Data Mining Summer Term 2020

Institute of Neural Information Processing
PD Dr. F. Schwenker

Assignment 5 (Submission until June 09, 2020)

Exercise 1 (2 points): Clustering of Digits with k-means [Python]

In this exercise, we want to complete our detailed discussion about the k-means algorithm with a selected example application. We are using a dataset consisting of 10000 handwritten digit images and try to cluster them. Each grey-scale image has a size of 16×16 and shows a single digit ranging from 0 to 9. For the test script, please pay also attention to Table 1.

1. Load the digits data via

   ```
   import pickle
   data, labels = pickle.load(open('digits.pickle', 'rb'))
   ```

   The images are already re-shaped to a single row-vector of size 1×256 with integer values denoting the grey value in the range $\{0, 1, \ldots, 255\}$. The labels represent the digit number for each data point.

2. Apply k-means clustering (with the obvious choice of k) on the dataset. Use the `KMeans` class from the `sklearn.cluster` package and set the seed to 1337. Hint: enable parallel execution with the `n_jobs` parameter to speed up the computation.

3. Show the clustering result by displaying the centroid of each cluster (Figure 1). Note: you need to reshape the image before you can display it.

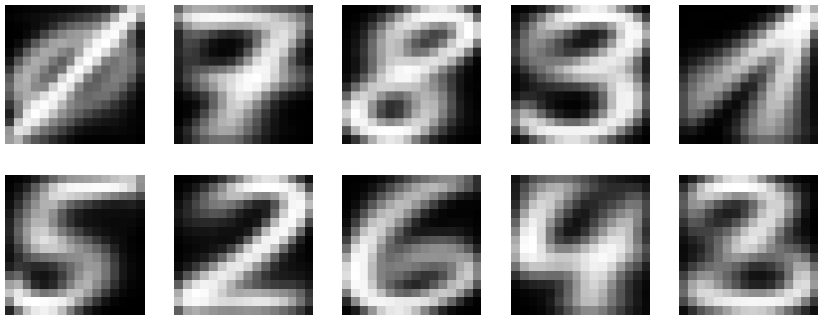


Figure 1: k-means cluster centroids obtained from the handwritten digits dataset.

Table 1: Variable names for the notebook `kMeansClustering.ipynb` (Exercise 1). You have to name your variables exactly as noted here so that the test script can check their content.

Name	Type	Description
`centroids`	Matrix	Definitions of the cluster centroids (shape 10 × 256).

Exercise 2 (5 points): DBSCAN [Pen and Paper]

In this exercise, we want to take a look at a new clustering algorithm which works fundamentally different from previous approaches like *k*-means: *density-based spatial clustering of applications with noise (DBSCAN)* [1]. The general observation in DBSCAN is that a cluster is formed by a dense group of points which are connected in a neighbourhood[1]. The goal is to determine how large this neighbourhood is, how many points it contains and how many neighbourhoods (clusters) there are in total. Additionally, it has the capability to recognize outliers in the dataset which do not really belong to any cluster. DBSCAN is especially suited to detect clusters with a shape different from the usual blob-like structure (an example follows in Exercise 3).

The critical part in DBSCAN is the definition and the detection of the neighbourhood for each point. For this, we introduce two new parameters:

- A neighbourhood radius ε around each point which controls how distant a point may be so that it is still considered a neighbour. When using the Euclidean distance (what we want to do here), this spans a circle around each point describing its neighbourhood.
- A minimum number of points $n_{\min}$ which must lie in the neighbourhood of a point (inside the circle) including the point itself. This controls the density of the cluster.

Each point defines its own neighbourhood but a cluster consists actually of a larger group of adjacent points. The key aspect here is that we want to define a cluster as a group of points which are directly or indirectly connected. That is, two points may be part of the same cluster even though they are not in the ε range of each other but with other points in between. This allows us to detect arbitrarily shaped clusters provided that enough points define the shape of the cluster.

The fact that points are linked together in a larger neighbourhood also sets up the main strategy to find the clusters. We start with an arbitrary point $\boldsymbol{p}$, set up the neighbourhood

$$N_\varepsilon(\boldsymbol{p}) = \{\boldsymbol{p}_i, i = 1, 2 \ldots \mid \left\|\boldsymbol{p}_i - \boldsymbol{p}\right\|_2 \leq \varepsilon\} \tag{1}$$

and check whether the point $\boldsymbol{p}$ is surrounded by enough other points. If $N_\varepsilon(\boldsymbol{p}) \geq n_{\min}$, then all points in $N_\varepsilon(\boldsymbol{p})$ are added to the cluster. Once assigned, a point never changes its label anymore.

[1] This goes back to human perception as it is very easy to recognize a dense group of points as one cluster, cf. Figure 1 of the original paper [1].

The process is repeated for every collected point so that we build the cluster by creating a connection from point to point. Remaining clusters are found by selecting a different seed point which does not have a label, yet. The full process is described in Algorithm 1.

Algorithm 1: DBSCAN

Input: data points $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n$.
Input: neighbourhood radius $\varepsilon > 0$.
Input: minimal number of points $n_{\min}$ in a neighbourhood.
Output: cluster labels ω_i per data point (0 = noise, 1, 2, ... = found clusters).

```
1  ∀i ∈ {1, 2, ..., n} : ω_i = −1
     // All points are unassigned in the beginning
2  c = 0                    // Cluster number
3  for i = 1 to n do
4      if ω_i > 0 then
5          continue         // Point already processed
6      end
7      if |N_ε(p_i)| < n_min then
8          ω_i = 0          // Noise (this label is subject to change)
9      else
10         c = c + 1        // Start with a new cluster
11         grow(p_i, c)     // p_i is the root element
12     end
13 end
14 Function grow(p, c)
15     foreach p_j ∈ N_ε(p) do
16         if ω_j > 0 then
17             continue     // Point already assigned to another cluster
18         end
19         ω_j = c          // Every neighbour is part of the cluster
20         if |N_ε(p_j)| ≥ n_min then
21             grow(p_j, c) // p_j is a core point
22         end
23     end
24 end
```

The process can also be described with the help of a tree structure. The seed element which satisfies $n_{\min}$ makes up the root of the tree and the child nodes are the neighbours. Now, we traverse the graph e.g. in a depth-first manner and analyse the neighbourhood of each child node. When the $n_{\min}$ constrained is satisfied, we append further children to the queue. If not, then the branch has come to its end. As soon as a node is appended to the tree, it is part of the cluster even when it does not have enough points in its own neighbourhood.

The process repeats by selecting one of the remaining points as new seed essentially building a tree structure for each cluster. It is possible that some points never get assigned to any cluster and also don't satisfy the n_{min} constraint. These points are considered noise. However, noise points may be added to a cluster later on when they are captured by the expansion process of another cluster. The algorithm stops when every point is either part of a cluster or declared as noise[2]. In the end, we may end up with three different kinds of points:

- A point which is part of a cluster and has enough points in its surrounding. This is called a *core point* (branch in the tree).
- A point which is part of a cluster but does not have enough points in its surrounding. This is called a *border point* (leaf in the tree).
- A point which is not part of any cluster and does also not satisfy the n_{min} constraint. This is a *noise point*.

Figure 2 (a) shows an example with a few points $\boldsymbol{p}_i$ in the two-dimensional space and the corresponding neighbourhood drawn as circles ($\varepsilon = 1.48$). Since the picture can get quite confusing with an increasing number of points, the figure is also available online[3] where circles can be interactively shown or hidden and the neighbourhood radius ε is adjustable. In Figure 2 (b), the corresponding tree is depicted when $\boldsymbol{p}_2$ is selected as the seed point and $n_{\text{min}} = 4$ is used. The tree is constructed as follows:

I. $\boldsymbol{p}_2$ has four points (including itself) in its neighbourhood so all are appended as child nodes to the tree.

II. We check $\boldsymbol{p}_4$ as the first node (arbitrarily chosen). $\boldsymbol{p}_4$ has also $\boldsymbol{p}_3$ in its neighbourhood so this point is added to the tree as well.

III. Since there are no new points in the neighbourhood of $\boldsymbol{p}_3$, this branch has come to its end.

IV. In a similar way, the branch of the point $\boldsymbol{p}_5$ ends.

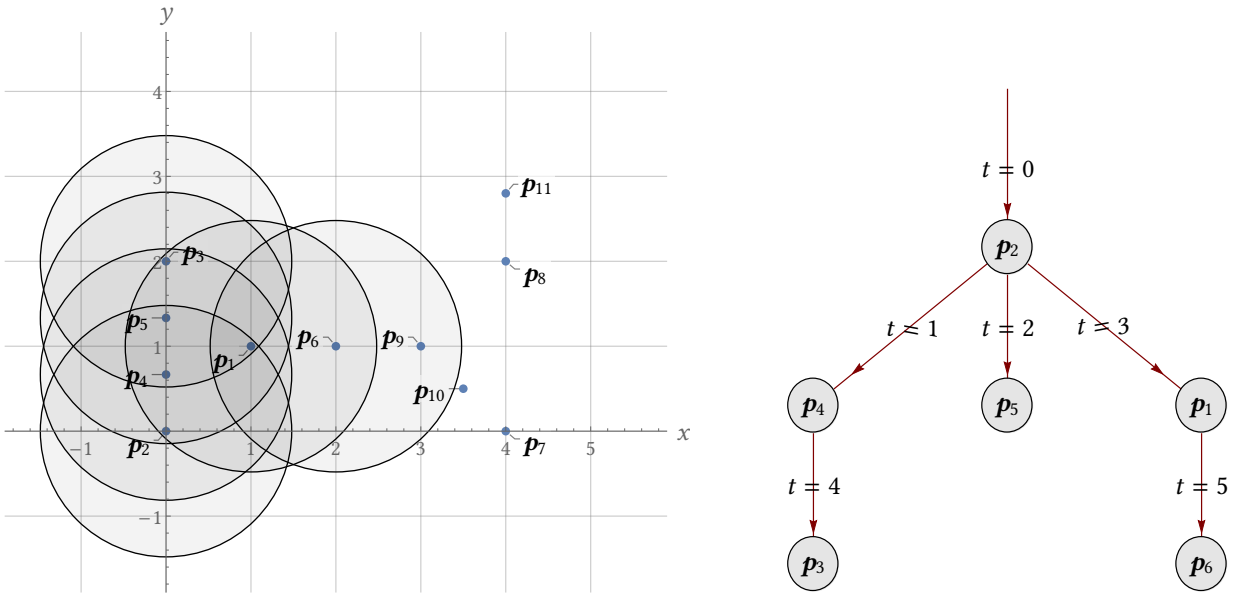
V. $\boldsymbol{p}_6$ is appended to $\boldsymbol{p}_1$ when processing the third branch.

VI. $\boldsymbol{p}_6$ does reach the point $\boldsymbol{p}_9$, however, there are only 3 points contained in its circle. Hence, no new points are added to the queue and with this the growth of the first cluster finished.

After these steps, we found the first cluster $C_1 = \{\boldsymbol{p}_2, \boldsymbol{p}_4, \boldsymbol{p}_5, \boldsymbol{p}_1, \boldsymbol{p}_3, \boldsymbol{p}_6\}$.

1. Proceed with the DBSCAN algorithm until it is completed. Use $\boldsymbol{p}_9$ as the next seed point and draw a tree for each remaining cluster. Set up the final cluster definitions C_i and note when a point is considered noise.

[2] In Algorithm 1, this is the case when we looped over every data point.
[3] https://milania.de/showcase/Neighbourhood_in_DBSCAN



(a) Example (b) Tree

Figure 2: Example dataset and the corresponding tree used to find the first cluster. The neighbourhood $N_{1.48}(\boldsymbol{p})$ (Equation 1) of the cluster points is shown as a circle around each point. The edges in the tree denote the steps and the node to which they are directed corresponds to the processed point.

2. Consider the point $\boldsymbol{p}_6$ again. This point was added to the cluster C_1, but, is this always the case? Argue whether you see any ambiguity in the cluster assignments here. Hint: the point $\boldsymbol{p}_2$ does not have to be the first seed point.

3. Use the parameters $\varepsilon = 0.9$ and $n_{\text{min}} = 2$ and repeat the algorithm. Set up the resulting clusters C_i and make a note for noise points. Select seed points of your choice. If you like, you can also draw a tree for each cluster.

4. We now want to put some emphasis on the differences between DBSCAN and k-means clustering. For comparison, answer the following questions for each of the algorithms.

   a) Is the algorithm deterministic, i.e. when we run it multiple times, do we get different results?

   b) Is the algorithm centroid- or density-based?

   c) Which parameter(s) do we need to specify before we can run the algorithm?

   d) How many iterations (processing of all points) are usually required for the algorithm?

   e) Can the algorithm detect noise points?

5. Open the online widget by Naftali Harris which lets you apply DBSCAN[4] and *k*-means[5] interactively on different examples. Select and run the algorithm on the SMILEY FACE dataset. DBSCAN should easily detect the four clusters while *k*-means fails to do so. Shortly explain why this is the case.

Exercise 3 (3 points): Parameters in DBSCAN [Python]

After covering the basic concepts of DBSCAN, we now want to take a look at a larger example. What is more, we want to use the opportunity and discuss how we can select suitable values for the parameters ε and n_{min}. Especially the former is very crucial and can highly affect the application performance. Here, we introduce a simple way to estimate the parameters based on the distances inside the k-neighbourhood of each point[6].

The basic idea is to look at the distance d_i to the k-th neighbour for every point $\boldsymbol{p}_i$, i.e. the distance to the point furthest away in the k-neighbourhood. Like before, the k-neighbourhood includes the point itself so that we always get $d_i = 0$ for $k = 1$. A larger distance d_i indicates that a point is relatively alone, i.e. probably a noise point.

The parameter k corresponds to the minimal number of points n_{min} in a neighbourhood, i.e. $k = n_{\text{min}}$ and is usually set to a small value in order to reduce the computational effort, e.g. we are using $k = 4$ here. Also, the parameter is less critical and hence requires less tuning.

Of higher importance is the neighbourhood radius ε. To estimate it, we plot the distance values d_i in a sorted way and look for a sudden increase (a "knee") in the graph and set ε correspondingly so that most of the points have a distance value $d_i \leq \varepsilon$. All of these points are included in a cluster since we already know that the neighbourhood $N_{d_i \leq \varepsilon}(\boldsymbol{p}_i)$ contains at least k points. The remaining points with $d_i > \varepsilon$ are either border or noise points.

We are now going to implement this idea for an artificial circle example which is especially well suited for DBSCAN. For the test script, please pay also attention to Table 2.

1. Generate the example dataset via the following code snippet:

```
import numpy as np
from sklearn import datasets

np.random.seed(1337)
data = datasets.make_circles(n_samples=500, noise=0.1, factor
    =0.4)[0]
```

[4] https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/
[5] https://www.naftaliharris.com/blog/visualizing-k-means-clustering/
[6] This idea was already proposed in the original DBSCAN paper [1].

2. Calculate the distance d_i for each data point $\boldsymbol{p}_i$ in the dataset, i.e. the maximal distance in the k-neighbourhood ($k = 4$). The `NearestNeighbors` class from the `sklearn.neighbors` package is helpful for this task.

3. Plot the distance values d_i. A possible solution is shown in Figure 3.

   a) Show a scatter plot of the dataset and visualize the distance d_i via the colour of the points. Does the result match with your expectation?

   b) Sort the distance values d_i, plot them and estimate a threshold ε (e.g. $\varepsilon = 0.15$ for the test script).

   c) Under which condition becomes a point above the threshold in the sorted 4-dist plot of Figure 3 a border instead of a noise point (orange line)?

4. Apply DBSCAN on the dataset with your chosen neighbourhood radius ε and visualize the result (Figure 4). The algorithm is implemented in the `sklearn.cluster` package.

References

[1] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise”. In: AAAI Press, 1996, pp. 226–231.

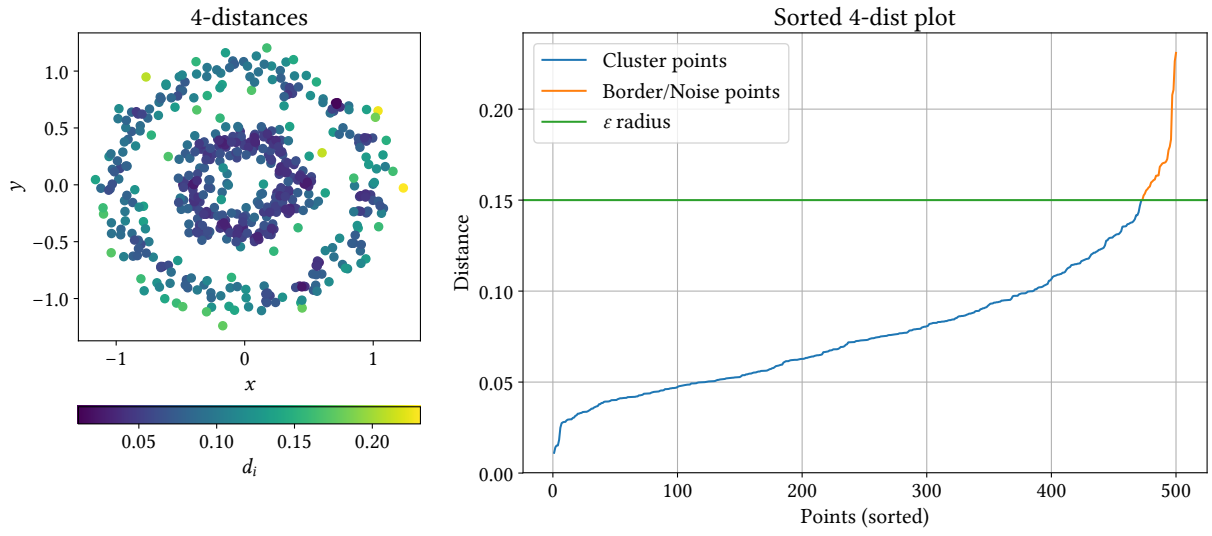


Figure 3: Visualization of the distances d_i. The left plot maps the distance values to the data points in the dataset and the right plot sorts the distances ascendingly.

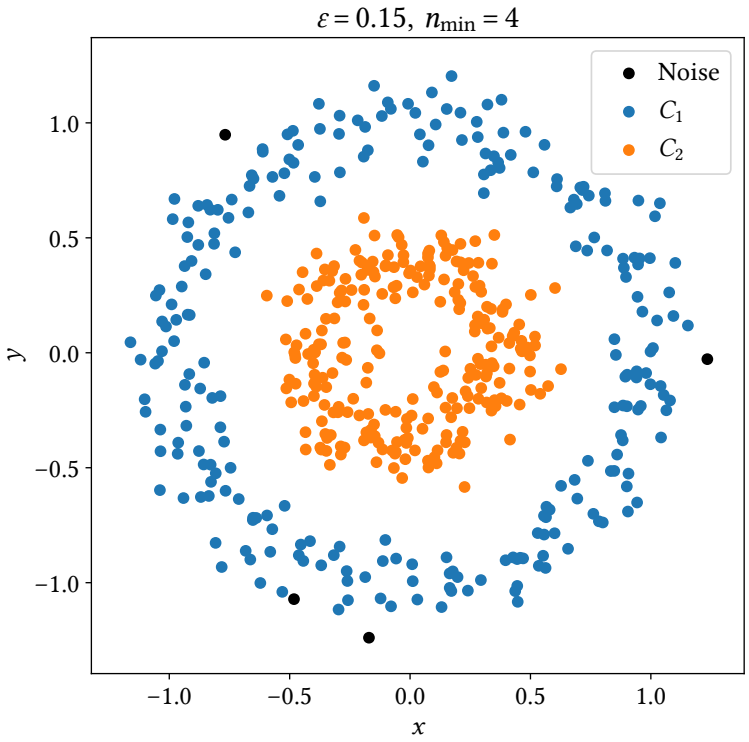


Figure 4: Clustering result of DBSCAN applied to an artificial circle dataset.

Table 2: Variable names for the notebook `DBSCANParameters.ipynb` (Exercise 3). You have to name your variables exactly as noted here so that the test script can check their content.

Name	Type	Description
`k_dists`	Vector	Distance d_i to the k-th neighbour for each data point $\boldsymbol{p}_i$.
`labels`	Vector	Cluster label ω_i for each data point $\boldsymbol{p}_i$.