Notes 06-5: Density Methods

A density method groups neighbouring objects into clusters based upon density conditions (i.e., such as the number of objects within a given radius of each object in the cluster exceeding some threshold).

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) method grows regions with sufficiently high density into clusters and discovers clusters of arbitrary shape in databases with noise (i.e., outliers).

Example – shape of clusters that DBSCAN can find

DIAGRAM = Clustering.G.1.a

The basic idea is that for each instance in a cluster, the neighbourhood of a given radius has to contain at least a minimum number of instances (i.e., the density in the neighbourhood has to exceed some threshold).

A discussion of DBSCAN relies on a number of concepts:

The *ε-neighbourhood* of an instance (i.e., a point) consists of the instances (i.e., points) within a radius ε of the instance.

A *core instance* is an instance whose ε-neighbourhood contains at least some minimum number of instances, `minPts`.

Example – core instances

DIAGRAM = Clustering.G.1.c1

Given a set of instances, D, an instance i is directly density-reachable from instances j if i is within the ε-neighbourhood of j, and j is a core instance.

Example – directly density-reachable instances

DIAGRAM = Clustering.G.1.c2

An instance i is *density-reachable* from instance j with respect to ε and `minPts` in a set of instances, D, if there is a chain of instances $i_1, i_2, \ldots, i_n$, $i_1 = j$ and $i_n = i$, such that i_{k+1} is directly density-reachable from i_k with respect to ε and `minPts`, for $1 \leq k \leq n$, $i_k \in D$.

Example – density-reachable instances

DIAGRAM = Clustering.G.1.c3

An instance i is *density-connected* to instance j with respect to ε and `minPts` in a set of instances, D, if there is an instance $p \in D$ such that both i and j are density-reachable from p with respect to ε and `minPts`.

Example – density-connected instances

DIAGRAM = Clustering.G.1.c4

A cluster K with respect to ε and `minPts` is a non-empty subset of a set of instances, D, that satisfies the following conditions:

- For all instances i and j, if $i \in K$ and j is density reachable from i with respect to ε and `minPts`, then $j \in K$.

- For all instances $i, j \in K$, i is density-connected to j with respect to ε and `minPts`.

If $K_1, \ldots, K_k$ are the clusters of a set of instances, D, with respect to ε_i and $\texttt{minPts}_i$, $i = 1, \ldots, k$, then noise is the set of instances in D not belonging to any K_i.

The general approach used by DBSCAN

DBSCAN searches for clusters by checking the ε-neighbourhood of each instance in the database.

If the ε-neighbourhood of an instance i contains more than `minPts`, a new cluster with i as the core instance is created.

Directly density-reachable instances from these core instances are iteratively collected (may involve merging of a few density-reachable clusters).

The process terminates when no new instances are added to any cluster.

The DBSCAN method

```
Algorithm: DBSCAN
```

```
Input: D = a set of n instances of the form (p1, p2, ..., pm)
       ε = the radius of the neighbourhood for each instance
       minPts = the minimum number of instances in an ε-
neighbourhood required for an instance to be a core instance
Output: K = a set of clusters
Method:
 1.  clusterID = 1
 2.  for i = 1 to |D|
 3.      currentInstance = GetNextInstance (D, i)
 4.      if currentInstance.clusterID == UNCLASSIFIED
 5.          if ExpandCluster (D, currentInstance, clusterID, ε,
     minPts)
 6.              clusterID ++
 7.  for i = 1 to |D|
 8.      currentInstance = GetNextInstance (D, i)
 9.      if currentInstance.clusterID != NOISE
 10.         K_currentInstance.clusterID = K_currentInstance.clusterID ∪
     currentInstance
 11. for i = 1 to clusterID - 1
 12.     K = K ∪ K_i

Algorithm: ExpandCluster
Input: D = a set of n instances of the form (p1, p2, ..., pm)
       currentInstance = an instance in D
       clusterID = identifier for the cluster currently being
expanded
       ε = the radius of the neighbourhood for each instance
       minPts = the minimum number of instances in an ε-
neighbourhood required for an instance to be a core instance
Output: TRUE if the cluster was expanded, otherwise, FALSE
Method:
 1.  D_seeds = InstancesIn_ε_Neighbourhood (currentInstance, D, ε)
 2.  if minPts > |D_seeds|
 3.      ChangeClusterID (D, currentInstance, NOISE)
 4.      return FALSE
 5.  else
 6.      for i = 1 to |D_seeds|
 7.          currentSeed = GetNextInstance (D_seeds, i)
 8.          ChangeClusterID (D, currentSeed, clusterID)
 9.      Delete (currentInstance, D_seeds)
 10.     while | D_seeds | > 0
 11.         currentInstance = GetFirstInstance (D_seeds)
 12.         ε_Neighbourhood = InstancesIn_ε_Neighbourhood
     (currentInstance, D, ε)
```

```
        if |ε_Neighbourhood| >= minPts
            for i = 1 to |ε_Neighbourhood|
                candidateInstance = GetNextInstance (ε_Neighbourhood, i)
                if candidateInstance.clusterID ε {UNCLASSIFIED, NOISE}
                    if candidateInstance.clusterID == UNCLASSIFIED
                        Append (Dseeds, candidateInstance)
                    ChangeClusterID (D, candidateInstance, clusterID)
        Delete (Dseeds, currentInstance)
    return TRUE
```

Example – DBSCAN

Instance	x	y	clusterID
1	2	3	*Unclassified*
2	2	5	*Unclassified*
3	2	6	*Unclassified*
4	3	2	*Unclassified*
5	3	4	*Unclassified*
6	3	6	*Unclassified*
7	3	7	*Unclassified*
8	4	1	*Unclassified*
9	4	4	*Unclassified*
10	4	5	*Unclassified*
11	4	7	*Unclassified*
12	5	2	*Unclassified*
13	5	6	*Unclassified*
14	5	9	*Unclassified*
15	6	2	*Unclassified*
16	6	5	*Unclassified*
17	6	7	*Unclassified*
18	6	9	*Unclassified*
19	7	6	*Unclassified*
20	7	7	*Unclassified*
21	8	6	*Unclassified*

Assume $\varepsilon = 1$ and `minPts = 3`.

Step 1: Initialize `clusterID`.

Step 2: Initialize `i = 1`. Since `i <= |`D`|`, go to Step 3.

Step 3: `currentInstance = (2,3,U)`

Step 4: Since `currentInstance.clusterID = UNCLASSIFIED`, go to Step 5.

Step 5: `ExpandCluster (`D`,(2,3,U),1,1,3)`

Step 5.1: Determine the instances within the ε-neighbourhood of `currentInstance = (2,3,U)`. Thus, D_{seeds} = `{(2,3,U)}`.

Step 5.2: Since `minPts > |`D_{seeds}`|`, there are not sufficient instances within the ε-neighbourhood, so go to Step 5.3.

Step 5.3: At this point, `currentInstances` is considered to be noise. Thus, `currentInstance = (2,3,N)`.

Step 5.4: The cluster could not be expanded around `currentInstance`, so return `FALSE` and go back to Step 2.

Step 2: Increment `i = 2`. Since `i <= |`D`|`, go to Step 3.

Step 3: `currentInstance = (2,5,U)`.

Step 4: Since `currentInstance.clusterID = UNCLASSIFIED`, go to Step 5.

Step 5: `ExpandCluster (`D`,(2,5,U),1,1,3)`.

Step 5.1: Determine the instances within the ε-neighbourhood of `currentInstance = (2,5,U)`. Thus, D_{seeds} `= {(2,5,U), (2,6,U)}`.

Step 5.2: Since `minPts > |`D_{seeds}`|`, there are not sufficient instances within the ε-neighbourhood, so go to Step 5.3.

Step 5.3: At this point, `currentInstance` is considered to be noise. Thus, `currentInstance = (2,5,N)`.

Step 5.4: The cluster could not be expanded around `currentInstance`, so return `FALSE` and go back to Step 2.

Step 2: Increment `i = 3`. Since `i <= |`D`|`, go to Step 3.

Step 3: `currentInstance = (2,6,U).`

Step 4: Since `currentInstance.clusterID = UNCLASSIFIED`, go to Step 5.

Step 5: `ExpandCluster (`D`,(2,6,U),1,1,3).`

Step 5.1: Determine the instances within the ε-neighbourhood of `currentInstance = (2,6,U)`. Thus, D_{seeds} `= {(2,6,U), (2,5,N), (3,6,U)}.`

Step 5.2: Since `minPts = |`D_{seeds}`|`, there are sufficient instances within the ε-neighbourhood, so go to Step 5.6.

Step 5.6: Initialize `i = 1`. Since `i <= |`D_{seeds}`|`, go to step 5.7.

Step 5.7: `currentSeed = (2,6,U).`

Step 5.8: Since `clusterID = 1`, `currentSeed = (2,6,1).`

Step 5.6: Increment `i = 2`. Since `i <= |`D_{seeds}`|`, go to step 5.7.

Step 5.7: `currentSeed = (2,5,N).`

Step 5.8: Since `clusterID = 1`, `currentSeed = (2,5,1).`

Step 5.6: Increment `i = 3`. Since `i <= |`D_{seeds}`|`, go to step 5.7.

Step 5.7: `currentSeed = (3,6,U).`

Step 5.8: Since `clusterID = 1`, `currentSeed = (3,6,1).`

Step 5.9: `currentInstance = (2,6,1)` is removed from D_{seeds}. Thus, D_{seeds} `= {(2,5,1), (3,6,1)}.`

Step 5.10: Since $|D_{\text{seeds}}|$ > 0, go to step 5.11.

Step 5.11: currentInstance = (2,5,1).

Step 5.12: ε_neighbourhood = {(2,5,1)}.

Step 5.13: Since | ε_neighbourhood| < minPts, go to step 5.20.

Step 5.20: currentInstance = (2,5,1) is removed from D_{seeds}. Thus, D_{seeds} = {(3,6,1)}. Go back to step 5.10.

Step 5.10: Since $|D_{\text{seeds}}|$ > 0, go to step 5.11.

Step 5.11: currentInstance = (3,6,1).

Step 5.12: ε_neighbourhood = {(3,6,1), (2,6,1), (3,7,U)}.

Step 5.13: Since |ε_neighbourhood| >= minPts, go to step 5.14.

Step 5.14: Initialize i = 1. Since i <= |ε_neighbourhood|, go to step 5.15.

Step 5.15: candidateInstance = (3,6,1).

Step 5.16: Since candidateInstance.ClusterID = 1, go back to step 5.14.

Step 5.14: Increment i = 2. Since i <= |ε_neighbourhood|, go to step 5.15.

Step 5.15: candidateInstance = (2,6,1).

Step 5.16: Since candidateInstance.ClusterID = 1, go back to step 5.14.

Step 5.14: Increment i = 3. Since i <= |ε_neighbourhood|, go to step 5.15.

Step 5.15: candidateInstance = (3,7,U).

Step 5.16: Since `candidateInstance.ClusterID = UNCLASSIFIED`, go to step 5.17.

Step 5.17: Since `candidateInstance.ClusterID = UNCLASSIFIED`, go to step 5.18.

Step 5.18: `candidateInstance` has the potential to expand the cluster, so D_{seeds} `= {(3,6,1), (3,7,U)}`.

Step 5.19: Since `clusterID = 1`, `candidateInstance = (3,7,1)`.

Step 5.20: `currentInstance = (3,6,1)` is removed from D_{seeds}. Thus, D_{seeds} `= {(3,7,1)}`. Go back to step 5.10.

Step 5.10: Since $|D_{seeds}| > 0$, go to step 5.11.

Step 5.11: `currentInstance = (3,7,1)`.

Step 5.12: ε`_neighbourhood = {(3,7,1), (3,6,1), (4,7,U)}`.

Step 5.13: Since |ε`_neighbourhood| >= minPts`, go to step 5.14 and repeat step 5.14 to 5.16 for `(3,7,1)` and `(3,6,1)`. As a result of this, there is no change to `(3,7,1), (3,6,1)` and D_{seeds}.

Step 5.14: Initialize `i = 3`. Since `i <= |`ε`_neighbourhood|`, go to step 5.15.

Step 5.15: `candidateInstance = (4,7,U)`.

Step 5.16: Since `candidateInstance.ClusterID = UNCLASSIFIED`, go to step 5.17.

Step 5.17: Since `candidateInstance.ClusterID = UNCLASSIFIED`, go to step 5.18.

Step 5.18: `candidateInstance` has the potential to expand the cluster, so D_{seeds} `= {(3,7,1), (4,7,U)}`.

Step 5.19: Since `clusterID = 1`, `candidateInstance = (4,7,1)`.

Step 5.20: `currentInstance = (3,7,1)` is removed from D_{seeds}. Thus, D_{seeds} `= {(4,7,1)}`. Go back to step 5.10.

Step 5.10: Since `|`D_{seeds}`| > 0`, go to step 5.11.

Step 5.11: `currentInstance = (4,7,1).`

Step 5.12: ε`_neighbourhood = {(4,7,1), (3,7,1)}.`

Step 5.13: Since `|`ε`_neighbourhood| < minPts`, go to step 5.20.

Step 5.20: `currentInstance = (4,7,1)` is removed from D_{seeds}. Thus, D_{seeds} `=` $\emptyset$. Go back to step 5.10.

Step 5.10: Since `|`D_{seeds}`| = 0`, go to step 5.21.

Step 5.21: Return `TRUE`.

Step 6: Increment `clusterID = 2`. Go back to step 2.

Step 2: Increment `i = 4`. Since `i <= |`D`|`, go to Step 3.

Step 3: `currentInstance = (3,2,U).`

Step 4: Since `currentInstance.clusterID = UNCLASSIFIED`, go to Step 5.

Step 5: `ExpandCluster (`D`,(3,2,U),2,1,3).`

Step 5.1: Determine the instances within the ε-neighbourhood of `currentInstance = (3,2,U)`. Thus, D_{seeds} `= {(3,2,U)}.`

Step 5.2: Since `minPts > |`D_{seeds}`|`, there are not sufficient instance within the ε-neighbourhood, so go to Step 5.3.

Step 5.3: At this point, `currentInstance` is considered to be noise. Thus, `currentInstance = (3,2,N).`

Step 5.4: The cluster could not be expanded around `currentInstance`, so return `FALSE` and go back to Step 2.

Step 2: Increment `i = 5`. Since `i <= |`D`|`, go to Step 3.

Step 3: `currentInstance = (3,4,U).`

Step 4: Since `currentInstance.clusterID = UNCLASSIFIED`, go to Step 5.

Step 5: `ExpandCluster (`D`,(3,4,U),2,1,3).`

Step 5.1: Determine the instances within the ε-neighbourhood of `currentInstance = (3,4,U)`. Thus, D_{seeds} `= {(3,4,U), (4,4,U)}.`

Step 5.2: Since `minPts > |`D_{seeds}`|`, there are not sufficient instances within the ε-neighbourhood, so go to Step 5.3.

Step 5.3: At this point, `currentInstance` is considered to be noise. Thus, `currentInstance = (3,4,N).`

Step 5.4: The cluster could not be expanded around `currentInstance`, so return `FALSE` and go back to Step 2.

Step 2: Increment `i = 6` and `i = 7` and repeat steps 3 and 4 for `(3,6,1)` and `(3,7,1)` which are have already been placed in cluster 1.

Step 2: Increment `i = 8` and do steps 2 to 5.4 for `(4,1,N)` which has already been classified as noise.

Step 2: Increment `i = 9`. Since `i <= |`D`|`, go to Step 3.

Step 3: `currentInstance = (4,4,U).`

Step 4: Since `currentInstance.clusterID = UNCLASSIFIED`, go to Step 5.

Step 5: `ExpandCluster (`D`,(4,4,U),2,1,3).`

Step 5.1: Determine the instances within the ε-neighbourhood of `currentInstance = (4,4,U)`. Thus, D_{seeds} `= {(4,4,U), (3,4,N), (4,5,U)}.`

Step 5.2: Since `minPts = |`$D_{\texttt{seeds}}$`|`, there are sufficient instances within the ε-neighbourhood, so go to Step 5.6.

Step 5.6: Initialize `i = 1`. Since `i <= |`$D_{\texttt{seeds}}$`|`, go to step 5.7.

Step 5.7: `currentSeed = (4,4,U).`

Step 5.8: Since `clusterID = 2`, `currentSeed = (4,4,2).`

Step 5.6: Increment `i = 2`. Since `i <= |`$D_{\texttt{seeds}}$`|`, go to step 5.7.

Step 5.7: `currentSeed = (3,4,N).`

Step 5.8: Since `clusterID = 2`, `currentSeed = (3,4,2).`

Step 5.6: Increment `i = 3`. Since `i <= |`$D_{\texttt{seeds}}$`|`, go to step 5.7.

Step 5.7: `currentSeed = (4,5,U).`

Step 5.8: Since `clusterID = 1`, `currentSeed = (4,5,2).`

.

.

.

Step 7: Initialize `i = 1`. Since `i <= |`D`|`, go to Step 8.

Step 8: `currentInstance = (2,3,N).`

Step 9: Since `currentInstance.clusterID = NOISE`, go back to step 7.

Step 7: Increment `i = 2`. Since `i <= |`D`|`, go to Step 8.

Step 8: `currentInstance = (2,5,1).`

Step 9: Since `currentInstance.clusterID = 1`, go to step 10.

Step 10: K_1 `= {(2,5,1)}`. Go back to step 7.

Step 7: Increment `i = 3`. Since `i <= |`D`|`, go to Step 8.

Step 8: `currentInstance = (2,6,1)`.

Step 9: Since `currentInstance.clusterID = 1`, go to step 10.

Step 10: K_1 `= {(2,5,1), (2,6,1)}`. Go back to step 7.

.

.

.

Step 7: Increment `i = 5`. Since `i <= |`D`|`, go to Step 8.

Step 8: `currentInstance = (3,4,2)`.

Step 9: Since `currentInstance.clusterID = 2`, go to step 10.

Step 10: K_2 `= {(3,4,2)}`. Go back to step 7.

.

.

.

Step 10: K_1 `= {(2,5,1), (2,6,1), (3,6,1)}`. Go back to step 7.

.

.

.

Step 10: K_2 `= {(3,4,2), (4,4,2)}`. Go back to step 7.

.

.

.

Step 11: Initialize `i = 1`. Since `i <= clusterID - 1`, go to step 12.

Step 12: K `= {{2,5,1), (2,6,1), (3,6,1), (3,7,1), (4,7,1)}}`. Go back to step 11.

Step 11: Initialize `i = 2`. Since `i <= clusterID - 1`, go to step 12.

Step 12: K `= {{2,5,1), (2,6,1), (3,6,1), (3,7,1), (4,7,1)}, {(3,4,2), (4,4,2), (4,5,2)}}`. Go back to step 11.

.

.
.

The average run time complexity of DBSCAN is O(n log n).

The experimental results reported in the original paper are all incorrect because the authors were unaware of a serious bug in their program (they weren't clustering all the points in the dataset).

The OPTICS (Ordering Points To Identify the Clustering Structure) method extends the DBSCAN method to consider a set of distance parameter values (i.e., a set of ε's) in order to generate a set of clusters whose densities may be different.

Example – Clusters with different density parameters

DIAGRAM = Clustering.G.2.a

Example – Nested clusters

DIAGRAM = Clustering.G.2.b