Warsaw University of Technology
Project teacher: dr. Robert Bembenik
Author: Mateusz Węcławski

EDAMI 2024
Neighbourhood-based clustering

1. Problem definition and introduction

The goal of this project is to implement an efficient algorithm for neighbor-based clustering inspired by the work of Zhou et al. (2005). This method aims to group data into meaningful clusters, which is important for both data mining and artificial intelligence. Their approach identifies clusters using proximity relationships while incorporating neighborhood information to enhance clustering accuracy and reduce computational complexity. Additional advantages are that their NBC algorithm needs fewer input parameters than existing NBC algorithms, and that it is effective in discovering clusters of arbitrary shapes and different densities.

2. Description of the proposed algorithm/solution with reference to the literature

Basic concepts:

- To measure the distance in dataset D between points p and q (denoted as dist (p,q)) we use Euclidean distance.
- knB(p) - k's nearest neighbourhood of p

- | knB(p) | - number of objects being a member of k-nearest neighbourhood of p
- R-knB(p) - set of objects whose knB contains p
- | R-knB(p) | - number of objects taking p as a member of their k-nearest neighbourhood
- NDF (Neighbourhood density factor) is calculated as follows: NDF(p) = | R-kNB(p) | / | kNB(p) |
- DP - local dense point. Object p is a local dense point, if its NDF(p) is greater than 1. The larger NDF(p) is, the denser p's k neighbourhood is.
- SP - local sparse point. Object p is a local sparse point, if its NDF(p) is less than 1. The smaller NDF(p) is, the sparser p's k neighbourhood is.
- EP - local even point. Object p is a local even point, if its NDF(p) is equal (or approximately equal) to 1.

  It can be also written as following:

  Dense Points (DP): Objects where NDF(p) > 1.

  Sparse Points (SP): Objects where NDF(p) < 1.

  Even Points (EP): Objects where NDF(p) ≈ 1.

- Given two objects p and q in dataset D, p is directly neighbourhood-based density reachable (directly ND-reachable) from q if q is a DP or EP and p is a member of kNB(q)
- Given two objects p and q in dataset D, p is neighbourhood-based density reachable (ND-reachable) if there is a chain of objects p1, … pn, where p1=p and pn = q, such that pi is directly ND-reachable from pi+1
- If object p is ND-reachable from object q, then object q is also ND-reachable from object p
- Given two objects p and q in dataset D, p and q are neighbourhood-density based connected (ND-connected), if p is

ND-reachable from q or q is ND-reachable from p or there is a third object o such that p and q are both ND-reachable from o

- Neighbourhood-based cluster: given a dataset D, a cluster C is a non-empty subset of D that:
    a) For two objects p and q in C, p and q are ND-connected
    b) If p belongs to C and q is ND-connected to p, then q also belongs to C

NBC algorithm consists of two major steps:

1) Evaluating NDF values. Search kNB and R-kNB for each object in the target dataset and then calculate its NDF.
2) Clustering the dataset. Use the NDF values to identify dense regions into clusters.
   More specifically, given a DP or EP object p from dataset D, first find points directly ND-reachable from p. First batch of these objects are in kNB of p, move them into the p's cluster. Then find directly ND-reachable points from each DP or EP in p's kNB, until there is no more potential point that can be added to cluster p. Nextly, do this for the next point DP or EP from the dataset (find new cluster). If there are no more DP or EP points to create clusters, the algorithm comes to an end.
   Points belonging to no cluster are considered noise or outliers.

Pseudocode showcased in the literature [1]:

```
NBC(Dataset, k)
{
      for each object p in Dataset
      p.clst_no=NULL; // initialize cluster number for each object
      CalcNDF(Dataset, k); // calculate NDF
      NoiseSet.empty(); // initialize the set for storing noise
      Cluster_count = 0; // set the first cluster number to 0
      for each object p in Dataset
```

```
{ // scan dataset
   if(p.clst_no!=NULL or p.ndf < 1) continue;
   p.clst_no = cluster_count; // label a new cluster
   DPSet.empty(); // initialize DPSet
   for each object q in kNB(p)
   {
         q.clst_no = cluster_count;
         if(q.ndf>=1) DPset.add(q)
   }
   while (DPset is not empty)
   { // expanding the cluster
         p = DPset.getFirstObject();
         for each object q in kNB(p)
         {
               if(q.clst_no!=NULL)continue;
               q.clst_no = cluster_count;
               if(q.ndf>=1) DPset.add(q);
         }
         DPset.remove(p);
   }
cluster_count++;
}
for each object p in Dataset
{ // label noise
   if(p.clst_no=NULL) NoiseSet.add(p);
}
}
```

3. Description of the implementation

- Programming language: Python 3.7

- Programming environment: Python Idle

Project consists of 3 files

1) DataMiningProj - main file, used to calculate the clusters by nbc and uses the library version of k-means algorithm to use it for later comparison. It generates and saves images of clusters in a loop.

2) PlotMaker - supplement file, generates 2 plots, one that compares time of 2 algorithms, second one that compares the silhouette measurement
3) GifMaker - supplement file, used to generate gif off the images (so that it helps visualize the performance)

Implemented algorithm of nbc in DataMiningProj:

```
def calculate_distance_matrix(self, data):
    return cdist(data, data)

def calculate_kNB_and_R_kNB(self, distance_matrix):
    n = distance_matrix.shape[0]
    kNB = [set() for _ in range(n)]
    R_kNB = [set() for _ in range(n)]
    for i in range(n):
        neighbors = np.argsort(distance_matrix[i])[1:self.k + 1]
        kNB[i].update(neighbors)
        for neighbor in neighbors:
            R_kNB[neighbor].add(i)
    return kNB, R_kNB

def calculate_NDF(self, kNB, R_kNB):
    n = len(kNB)
    NDF = np.zeros(n)
    for i in range(n):
        if len(kNB[i]) > 0:
            NDF[i] = len(R_kNB[i]) / len(kNB[i])
    return NDF
```

```
def cluster(self, data):
    # Step 1: Calculate distances, kNB, R_kNB, and NDF
    distance_matrix = self.calculate_distance_matrix(data)
    kNB, R_kNB = self.calculate_kNB_and_R_kNB(distance_matrix)
    NDF = self.calculate_NDF(kNB, R_kNB)

    # Step 2: Clustering based on NDF
    n = data.shape[0]
    cluster_labels = [-1] * n  # Initialize all points as unassigned
    cluster_count = 0  # Start cluster count
    noise_set = set()

    for i in range(n):
        if cluster_labels[i] != -1 or NDF[i] < 1:  # Skip non-DP/EP points
            continue

        # Assign a new cluster
        cluster_count += 1
        cluster_labels[i] = cluster_count
        DP_set = set()

        # Expand cluster with kNB
        for neighbor in kNB[i]:
            cluster_labels[neighbor] = cluster_count
            if NDF[neighbor] >= 1:  # DP or EP
                DP_set.add(neighbor)

        # Process DP set to expand the cluster
        while DP_set:
            current_point = DP_set.pop()
            for neighbor in kNB[current_point]:
                if cluster_labels[neighbor] == -1:  # Unassigned
                    cluster_labels[neighbor] = cluster_count
                    if NDF[neighbor] >= 1:  # DP or EP
                        DP_set.add(neighbor)

    # Identify noise points
    for i in range(n):
        if cluster_labels[i] == -1:
            noise_set.add(i)

    return cluster_labels, noise_set
```

4. User's Manual

To run the algorithm, just run module (F5). There is no user input to be made in the console or in GUI. To change the dataset, just change the file name inside the script.

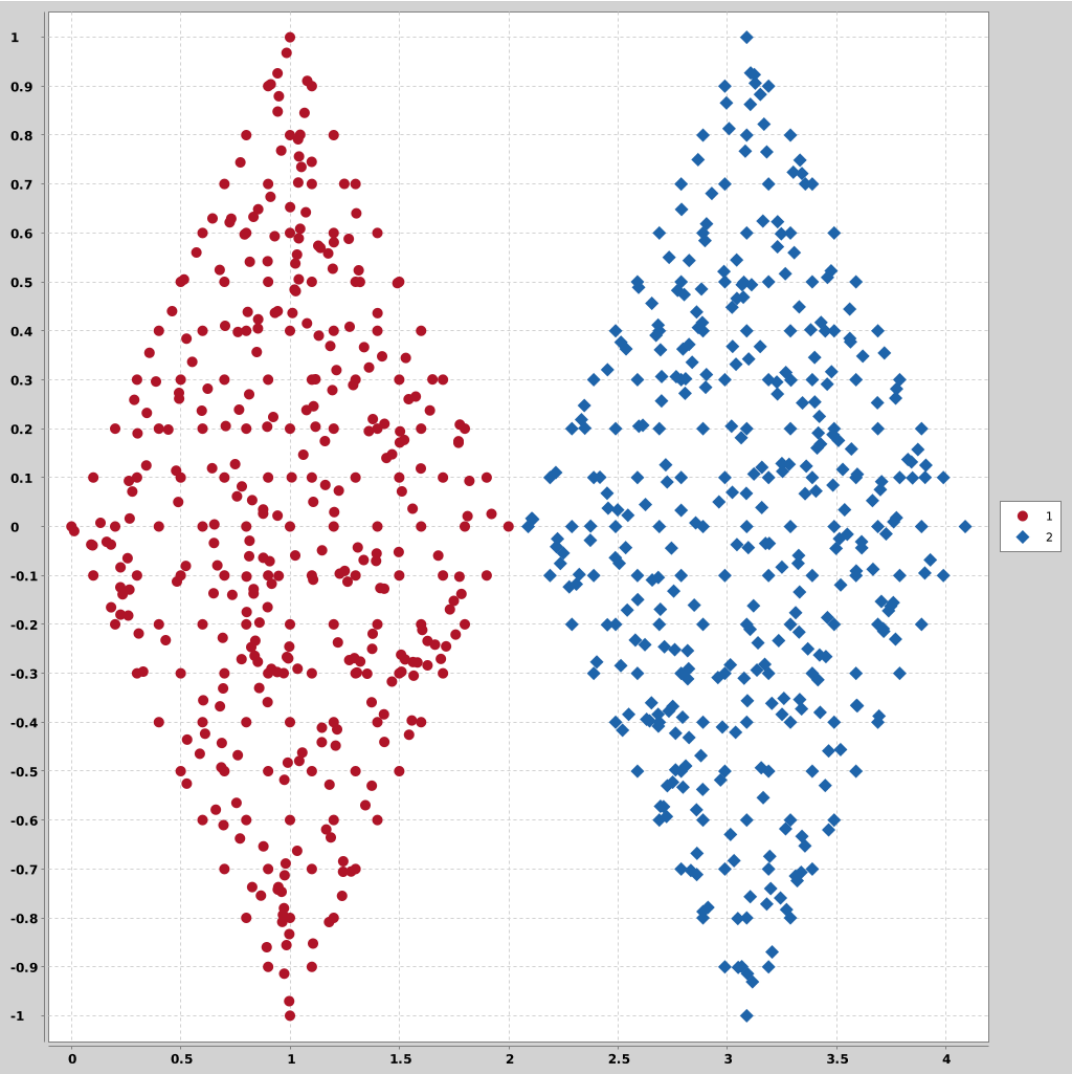
- To run my implemented algorithm, it is necessary to have: (**Python version 3.7 or higher.**)
- The following libraries installed:
  - **numpy** for matrix calculations.
  - **matplotlib** for graphical visualisation.
  - **sklearn** for silhouette score, mds visualisation, kmeans algorithm for comparison - I did not use any parts of this library for the nbc algorithm.
  - **Pandas** for reading datasets
  - **PIL, glob** for correct gif creation
  - **Os** for measuring time
  - **Arff** for opening .arff files
  - **Scipy** for measuring distance between 2 points

5. Description of the used datasets

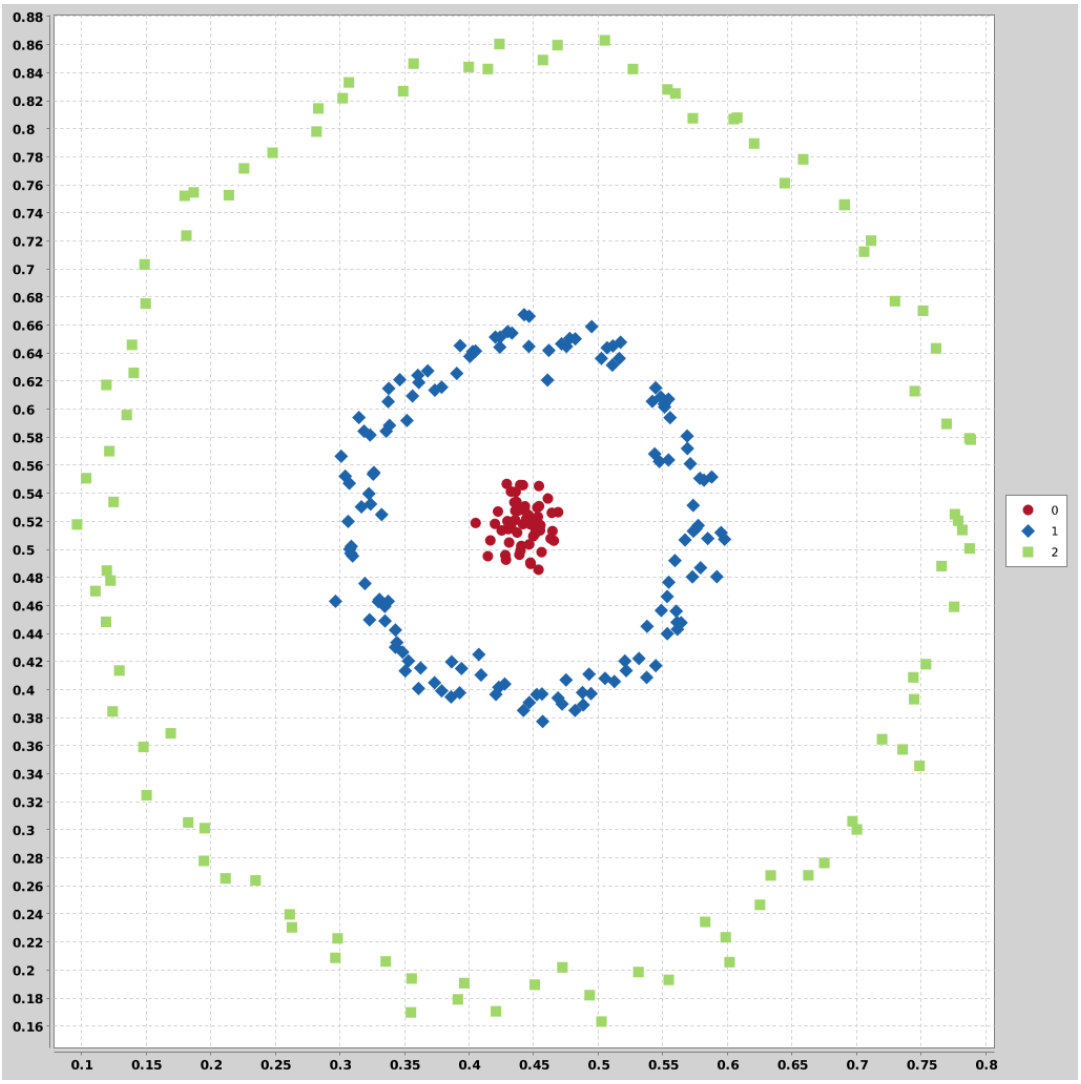
<Disclaimer: I used gifs to present created clusters, so it is the best to see this documentation in the docx format, as pdf format makes these gif files stationary and only show the first frame (k=2)>

In experiments, I used few artificially created datasets commonly used for clustering benchmarks, available on: https://github.com/deric/clustering-benchmark?tab=readme-ov-file

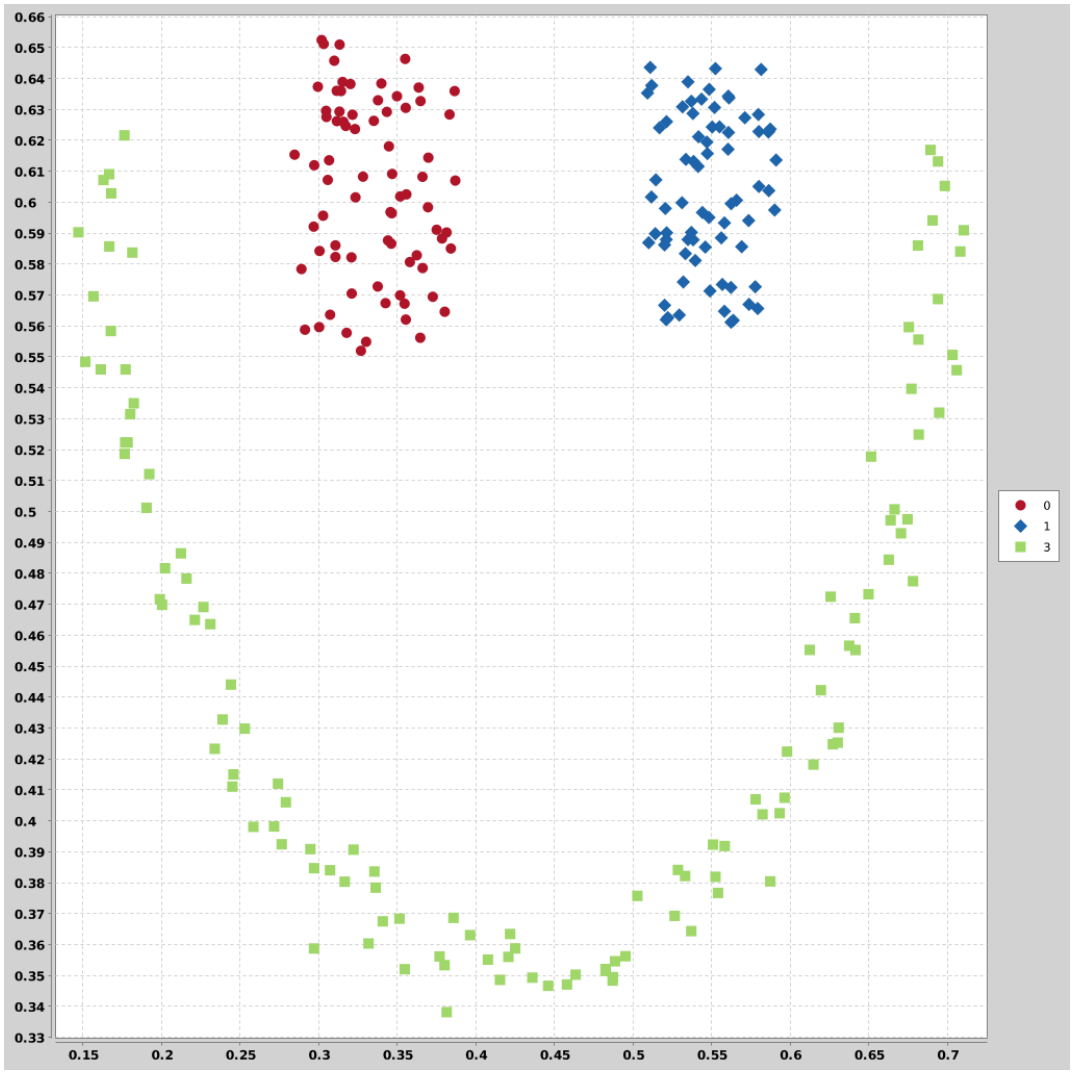
1) “Two diamonds”



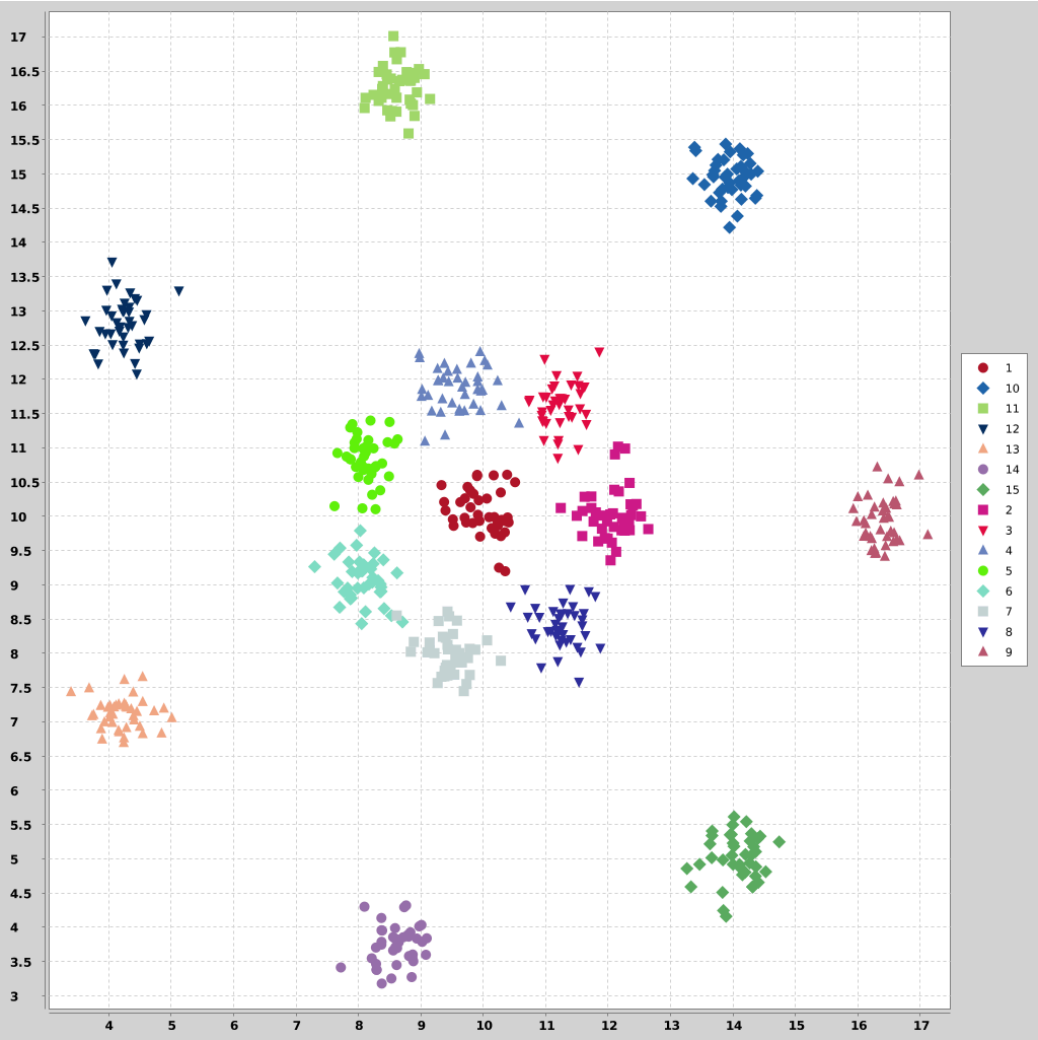
2) “Zelnik 1”


0.88
0.86
0.84
0.82
0.8
0.78
0.76
0.74
0.72
0.7
0.68
0.66
0.64
0.62
0.6
0.58
0.56
0.54
0.52
0.5
0.48
0.46
0.44
0.42
0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.2
0.18
0.16
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0
1
2

3) “Zelnik 3”


0.66
0.65
0.64
0.63
0.62
0.61
0.6
0.59
0.58
0.57
0.56
0.55
0.54
0.53
0.52
0.51
0.5
0.49
0.48
0.47
0.46
0.45
0.44
0.43
0.42
0.41
0.4
0.39
0.38
0.37
0.36
0.35
0.34
0.33
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0
1
3

4) “R15”

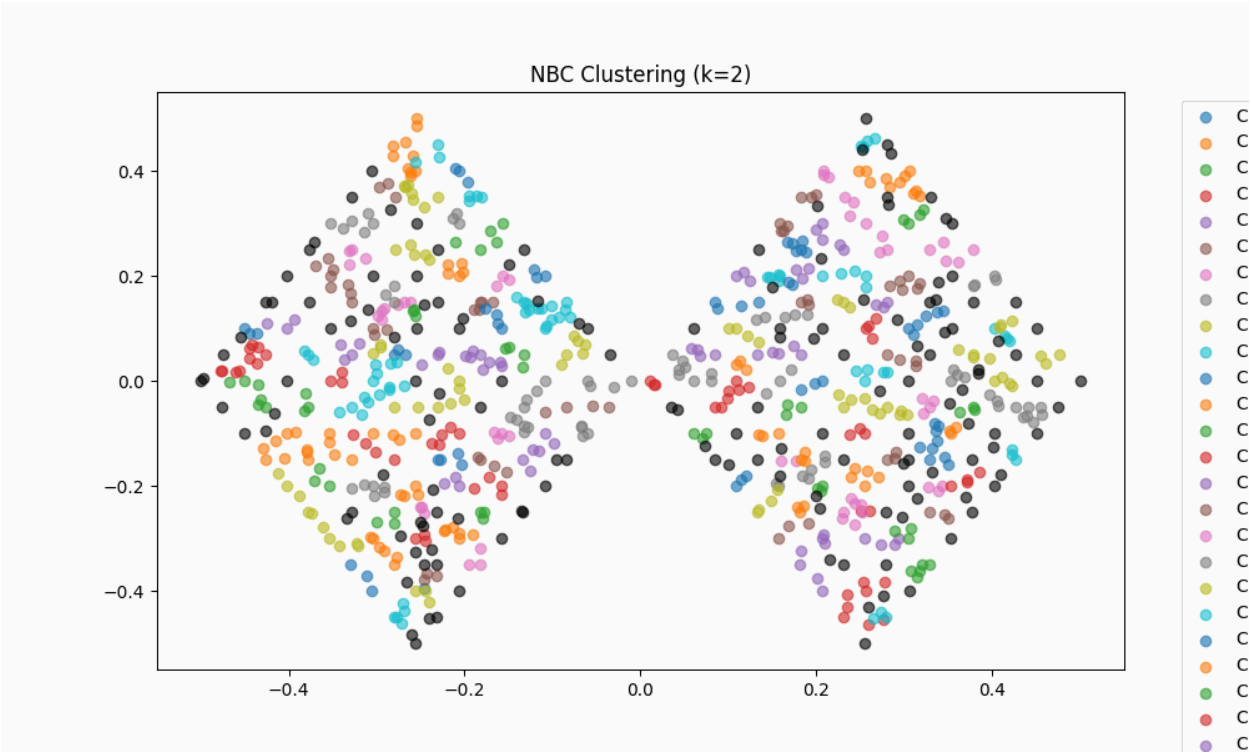


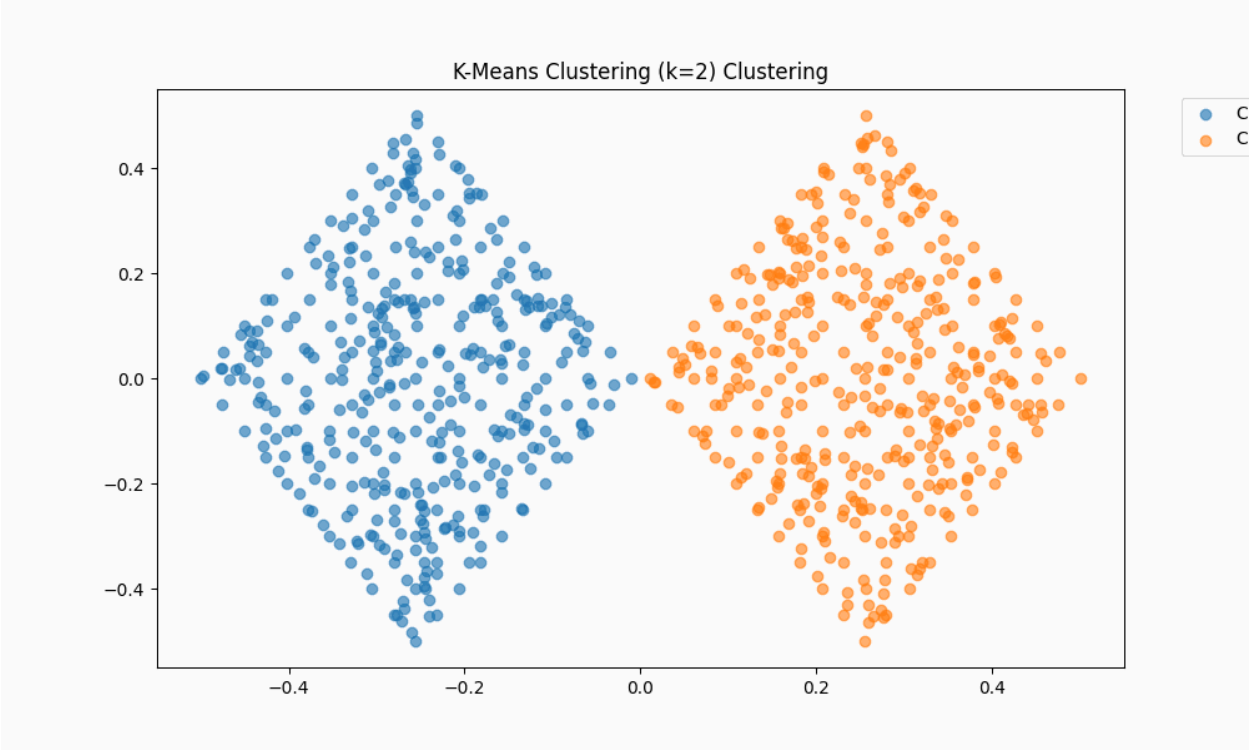
6. Experimental results

Conducted experiments:

I compared my implementation of the nbc algorithm with the publicly available k-means algorithm. I also test different values of k, from 2 to 19.

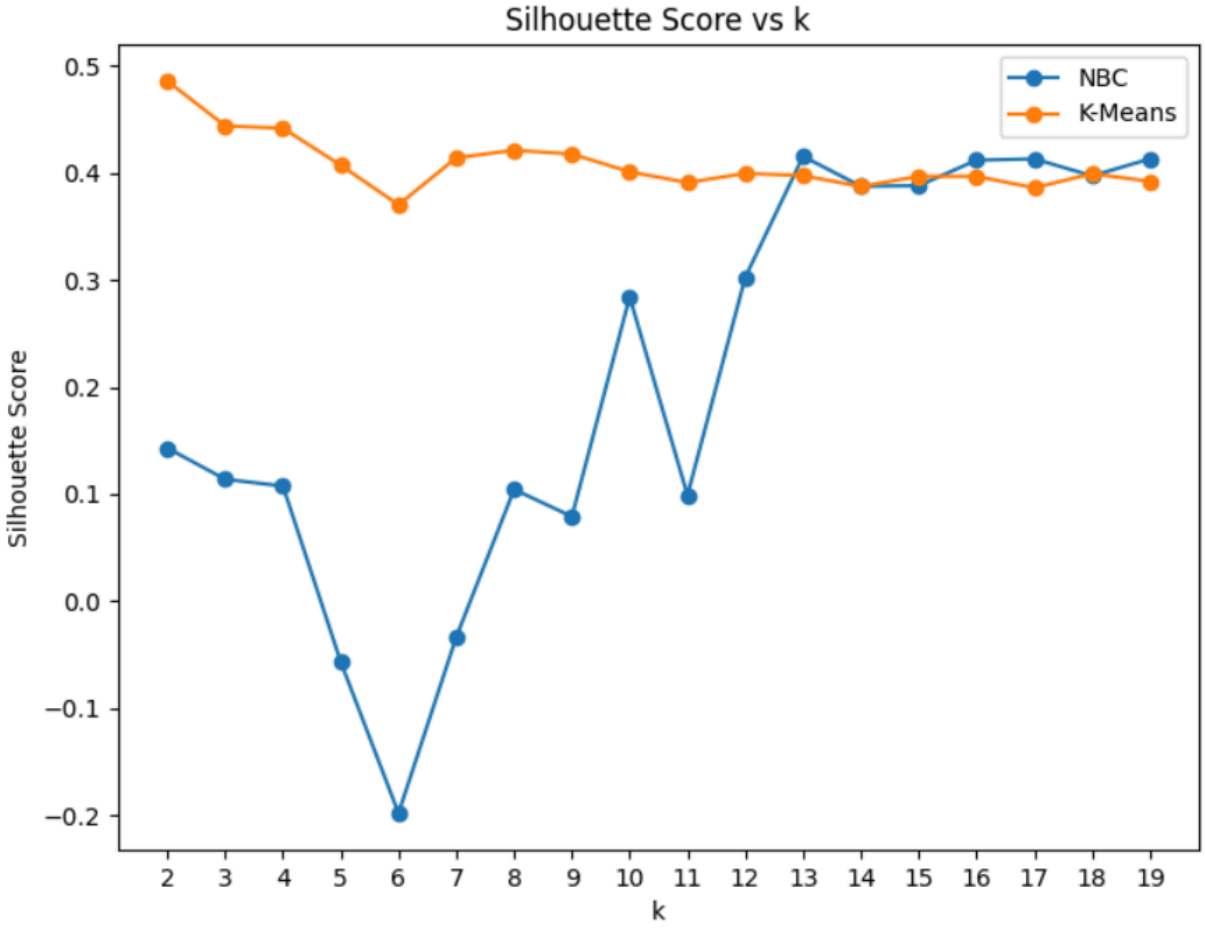
1) Two diamonds dataset

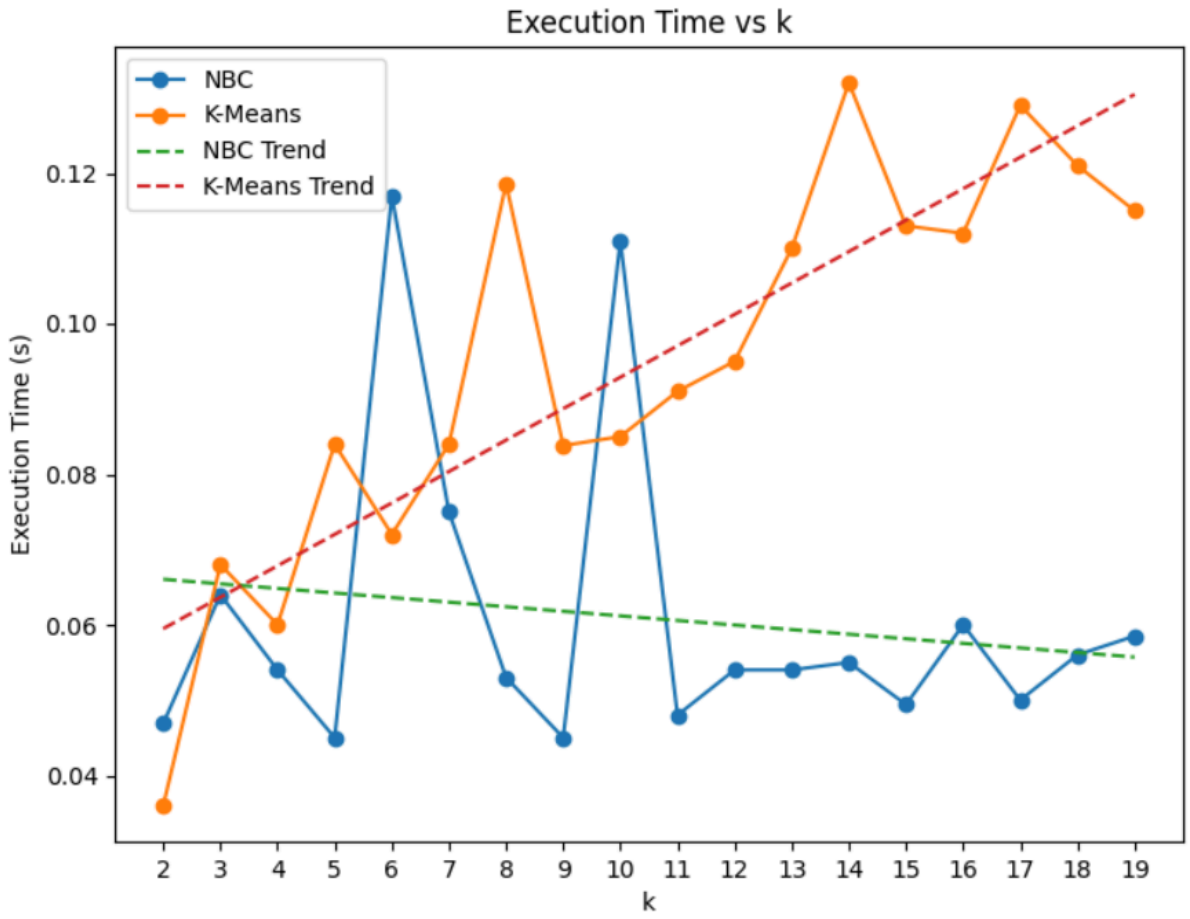




The k-means algorithm produced optimal result with just k=2, where the nbc algorithm needed k=13, but still gave worse result due to noise classification (for greater values there was still some points classified at

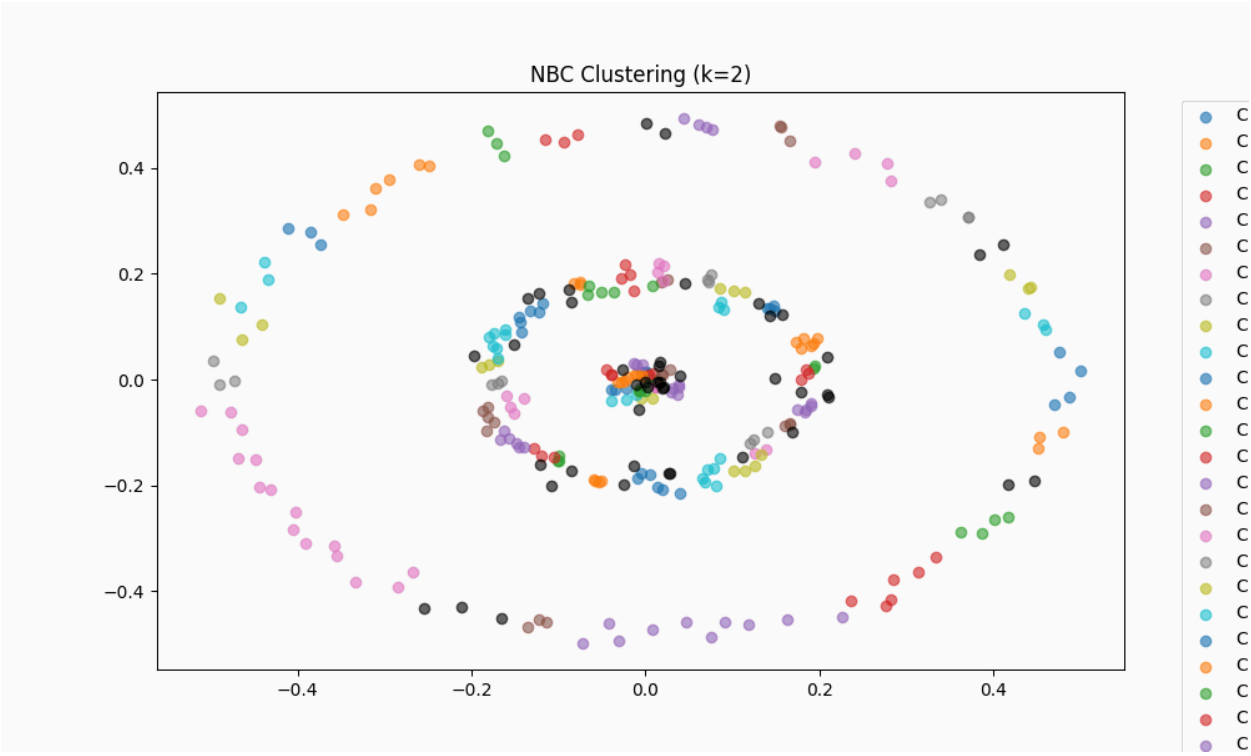
noise, but the difference was mainly that the amount of them varied slightly (1-3 points) and which points were classified as noise.

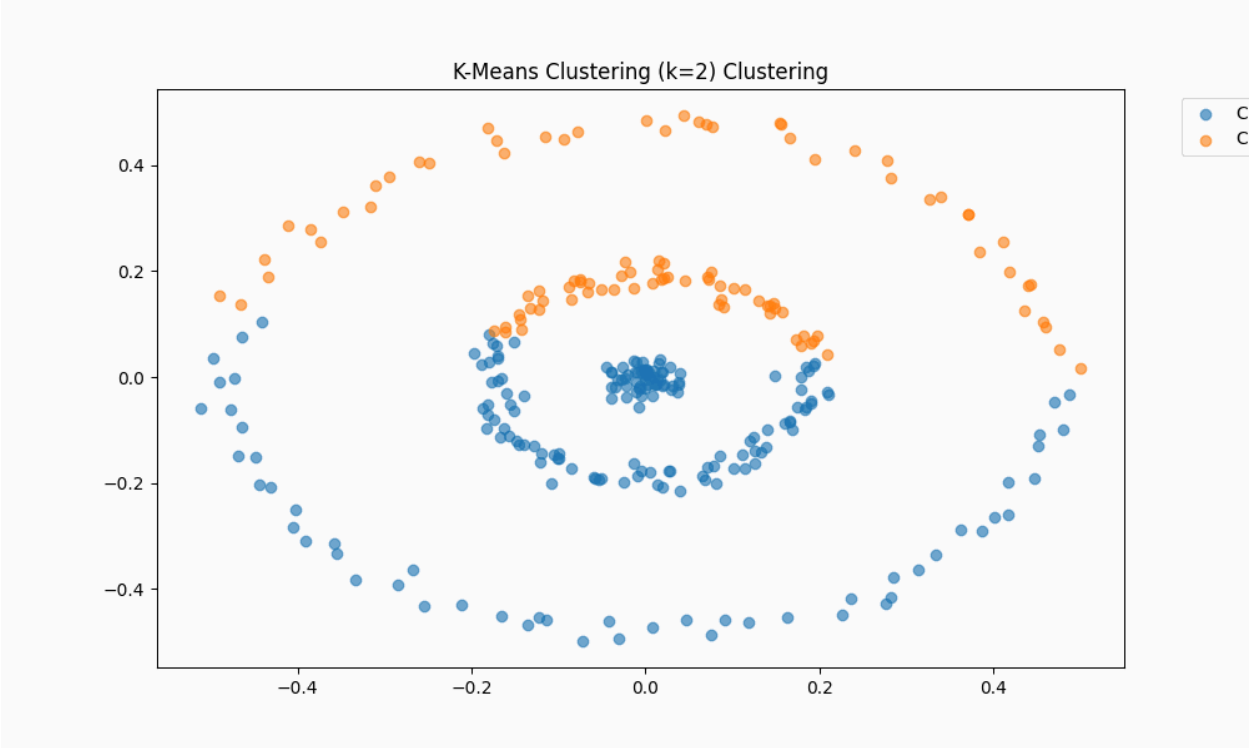




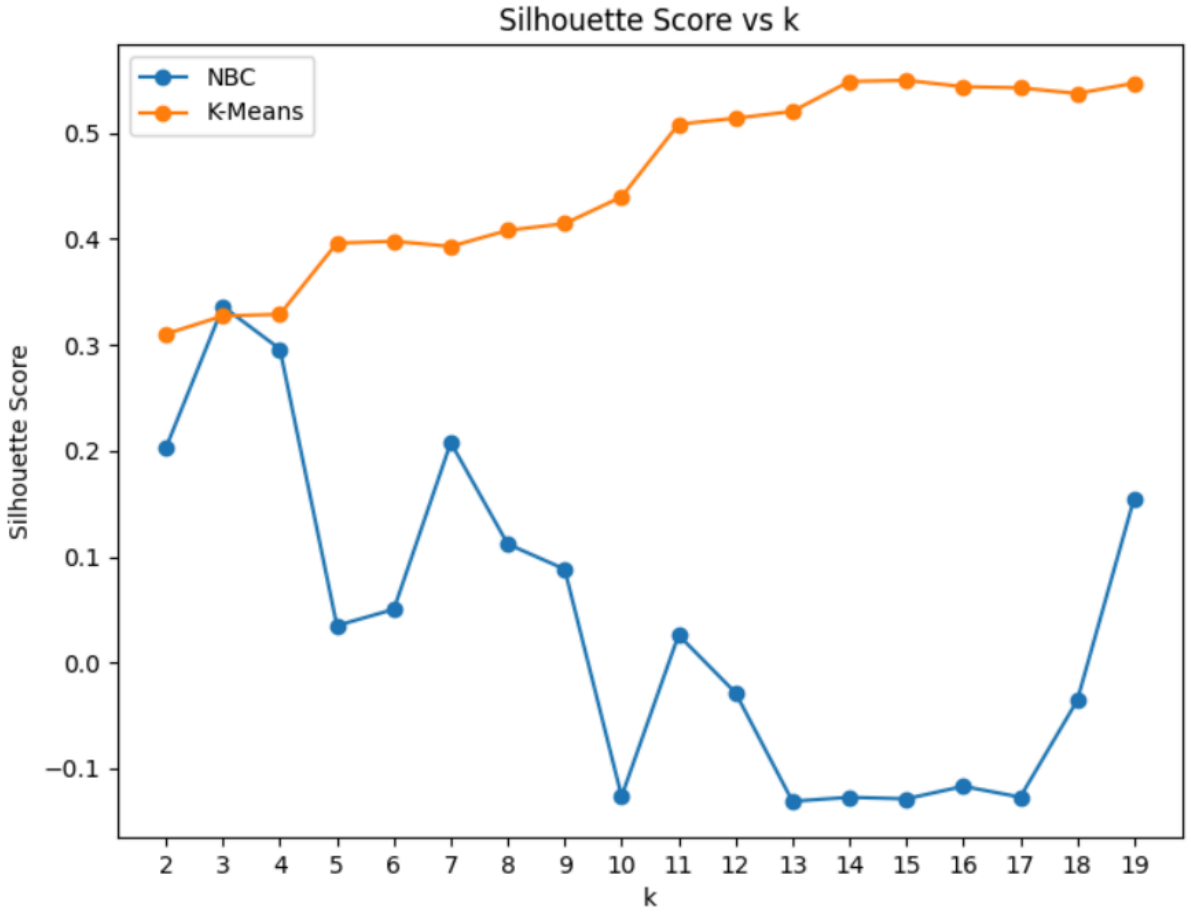
Nbc algorithm performs on average faster, especially with increasing k, but I still mark the k-means as better for this dataset.

2) Zelnik 1

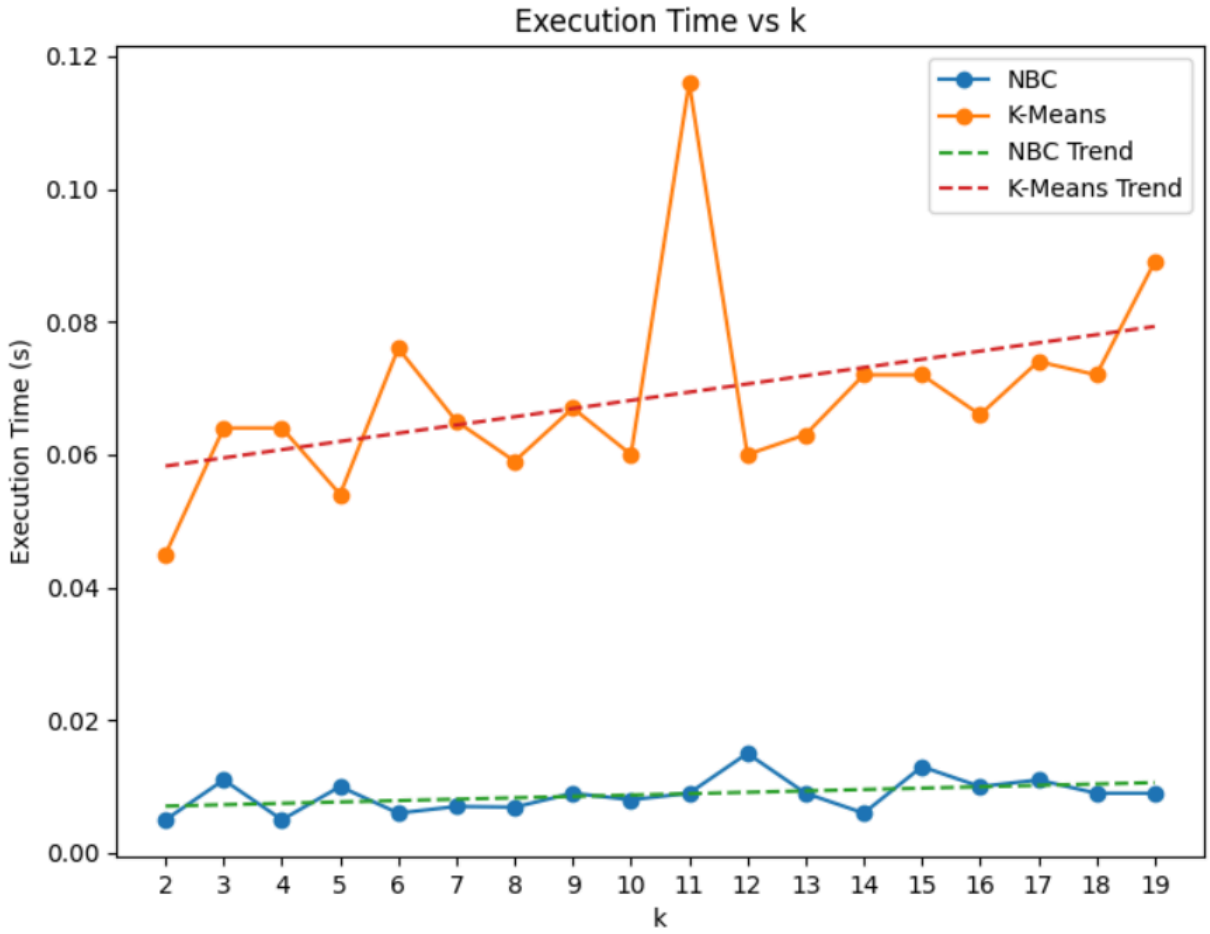




For this dataset there is a clear winner - nbc. The k-means clustering couldn’t generate optimal results for all of the k values I tested, where the nbc algorithm generated optimal results (with some noise) at k=13.

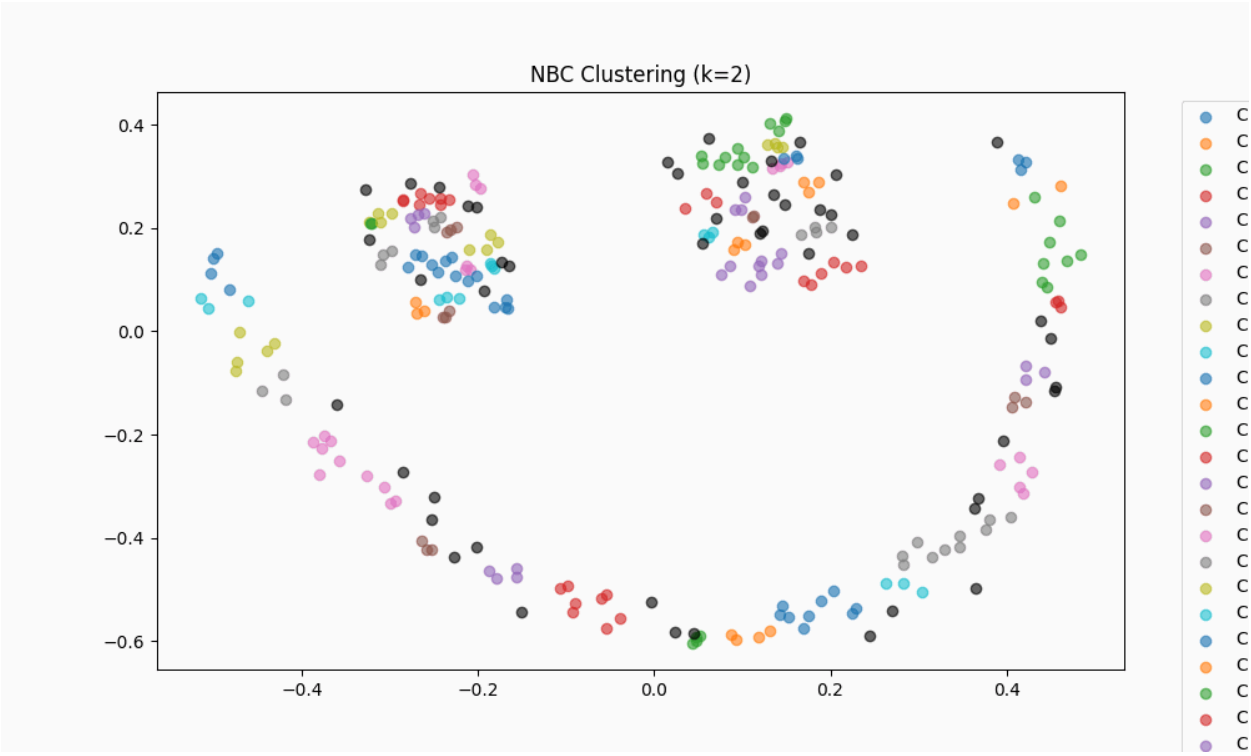


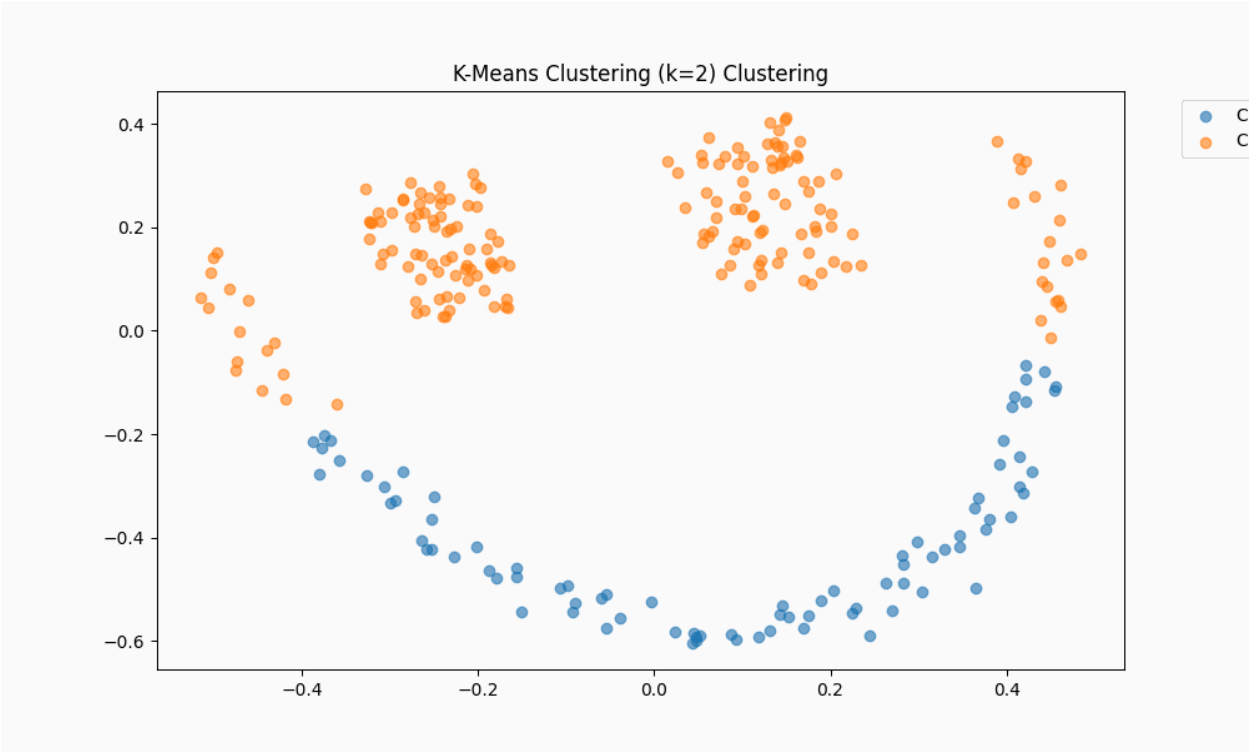
During experimenting with this dataset, I discovered that the silhouette score measure is not a meaningful way to measure these 2 methods, as it says that NBC performed worse, when it clearly produced optimal results. From now on, I will not use this metric.



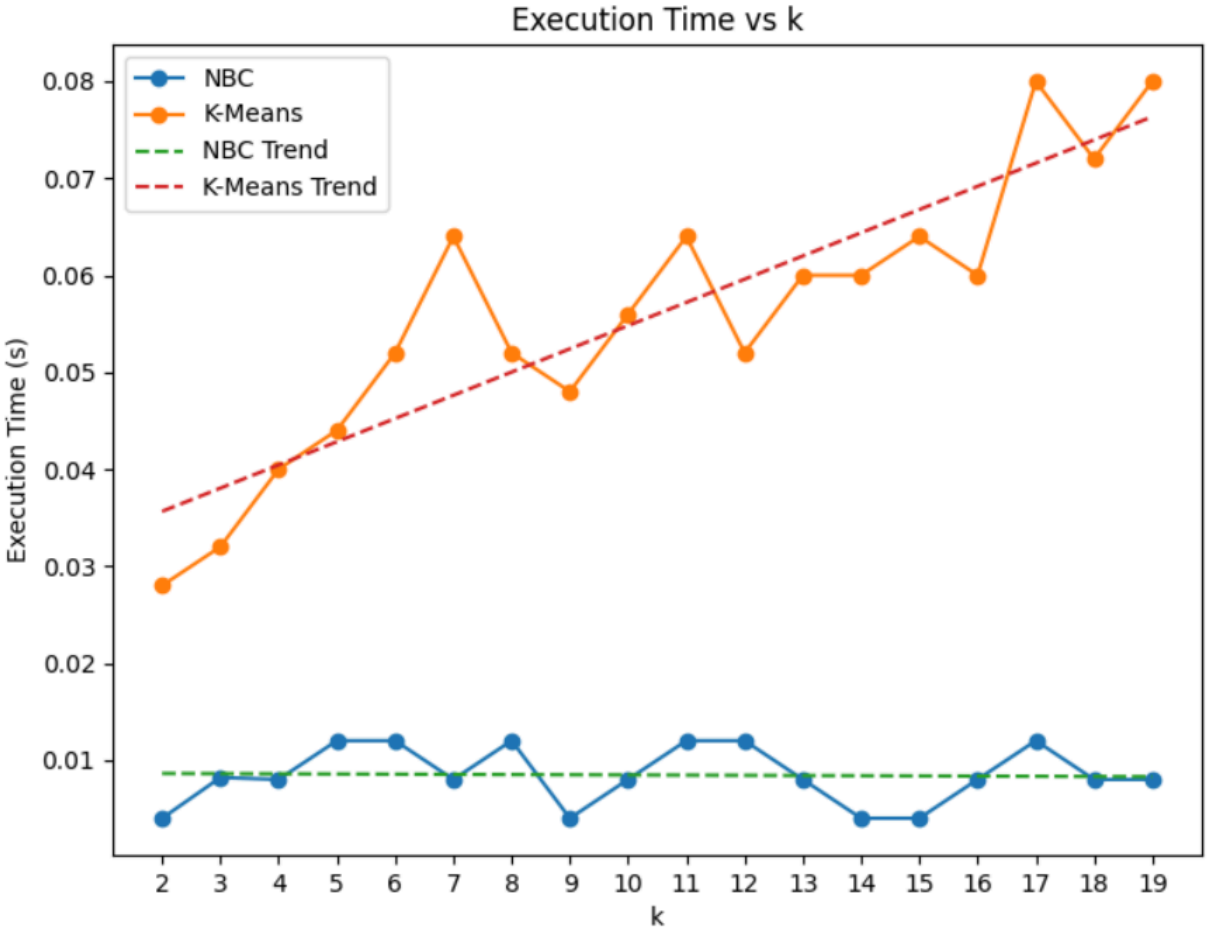
The execution time difference is even greater for this dataset.

3) Zelnik 3



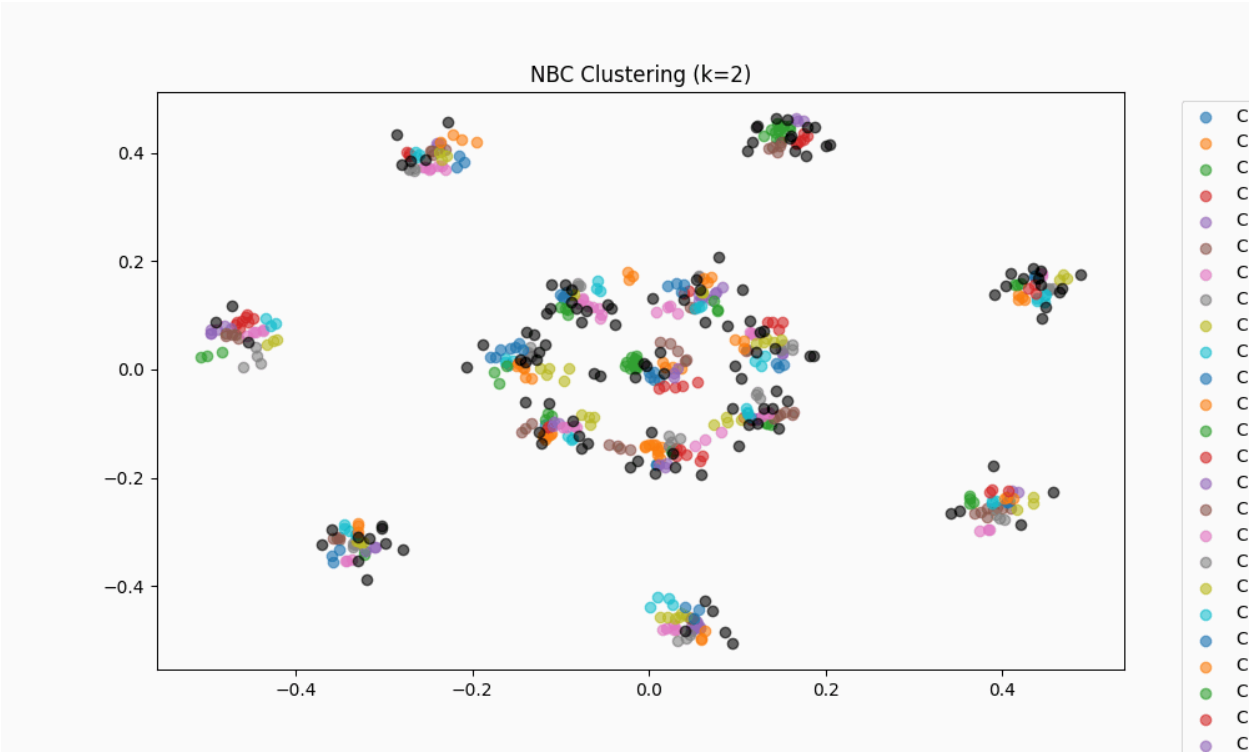


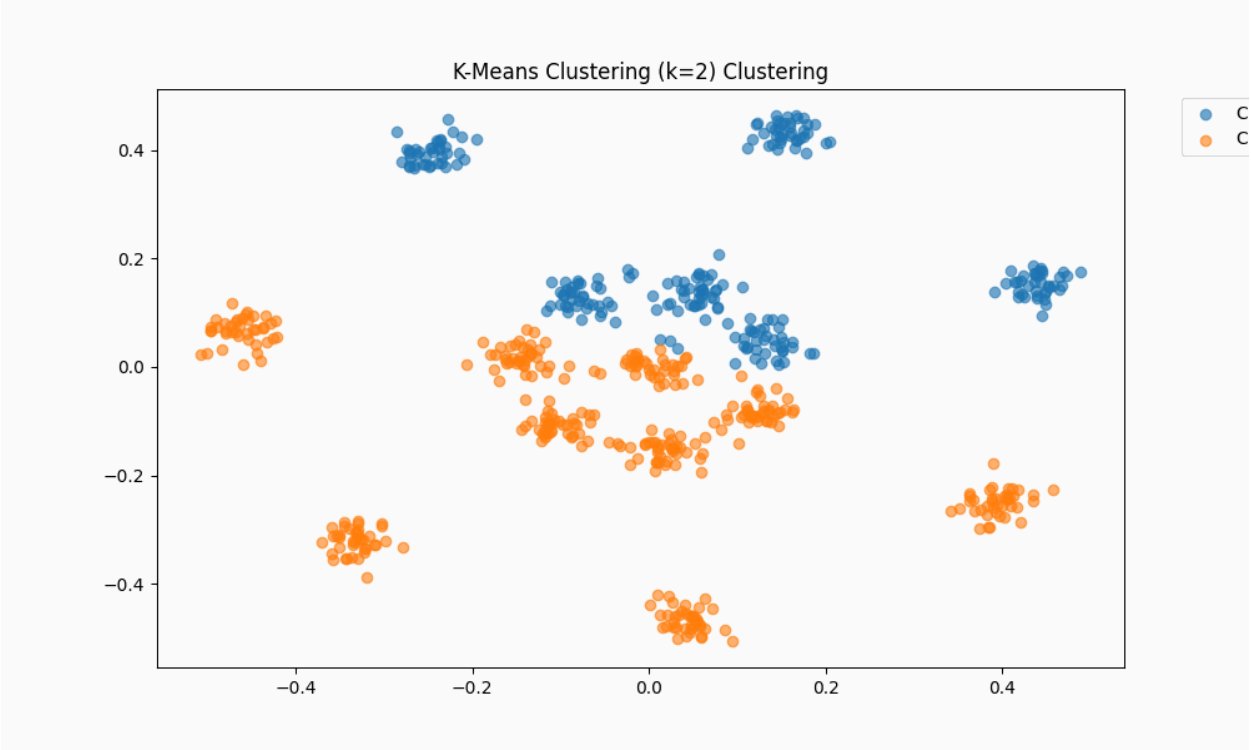
The obtained results are similar to zelnik1, k-means could not generate the desired result (closest one was k=3), where nbc generated adequate result for k=17.



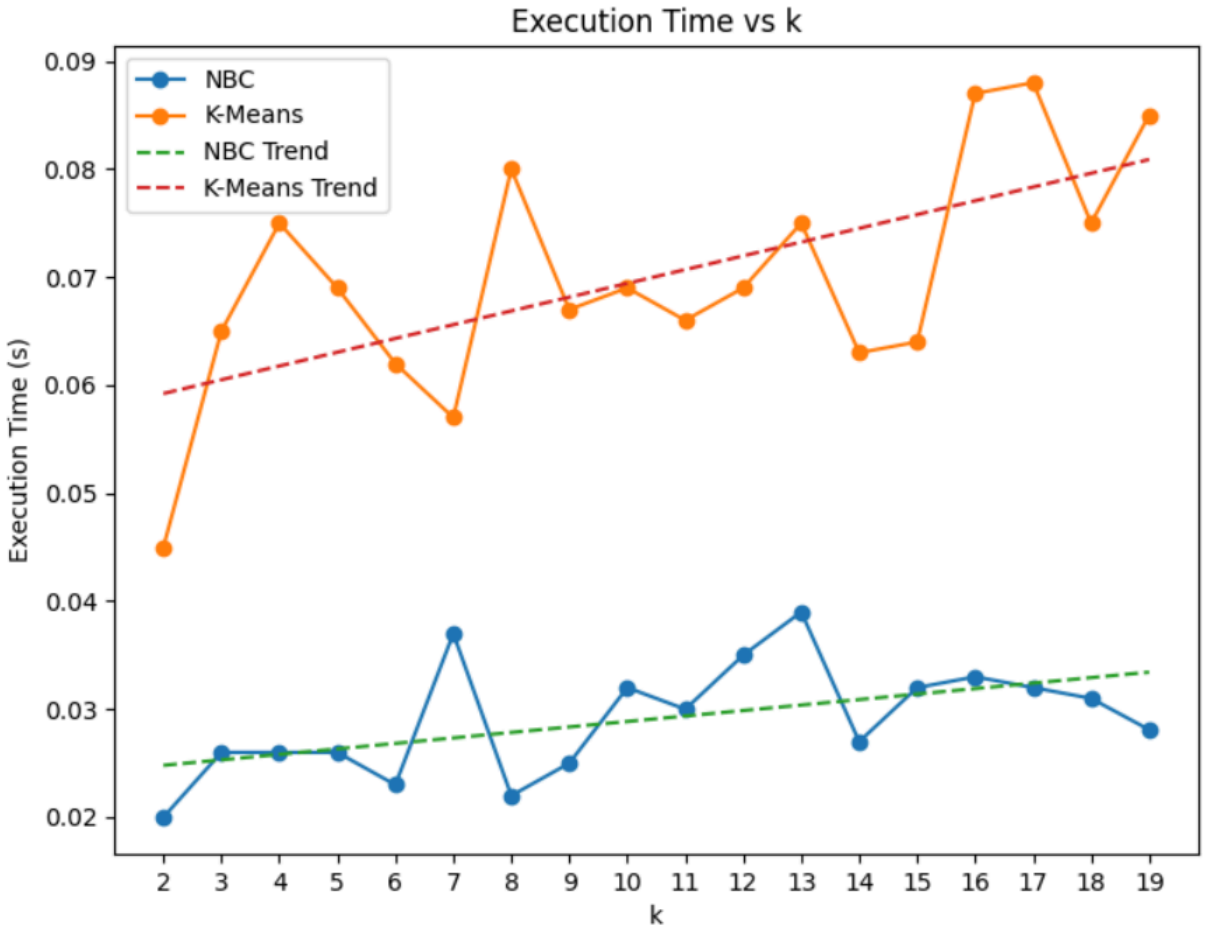
Just like before, the nbc confirms to be a faster algorithm.

4) R15





Similar scenarios as the 2 before - nbc can generate good enough results -starting at k=8 (there are some noise points), but k-means struggles to do so.



7. Conclusions

For 4 tested datasets, the nbc algorithm was better 3 times. It made clusters really close to the desired ones. The main visible problem on the plot is the noise. It performed worse on the ideally divided data (2 diamonds). Execution time is notably faster and the trend line shows that with increasing k value, the time is not going to grow at large exponential time. It can be a major advantage to use this algorithm when working on large datasets. It performed worse in terms of obtained results in the “easy dataset” - 2 ideally or almost ideally divided clusters. Taking that into account, I think the nbc algorithm should be performed on larger and more complex datasets - many clusters, irregular sizes, shapes, etc.

8. Bibliography

[1] Shuigeng Zhou, Yue Zhao, Jihong Guan, Joshua Zhexue Huang: A Neighborhood-Based
Clustering Algorithm. PAKDD 2005: 361-371