of nodes. However, one drawback still remains–BIRCH is useful for detecting only spherical clusters.

4.8 Density-based Methods

In density-based methods, clusters are grown by starting with some data points and then including other neighbouring points as long as the neighbourhood is sufficiently dense. These methods can find clusters of arbitrary shape and are useful in the applications where such a feature is desired. As an example, when applied to datasets such as the one shown in Fig. 4.7, these algorithms will output two desired clusters that are visible.

4.8.1 DBSCAN Algorithm

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm discovers clusters as contiguous dense regions in the instance space. Before describing the algorithm, it is first necessary to make the notion of a 'dense region' precise.

Dense Region Definition

Given below is the definition of the intuitive notion of the neighbourhood of a point, as this must contain enough points to qualify as a dense region.

Neighbourhood The neighbourhood of a data point p consists of all the data points q such that $d(p, q) \leq \varepsilon$, where ε is a user-given parameter and d is the distance metric.

Core Point A data point is a core point if its neighbourhood contains at least μ data points, where m is a user-given parameter.

Intuitively, the implication of this definition is that the neighbourhood of a core point is a dense region.

Directly Density-reachable All the data points that are within the neighbourhood of a core point are said to be directly density-reachable from it.

Note that if p is directly density-reachable from a point q, then it is not necessary that q is directly density-reachable from p because p may not be a core point.

Density-reachable A data point p is said to be density-reachable from a core point q if there exists a sequence of data points $p_1, p_2, p_3, \ldots, p_n$, such that $p_1 = p$ and $p_n = q$, and p_i is directly density-reachable from p_{i+1}, for i in $1, \ldots, n$.

Density-connected All the data points that are density-reachable from a core point are said to be density-connected to each other.

Density-based Cluster A maximal set of mutually density-connected data points constitutes a density-based cluster.

This definition finally captures the intuitive notion of a contiguous dense region, i.e. a maximal set of mutually density-connected data points. Here the adjective maximal is used, if not, a portion of a contiguous dense region (that is also contiguous) will qualify as a cluster.

The Algorithm

The pseudo-code for DBSCAN is shown in Fig. 4.9, and described below.

The DBSCAN algorithm operates by searching the dataset for core points by examining the density of the neighbourhood of each data point (steps 1–4 in Fig. 4.9). If a spatial index is available, then the *neighbourhood*( ) function can be implemented efficiently. Or else, for each data point, the list of distances to the other points needs to be computed and sorted initially (as in the first iteration of Illustration 4.4). Step 2 in the pseudo-code is to ensure that the *neighbourhood* ( ) function is called only once for each point.

```
DBSCAN (Data D, ε, μ):

1. for each point p in D:
2.     if p has not been processed earlier:
3.         n = neighbourhood(p, ε)
4.         if |n| ≥ μ:          # p is a core point
5.             C = { p } ∪ n       # start a new cluster C
6.             Expand(C, n, ε, μ)

expand (Cluster C, n, ε, μ):

1. for each point q in n:
2.     m = neighbourhood(q, ε)
3.     if |m| ≥ μ:      # q is a core point
4.         C = C ∪ m
5.         Expand(C, m, ε, μ)
```

Fig. 4.9 The DBSCAN algorithm

Once a core point *p* is identified, the algorithm initializes a cluster *C* that consists of *p* and all the data points in the neighbourhood of *p* (step 5). Next, the algorithm calls the *expand* function that checks recursively for all data points that are density-reachable from p and adds them to C. In the actual implementation, the tail recursion in this function can be replaced by a more efficient iterative loop.

The DBSCAN algorithm deviates from the above pseudo-code in one aspect–during execution, it may turn out that some (non-core) point is density-reachable from two or more clusters. DBSCAN then assigns that point to the cluster discovered first.

Illustration 4.6 Figure 4.10(a) shows DBSCAN in execution on a small portion of a dataset.

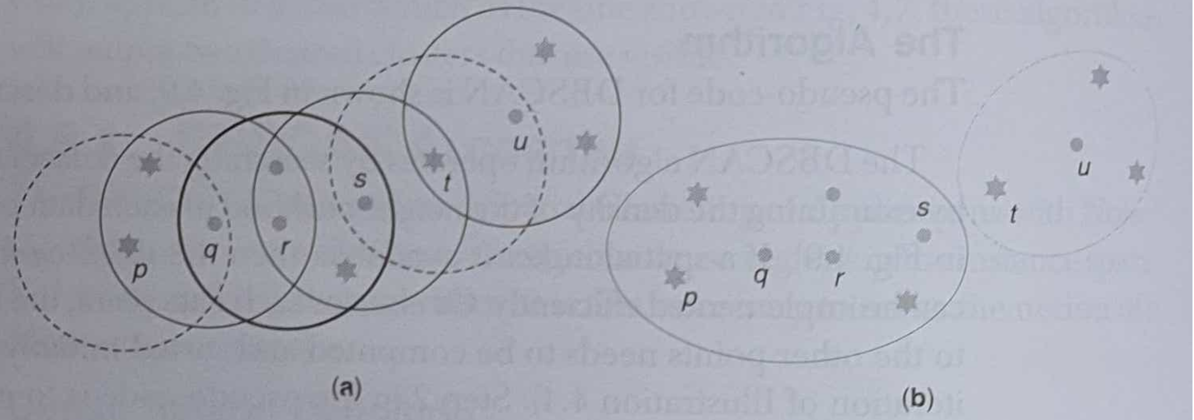

Fig. 4.10 (a) DBSCAN in execution with $\mu = 3$; (b) Final clusters obtained

Here μ (the minimum number of data points needed to form a core point) is set as 3. The core points are shown as dots whereas, the non-core points are shown as stars. Figure 4.10(b) shows the final clusters obtained for the same portion of the dataset.

To see how DBSCAN might have operated on the dataset shown in Fig. 4.10(a), suppose that the data points are ordered on disk in such a manner that DBSCAN first comes across point p. It retrieves the neighbourhood of p (shown as a dashed circle) and recognizes that it contains only two data points. Since $\mu = 3$, p is not a core point.

The next data point is retrieved from the dataset. Suppose that the point is μ. Its neighbourhood is retrieved (shown as a regular circle) and it contains four points. It is a core point. Hence, a cluster C_1 is created containing these four points in addition to μ.

Next, these four points in the neighbourhood of μ are checked for whether they are core points or not. In this case, none of them are core points, so the algorithm proceeds to the next data point on disk. Let us suppose that it is q. Its neighbourhood is retrieved (shown as a regular circle) and it contains four points. It is therefore, a core point. Hence, a cluster C_2 is created containing these four points in addition to q.

Next, these four points in the neighbourhood of q are checked for whether they are core points or not. If they are core points (for example, r), then the algorithm adds the data points in their neighbourhood to C_2. This way the cluster C_2 is grown and s

is also added to C_2. The algorithm would also attempt to add t to C_2. But since t is already in C_1, this fails and t is retained in C.

Analysis It is clear from the pseudo-code shown in Fig. 4.9 that the *neighbourhood* ( ) function is called only once for each data point. If a spatial index is available, this function can be executed in *log n* time where *n* is the number of data points. Hence, the complexity of the algorithm becomes $O(n \log n)$. Or else, for each data point, the list of distances to the other points needs to be computed and sorted initially (as in the first iteration of Illustration 4.4). This requires n^2 computations, and so the complexity becomes $O(n^2)$.

OPTICS: An enhancement to DBSCAN

Clusters in real-life datasets may have varying densities. This means that the ε- (or μ-) value needs to be selected differently for each cluster. OPTICS is an algorithm that achieves this end. Rather than taking a fixed ε-value from the user, it takes ε_{max} as input and in essence produces multiple clusterings—as though DBSCAN is called with all the possible values of ε that are less than or equal to ε_{max}.

The main idea is as follows: If DBSCAN is to produce two clusterings for two different user-given values of ε, then the pseudo-code in Fig. 4.9 can be modified to process both the ε-values simultaneously. The only changes required will be in providing the ε-values to the calls of the *neighbourhood*( ) function.

At this time, instead of making two different calls to the *neighbourhood*( ) function, this function can be modified to take both ε-values and return two different answer-sets. Notice now, that the answer-set to the smaller of the ε-values will be a subset of the answer-set to the bigger ε-value. Therefore, this can be implemented easily because the function can operate as though it is given the bigger ε-values as input, but the returned answer-set is sorted so that its first part contains the answer-set to the smaller ε-value.

The above idea can be generalized to process not just two different ε-values, but all the possible ε-values below some user-given ε_{max}. The *neighbourhood*( ) function only has to sort its answer-set of data points by their distance from the respective core point.

The resulting OPTICS algorithm makes use of the above intelligent strategy and has the same time complexity as DBSCAN. Effectively it stores, for every data point, its neighbourhood consisting of data points sorted by distance. It makes use of this information at run-time to provide the user with the flexibility of selecting any ε-value (not exceeding ε_{max}) and shows the resulting clusters interactively.

4.9 Grid-Based Methods

In grid-based methods, the space of instances is divided into a grid structure. Clustering techniques are then applied using the cells of the grid, instead of individual data points, as the basic units. This has the advantage of improving the processing time significantly.

4.9.1 CLIQUE Algorithm

One of the algorithms that use a grid-based technique is CLIQUE (CLustering In QUEst). It has been designed to specifically handle datasets with a large number of dimensions. A high-level pseudo-code for the algorithm is shown in Fig. 4.11, and described below.

The algorithm first partitions the data space into a grid. This is done by partitioning each dimension into equal intervals known as units (lines 1 and 2 in Fig. 4.11). It then identifies dense units (line 3)–a unit is dense if the fraction of the data points in it is more than a user-given parameter.

```
CLIQUE (Data D):
# Phase 1: Find dense units
1. for each dimension d in D:
2.     Partition d into equal intervals (known as units)
3.     Identify dense units
4. k = 2
5. while (1):
6.     for each combination of k dimensions d1, d2,...,dk:
7.         for each intersection i of dense units along the k dimensions
8.             if i is dense:
9.                 Mark i as a dense unit
10.            if no units marked as dense, break from while(1) loop.
11.    k = k + 1

# Phase 2: Find Clusters
12. Identify clusters as maximal sets of connected dense units
```

Fig. 4.11 The CLIQUE algorithm

Once the dense units along single dimensions are found, the algorithm attempts to find the dense units along two dimensions. The intuition for this is based on heuristic followed by frequent itemset mining algorithms—if a high-