University of Southern Denmark
IMADA
Ricardo Campello

Credits:

- Exercise 6.1 was adapted and extended from Arthur Zimek's original for DM870
- Exercise 6.5 is fully reproduced from Arthur Zimek's original for DM870

DM583: Data Mining

Exercise 6: Density Based Clustering and Outlier Detection

(NOTE: THIS DOCUMENT DOESN'T INCLUDE SOLUTIONS FOR EXERCISES 6.1 AND 6.5)

Exercise 6-1 Density-Based Clustering (Conceptual)

a) Consider a dataset in a D-dimensional Euclidean space, consisting of two clusters. Assume that these clusters are each contained within well-delineated spatial boundaries, specifically, each cluster is spatially restricted to the interior of its own D-dimensional "ball" centered at a particular location of the space. Assume these clusters both follow a *uniform distribution* within such boundaries. Initially, also assume that: (i) these boundaries do not overlap and are well separated; and (ii) an observation in the dataset is equally likely to come (i.e., to have been drawn) from either cluster. Finally, assume that the dataset (sample) size is large enough so a rough KNN density estimate as the one used by DBSCAN can reasonably capture the density profile of the dataset for a given choice of the neighborhood size, MinPts, allowing for both cluster detection and separability. Describe what you would expect in this case for different choices of the radius ε, including values that are too high or too low. Note: For simplicity, you can ignore DBSCAN's border points (i.e., you can think of DBSCAN*).

b) Repeat item (a), but now assuming that observations in the dataset are more likely to belong to one of the two clusters.

c) Repeat item (a), but now assuming that, within the same cluster boundaries as before, the distribution (shared by both equally likely clusters) is radial rather than uniform, i.e., density is the highest at the center and lowest at the borders.

d) If for each object (observation) in the dataset of size n we need to find, among all other n-1 objects, the ones within a distance ε, how come that the asymptotic computational complexity of DBSCAN can be made smaller than O(n^2) in certain application scenarios?

e) Discuss the relation between the algorithms DBSCAN*, HDBSCAN* and Single-Linkage for MinPts = 1 or 2. Is there any difference between the cases where MinPts = 1 or 2?

f) Argue that border points in DBSCAN (which do not exist in DBSCAN*) conceptually violate Hartigan's classic model of density-contour clusters.

g) Discuss possible ways to solve ties for border points shared by core points from different clusters.

Exercise 6-2 DBSCAN Algorithm at Work

a) Consider the (symmetric) dissimilarity matrix ***D*** below, containing the pairwise distance between the data objects in a dataset. Manually apply DBSCAN to cluster this dataset with ε = 5 and MinPts = 3 (which includes the query object in question), describing the partial outcomes step-by-step, then indicating the final type of each object (core, border or noise) as well as the resulting clusters. Note: in case a border object is within reach from core points from multiple clusters, you can just assign them to these clusters (overlapping assignment allowed).

$$\mathbf{D} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\begin{bmatrix} 0 & & & & & & \\ 1 & 0 & & & & & \\ 1 & 2 & 0 & & & & \\ 10 & 9 & 11 & 0 & & & \\ 11 & 10 & 12 & 2 & 0 & & \\ 21 & 20 & 22 & 18 & 19 & 0 & \\ 14 & 13 & 15 & 6 & 4 & 25 & 0 \end{bmatrix}$$

Proposed Solution: For the sake of simplicity and illustration we will keep an explicit list with the state of each object as noise (n), border (b), or core (c), as well as their state as undiscovered (u), discovered (d) and already queried (q). We can also keep their assigned cluster(s) in the same state list. All objects are initialized in this list as noise, undiscovered and, accordingly, not belonging to any cluster yet:

Object State = {1 (n,u,∅), 2 (n,u,∅), 3 (n,u,∅), 4 (n,u,∅), 5 (n,u,∅), 6 (n,u,∅), 7 (n,u,∅)}

If we start processing with the first object, upon querying the first column of matrix ***D*** (here, via simple linear search) we retrieve objects 2 and 3 as the only ones within a ball of radius $\varepsilon = 5$ from object 1, in addition to object 1 itself, so there are 3 objects within such a ball, which makes object 1 a core point since MinPts = 3. Object 1 is then assigned the first clusters label, say Cluster A, and the objects within its reach (2 and 3) are also assigned to the same cluster but provisionally labelled as border points. These are included in a container Ψ (list, queue, stack, whatever) for further processing. So, at this stage, the partial state is:

$\Psi = \{2, 3\}$
Object State = {1 (c,q,A), 2 (b,d,A), 3 (b,d,A), 4 (n,u,∅), 5 (n,u,∅), 6 (n,u,∅), 7 (n,u,∅)}

We then pop the next element from Ψ, object 2, and after querying ***D*** we retrieve objects 1 and 3 as the only ones within a distance of ε (in addition to obj. 2 itself), so obj. 2 is a core point given MinPts = 3, and its status is then changed as such. Since its ε-reachable neighbors had both already been discovered, nothing else is done.

$\Psi = \{3\}$
Object State = {1 (c,q,A), 2 (c,q,A), 3 (b,d,A), 4 (n,u,∅), 5 (n,u,∅), 6 (n,u,∅), 7 (n,u,∅)}

We then pop the next element from Ψ, object 3, and after querying ***D*** we retrieve objects 1 and 2 within a distance of ε (in addition to obj. 3 itself), so obj. 3 is a core point given MinPts = 3, and its status is then changed as such. Since its ε-reachable neighbors had both already been discovered, nothing else is done.

$\Psi = \{ \}$
Object State = {1 (c,q,A), 2 (c,q,A), 3 (c,q,A), 4 (n,u,∅), 5 (n,u,∅), 6 (n,u,∅), 7 (n,u,∅)}

Since Ψ is empty, we restart from one of the undiscovered objects[1], say object 4, and after querying ***D*** we retrieve object 5 as the only one within a distance of ε (in addition to obj. 4 itself), so object 4 is NOT a core point given MinPts = 3, and its status as noise remains unchanged at this stage. Object 5, which was previously undiscovered, is included into Ψ.

$\Psi = \{5\}$
Object State = {1 (c,q,A), 2 (c,q,A), 3 (c,q,A), 4 (n,q,∅), 5 (n,d,∅), 6 (n,u,∅), 7 (n,u,∅)}

We then pop the next element from Ψ, object 5, and retrieve objects 4 and 7 within a distance of ε (in addition

[1] In practice, pointers/references to and from these objects could be stored in a separate data structure for O(1) access.

to obj. 5 itself), so obj. 5 is a core point given MinPts = 3, and its status is then changed as such. Since obj. 5 had not been assigned an existing cluster label yet, it is then assigned a new (the second) label, say Cluster B, and the objects within its reach (4 and 7) are also assigned to the same cluster and provisionally promoted from noise to border points. Object 7, which had not been discovered yet, is also included into Ψ.

Ψ = {7}
Object State = {1 (c,q,A), 2 (c,q,A), 3 (c,q,A), 4 (b,q,B), 5 (c,q,B), 6 (n,u,∅), 7 (b,d,B)}

We then pop the next element from Ψ, object 7, and retrieve object 5 as the only one within a distance of ε (in addition to obj. 7 itself), so obj. 7 is NOT a core point given MinPts = 3, and its current status as border remains unchanged, indefinitely.

Ψ = {}
Object State = {1 (c,q,A), 2 (c,q,A), 3 (c,q,A), 4 (b,q,B), 5 (c,q,B), 6 (n,u,∅), 7 (b,q,B)}

Since Ψ is empty, we restart from the only remaining undiscovered object, namely, object 6, and after querying ***D*** we don't retrieve any other object within a distance of ε (in addition to obj. 6 itself), so object 6 is NOT a core point given MinPts = 3, and its status as noise remains unchanged.

Ψ = {}
Object State = {1 (c,q,A), 2 (c,q,A), 3 (c,q,A), 4 (b,q,B), 5 (c,q,B), 6 (n,q,∅), 7 (b,q,B)}

Since Ψ is empty and there aren't any undiscovered objects remaining, the algorithm stops. The final solution is as follows: Cluster A = {1, 2, 3} and Cluster B = {4, 5, 7}; Object 6 doesn't belong to any cluster, i.e., it is left unclustered as a *noise point*. Objects 4 and 7 belong to cluster B as *border points*. Objects 1, 2, 3 (within cluster A) and 5 (within cluster B) belong to their respective clusters as *core points*.

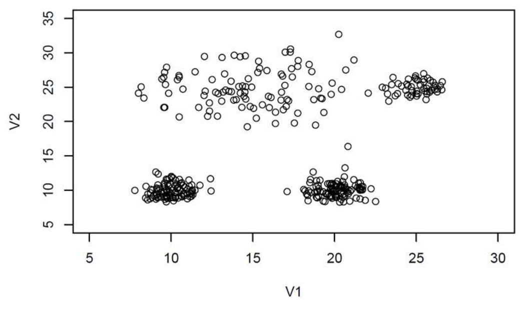
b) Repeat for other values of ε and MinPts.

Proposed Solution: Solution undisclosed.

Exercise 6-3 Practical DBSCAN and HDBSCAN* in R

a) Use trial-and-error to determine a suitable combination of values for minPts and ε that allows DBSCAN to reasonably recover the 4 clusters in the data set produced by the R code below (see figure). Plot your solution with the detected clusters in different colours and the noise points in black.

```
> set.seed(0)
> V11 <- rnorm(n = 100, mean = 10, sd = 1) # Cluster 1 (V1 coordinate)
> V21 <- rnorm(n = 100, mean = 10, sd = 1) # Cluster 1 (V2 coordinate)
> V12 <- rnorm(n = 100, mean = 20, sd = 1) # Cluster 2 (V1 coordinate)
> V22 <- rnorm(n = 100, mean = 10, sd = 1) # Cluster 2 (V2 coordinate)
> V13 <- rnorm(n = 100, mean = 15, sd = 3) # Cluster 3 (V1 coordinate)
> V23 <- rnorm(n = 100, mean = 25, sd = 3) # Cluster 3 (V2 coordinate)
> V14 <- rnorm(n = 50, mean = 25, sd = 1) # Cluster 4 (V1 coordinate)
> V24 <- rnorm(n = 50, mean = 25, sd = 1) # Cluster 4 (V2 coordinate)
> dat <- data.frame(V1 = c(V11,V12,V13,V14), V2 = c(V21,V22,V23,V24))
> plot(dat$V1, dat$V2, xlim = c(5,30), ylim = c(5,35), xlab = "V1", ylab = "V2")
```



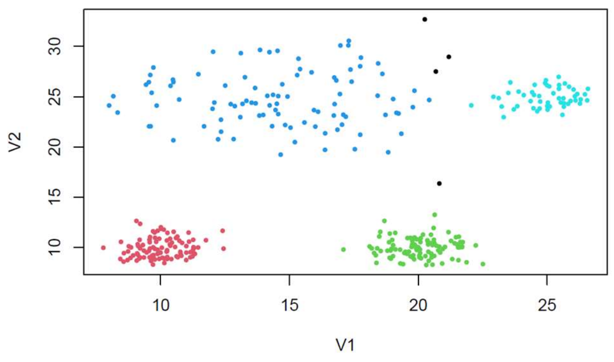
Proposed Solution:

After some trial-and-error based on visualization of the results (just for pedagogical purposes, as this wouldn't be possible in practice for higher dimensional datasets!!!), we got e.g. ε = 2.8 and MinPts = 15 as a good setup.

```
> require(dbscan)
> dbs_g <- dbscan(dat, eps = 2.8, minPts = 15)
> dbs_g

## DBSCAN clustering for 350 objects.
## Parameters: eps = 2.8, minPts = 15
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 4 cluster(s) and 4 noise points.
##
## 0 1 2 3 4
## 4 100 100 95 51
##
## Available fields: cluster, eps, minPts, dist, borderPoints

> color_1to8 <- function(x) ifelse(x==0,1,((x-1)%%7)+2)
> par(mfrow = c(1, 1))
> plot(dat, pch=19, cex=0.5, col=color_1to8(dbs_g$cluster), xlab="V1", ylab="V2")
```

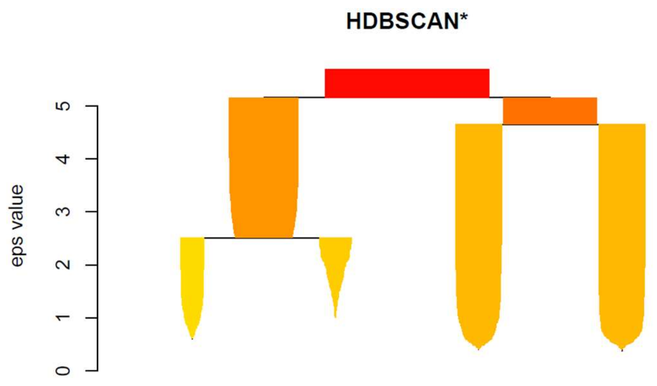


b) Repeat for HDBSCAN*, displaying both the hierarchical solution (simplified cluster tree, which you should interpret) as well as the flat partitioning that can (optionally) be automatically extracted by the algorithm.

Disclaimer about the R implementation of HDBSCAN*: *The R implementation of HDBSCAN* from package* `dbscan()` *– function* `hdbscan()` *– is not memory efficient (as of writing of this document) and may produce a memory overflow error for datasets larger than a few tens of thousands of observations, depending on your hardware specs. Alternative implementations exist that are computationally much more efficient, e.g., in Python*.

Proposed Solution:

```
> hdbs <- hdbscan(dat, minPts = 10)
> plot(hdbs)
```

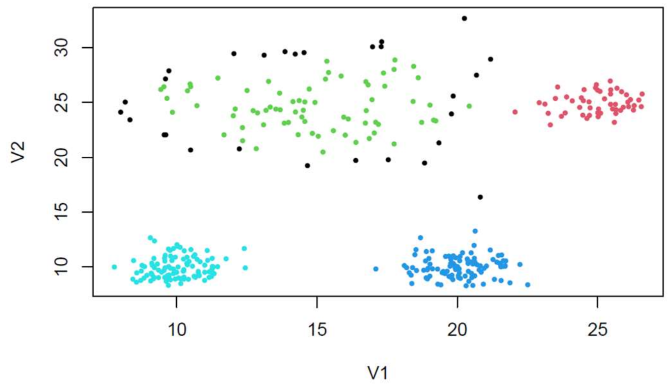


Interpretation of the simplified tree above: from top to bottom, we start with the whole dataset into a single large (red) "cluster", which as we raise the density threshold eventually splits into two (dark orange) clusters of similar sizes (based on their thickness), both of which eventually split into two descendant clusters each, and these in turn only "shrink" as sparser objects in their vicinity go below the increasing density threshold and become noise, until everything vanishes as noise. Notice that the dark orange clusters subdivide at very different density levels. This suggests that the one on the right has a clearer density separation between its subclusters than the one on the left (which will first noticeably shrink before splitting, which only happens at a higher density threshold).

```
> hdbs

## HDBSCAN clustering for 350 objects.
## Parameters: minPts = 10
## The clustering contains 4 cluster(s) and 28 noise points.
##
## 0 1 2 3 4
## 28 51 71 100 100
##
## Available fields: cluster, minPts, coredist, cluster_scores,
## membership_prob, outlier_scores, hc

> plot(dat, pch=19, cex=0.5, col=color_1to8(hdbs$cluster), xlab="V1", ylab="V2")
```



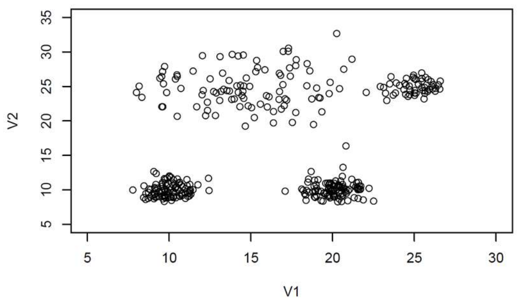
c) Discuss the main differences between the experiment with DBSCAN in item (a) as contrasted to the experiment with HDBSCAN* in item (b).

Proposed Solution: HDBSCAN* only required specification of MinPts, and the results usually tend to be less sensitive to this parameter than they generally are to ε. In addition, HDBSCAN* produced a complete hierarchy, from which a flat partition was automatically extracted without arbitrary (user-defined) specification of ε. However, border points were not automatically assigned to clusters in this flat partition, and for this reason, some points in the vicinity of the sparser cluster were left unclustered as noise by HDBSCAN* (assignment to cluster would require an additional post-processing step). This has not happened in the DBSCAN solution.

Exercise 6-4 Practical KNN Outlier and LOF in R

a) Compute and display the top-20 KNN outliers for the dataset produced by the following code (see figure below), using a neighbourhood size *k* = 4. Use the standard (not weighted) KNN method.

```
> set.seed(0)
> V11 <- rnorm(n = 100, mean = 10, sd = 1) # Cluster 1 (V1 coordinate)
> V21 <- rnorm(n = 100, mean = 10, sd = 1) # Cluster 1 (V2 coordinate)
> V12 <- rnorm(n = 100, mean = 20, sd = 1) # Cluster 2 (V1 coordinate)
> V22 <- rnorm(n = 100, mean = 10, sd = 1) # Cluster 2 (V2 coordinate)
> V13 <- rnorm(n = 100, mean = 15, sd = 3) # Cluster 3 (V1 coordinate)
> V23 <- rnorm(n = 100, mean = 25, sd = 3) # Cluster 3 (V2 coordinate)
> V14 <- rnorm(n = 50, mean = 25, sd = 1) # Cluster 4 (V1 coordinate)
> V24 <- rnorm(n = 50, mean = 25, sd = 1) # Cluster 4 (V2 coordinate)
> dat <- data.frame(V1 = c(V11,V12,V13,V14), V2 = c(V21,V22,V23,V24))
> plot(dat$V1, dat$V2, xlim = c(5,30), ylim = c(5,35), xlab = "V1", ylab = "V2")
```



Proposed Solution:

```
> require(dbscan)
> k <- 4 # KNN parameter
> KNN_Outlier <- kNNdist(x=dat, k = k, all = TRUE)[,k] # KNN distance (outlier score)
> top_n <- 20 # No. of top outliers to be displayed
> rank_KNN_Outlier <- order(x=KNN_Outlier, decreasing = TRUE) # Sorting (descending)
> KNN_Result <- data.frame(ID = rank_KNN_Outlier, score = KNN_Outlier[rank_KNN_Outlier])
> head(KNN_Result, top_n) # Display only top-20
```

```
##    ID    score
## 1  237 4.423726
## 2  208 4.177243
## 3  228 3.411198
## 4  240 2.709398
## 5  218 2.518054
## 6  243 2.464709
## 7  262 2.380678
## 8  251 2.303902
## 9  299 2.127866
## 10 203 2.092628
## 11 233 2.079155
## 12 281 2.078306
## 13 229 2.071261
## 14 191 2.023404
## 15 267 2.012530
## 16 231 1.967966
## 17 291 1.956565
## 18 256 1.845032
## 19 248 1.832962
## 20 300 1.831324
```

b) Repeat for LOF.

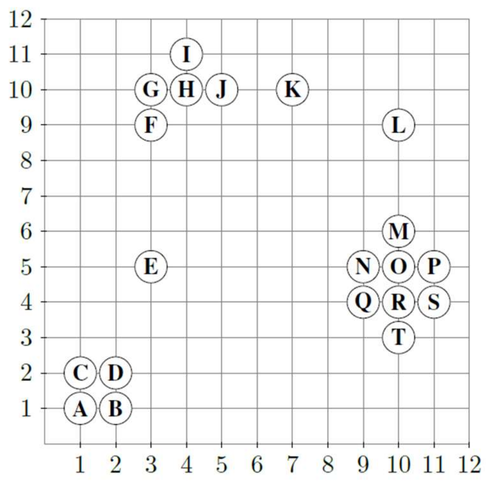
Proposed Solution:

```
> require(dbscan)
> k <- 4 # LOF parameter
> LOF_Outlier <- lof(x=dat, k = k) # LOF (outlier score) computation
> top_n <- 20 # No. of top outliers to be displayed
> rank_LOF_Outlier <- order(x=LOF_Outlier, decreasing = TRUE) # Sorting (descending)
> LOF_Result <- data.frame(ID = rank_LOF_Outlier, score = LOF_Outlier[rank_LOF_Outlier])
> head(LOF_Result, top_n)
```

```
##     ID  score
## 1  237 2.916624
## 2  191 2.908333
## 3  53  2.615192
## 4  130 2.539595
## 5  63  2.439944
## 6  10  2.402330
## 7  145 2.289441
## 8  54  2.091639
## 9  59  1.998936
## 10 333 1.992135
## 11 228 1.944133
## 12 104 1.840832
## 13 77  1.801824
## 14 188 1.766500
## 15 208 1.753433
## 16 168 1.726824
## 17 41  1.663926
## 18 331 1.662891
## 19 6   1.619528
## 20 15  1.616930
```

Exercise 6-5 KNN Outlier, Weighted KNN Outlier, and LOF at Work

Consider the following dataset in 2 dimensions:



As distance function, use Manhattan distance $L_1(a, b) := |a_1 - b_1| + |a_2 - b_2|$.

Compute the following (*without including the query point when determining the kNN*):

- KNN Outlier using $k = 2$ and $k = 4$ for all points
- Weighted (aggregated) KNN Outlier for $k = 2$ and $k = 4$ for all points (aggregated *k*NN distance = averaged distances to all the *k*NNs)
- LOF using $k = 2$ for points E, K and O
- LOF using $k = 4$ for points E, K and O