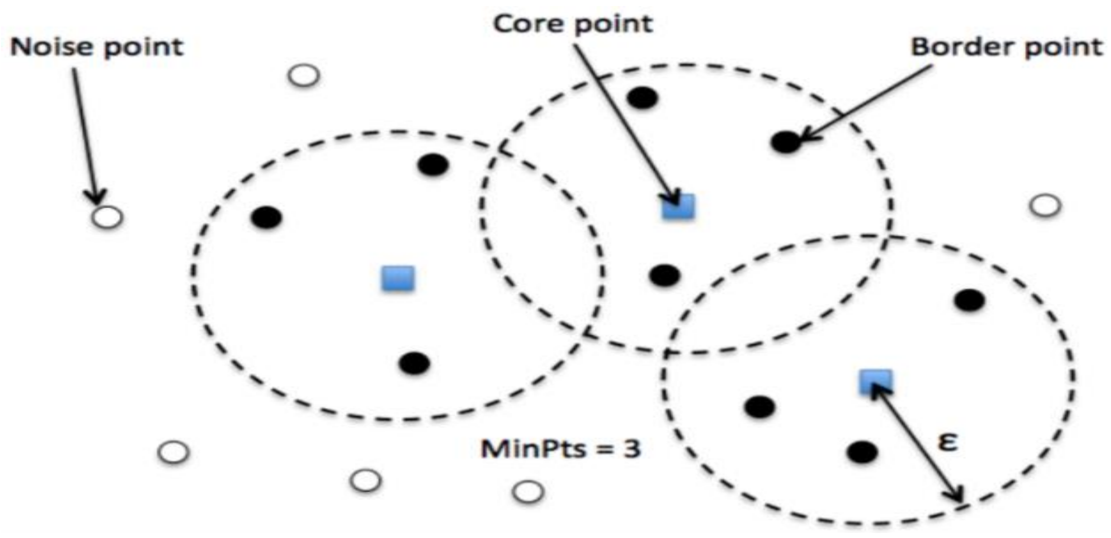
The DBSCAN algorithm basically requires 2 parameters:

eps: specifies how close points should be to each other to be considered a part of a cluster. It means that if the distance between two points is lower or equal to this value (eps), these points are considered neighbors.

minPoints: the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region.

DBSCAN has 3 types of data points:

1. **Core Point:** A point is a core point if it has more than MinPts points within eps.
2. **Border Point:** A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.
3. **Noise or outlier:** A point which is not a core point or border point.



Parameter estimation:

eps: if the eps value chosen is too small, a large part of the data will not be clustered. It will be considered outliers because don't satisfy the number of points to create a dense region. On the other hand, if the value that was chosen is too high, clusters will merge and the majority of objects will be in the same cluster. The eps should be chosen based on the distance of the dataset (we can use a k-distance graph to find it), but in general small eps values are preferable.

minPoints: As a genaeral rule, a minimum minPoints can be derived from a number of dimensions (D) in the data set, as minPoints ≥ D + 1. Larger values are usually better for data sets with noise and will form more significant clusters. The minimum value for the minPoints must be 3, but the larger the data set, the larger the minPoints value that should be chosen.

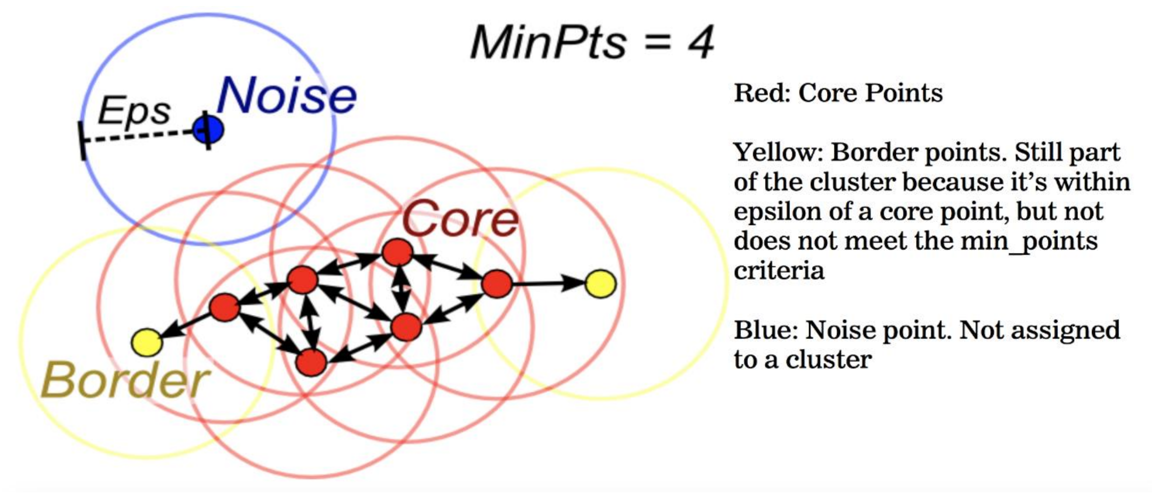
Distance function: The choice of distance function is tightly coupled to the choice of ε, and has a major impact on the results. In general, it will be necessary to first identify a reasonable measure of similarity for the data set, before the parameter ε can be chosen. There is no estimation for this 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.

Algorithm:

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

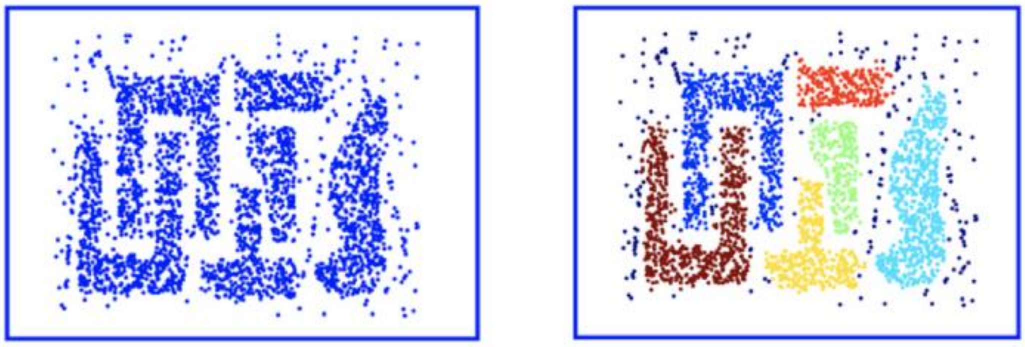
1. Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.
2. For each core point if it is not already assigned to a cluster, create a new cluster.
3. Find recursively all its density connected points and assign them to the same cluster as the core point.
   A point *a* and *b* are said to be density connected if there exist a point *c* which has a sufficient number of points in its neighbors and both the points *a* and *b* are within the *eps distance*. This is a chaining process. So, if *b* is neighbor of *c*, *c* is neighbor of *d*, *d* is neighbor of *e*, which in turn is neighbor of *a* implies that *b* is neighbor of *a*.
4. Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.


MinPts = 4
Eps
Noise
Core
Border
Red: Core Points
Yellow: Border points. Still part of the cluster because it's within epsilon of a core point, but not does not meet the min_points criteria
Blue: Noise point. Not assigned to a cluster

In this diagram, minPts = 4. The 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. The yellow points are not core points but are reachable from A (via other core points) and thus belong to the cluster as well. Blue point is a noise point that is neither a core point nor directly-reachable.

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.



The left image depicts the traditional clustering method that does not account for multi-dimensionality. Whereas the right image shows how DBSCAN can contort the data into different shapes and dimensions in order to find similar clusters.

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 is well understood.

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 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 Euclidean 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.
4. If the data and scale are not well understood, choosing a meaningful distance threshold ε can be difficult.

Complexity:

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). Worst Case Runtime 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.

NAME: Meith Navlakha
SAP: 60004190068
DWM Practical

CONCLUSION: In this experiment, Firstly, I learnt about Spatial Association Rules that can be used to generate multilevel association rules at a desired coarse level rather than a fine level. Unlike the traditional association rules, the underlying database being examined usually is not viewed as a set of transactions, instead, it is a set of spatial objects. Secondly, I learnt about Spatial Classification in which the concept of hierarchy is used along with sampling. In the experiment, I have used Spatial Decision Tree where the description of the classes is then assumed to be based on an aggregation of the most relevant predicates for objects nearby. Thirdly, I learnt about DBSCAN . The algorithm has a time complexity of O(nlogn) if ε is chosen in a meaningful way else worst case is $O(n^2)$.