Hanoi University of Science and Technology

School of Information and Communication Technology

Graduation Research II Report

Topic: Graph-based Clustering in Social Networks Analysis

Supervisor: Assoc. Prof. Than Quang Khoat

Academic year: 2021/2022

Student: Do Van Khai

Student ID: 20176790

Date: January 2021

Contents

1 Introduction

1.1 Overview

Cluster Analysis or Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than those in other groups (clusters). It is a main task of exploratory data analysis.

A large number of clustering techniques have been developed in a variety range of domains for different type of applications. However, None of them are suitable for all types of data, clusters, applications. In reality, there is always have a chance for new clustering algorithm that is suitable, more efficiency for a specific kind of data.

In the topic of Graduation Research II, I have researched about a Clustering technique that are suitable for Social Networks Analysis. And this technique is called: **Graph-based Clustering**.

1.2 About Graph-based Clustering

[3] Graph-based Clustering is a family of unsupervised classification algorithms that are designed to cluster the vertices and edges of graph instead of objects in a feature space.

The application field of this technique is in the Data Mining of Social Networks or the Web graph.

2 Graph-based Clustering Approaches

2.1 Key Approaches of Graph-based clustering [1]

The following are 4 key approaches of Graph-based clustering technique, with each approach, it is employed by some algorithms:

1. **Sparsify the proximity graph** to keep only the connections of an object with its nearest neighbor. This sparsification is useful for handling noise and outliers. It also allows the use of highly efficient graph partitioning algorithms that have been developed for sparse graphs.
   Two algorithms use this approach are: **MST, OPOSSUM**.

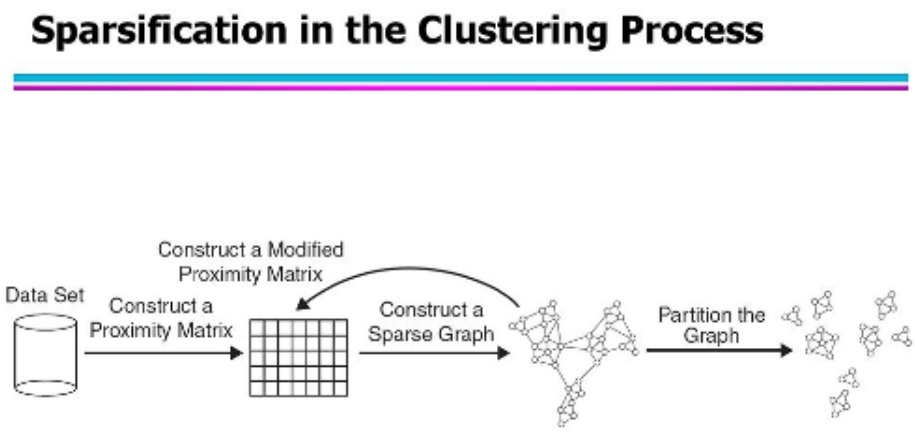


Figure 1: Sparsification

Benefit:

- Data-size is reduced : Sparsification can often eliminate more than 99% of entries in a proximity matrix.
- Clustering may work better :The Sparsification technique keep the connections to their nearest neighbors of an object while breaking the connections to more distant objects. (nearest-neighbor principle to reduce noise, outliers, sharpens distinction between cluster)

2. **Clustering Based on SNN Similarity Computational**. This approach, which is based on the observation that an object and its nearest neighbors usually belong to the same class, so it is useful for overcoming problems with high dimensionality and

clusters of varying density.
The Algorithm in this approach: **Javis-Patrick Clustering**
Benefits:

- Useful for overcoming problems with high dimensionality and clusters of varying density

3. **Define core objects and build clusters around them**. To do this for graph-based clustering, it is necessary to introduce a notion of density-based on a proximity graph or a sparsified graph. As with DBSCAN, building clusters around core objects leads to a clustering technique that can find clusters of different shapes and sizes.
The Algorithm in this approach: **SNN density-based**
Benefits:

- Can find clusters of different shapes and sizes.

4. **Use the information in the proximity graph** to provide a more sophisticated evaluation of whether two clusters should be merged. Specifically, two clusters are merged only if the resulting cluster will have characteristic similar to the original two clusters.
The Algorithm in this approach: **CHAMELEON**

Now, I will go though some algorithms that are mentioned above.

2.2 OPOSSUM Clustering

OPOSSUM is a clustering technique that was specifically designed for clustering sparse, high dimensional data, such as document or market basket data. It performs based on the sparsification of a proximity graph. To do sparsification, OPOSSUM use the METIS algorithm to partitioning sparse graphs.

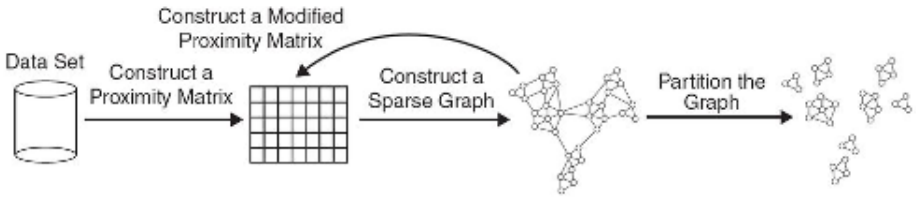


Figure 2: OPOSSUM algorithm

2.2.1 Algorithm

1. Compute a sparsified similarity graph.
2. Partitioning the similarity graph into k distinct components (clusters) using METIS.

The similarity measures used are those appropriate for sparse, high dimensional data such as Jaccard measure or Cosine measure.
In the partitioning step, it uses METIS to separate graph into k-distinct clusters. With k is user-specify parameter.
In order to (1) minimize the weight of the edges between components and (2) fullfill a balance constraints:

- **Sample balanced**: The number of objects in each cluster must be roughly the same.
- **Value balanced**: The sum of the attribute values must be roughly the same. It is useful when the attribute values represent cost of an item.

2.2.2 Strengths and Limitations

Stregths:

- Simple and fast
- Clusters generated are nearly equal-size (it also can be a weakness)

Limitations

- Because clusters are constrained to be roughly equal-size, clusters can be broken or combined.
- If we generate too many clusters, they are just a piece of a larger clusters.
- similar to the Initial step of CHAMELEON Algorithm

2.2.3 METIS Algorithm[2]

We will discuss about the METIS Algorithm. This algorithm is not only used in OPOSSUM, but also in CHAMELEON Algorithm.

Definition:
METIS is a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. The algorithms implemented in METIS are based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes developed in our lab.

METIS's key features:

- **Provides high quality partitions**:
  Experiments on a large number of graphs arising in various domains including finite element methods, linear programming, VLSI, and transportation show that METIS produces partitions that are consistently better than those produced by other widely used algorithms. The partitions produced by METIS are consistently 10% to 50% better than those produced by spectral partitioning algorithms.
- **It is extremely fast**:
  Experiments on a wide range of graphs has shown that METIS is one to two orders of magnitude faster than other widely used partitioning algorithms. Graphs with several millions of vertices can be partitioned in 256 parts in a few seconds on current generation workstations and PCs.
- **Produces low fill orderings**:
  The fill-reducing orderings produced by METIS are significantly better than those produced by other widely used algorithms including multiple minimum degree. For many classes of problems arising in scientific computations and linear programming, METIS is able to reduce the storage and computational requirements of sparse matrix factorization, by up to an order of magnitude. Moreover, unlike multiple minimum degree, the elimination trees produced by METIS are suitable for parallel direct factorization. Furthermore, METIS is able to compute these orderings very fast. Matrices with millions of rows can be reordered in just a few seconds on current generation workstations and PCs.

2.3 CHAMELEON Clustering

[4] CHAMELEON consits of 3 steps: sparsification, graph partitioning, and hierarchical clustering.

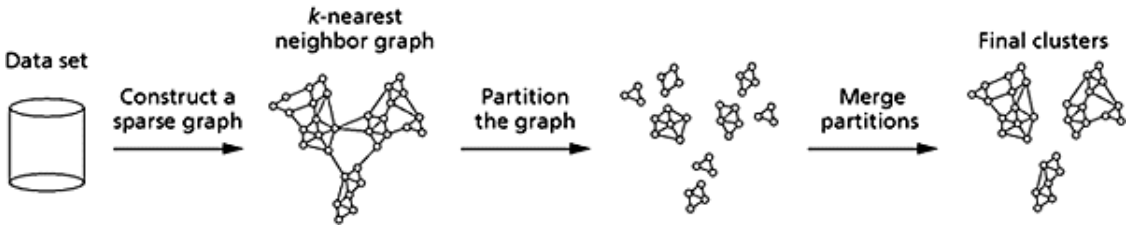


Figure 3: Chameleon clustering

2.3.1 Algorithm

1. Build a k-nearest neighbor graph.
2. Partitioning the graph using a multilevel graph partitioning.
3. **repeat**
4. Merge the clusters that best preserve the cluster self-similarity with respect to relative interconnectivity and relative closeness.
5. **until** No more clusters can be merged.

2.3.2 Explain

Sparsification:
This is the first step. In this step, we generate k-nearest neighnor graphs. A node only has links to its K nearest neighbors. This step helps to reduce the effects of noise, outliers and improve the computational efficiency.

Graph Partitioning
Once sparsified graph has been obtained, CHAMELEON uses METIS to partition the data-set. It starts with an all-inclusive graph (original graph)and then bisects the largest current subgraph (cluster) until no cluster has more than MIN_SIZE points, where MIN_SIZE is a user specified paramter. This process results in a large number of roughly equally size groups of well connected vertices (highly similar data points). The goal is to ensure that each partition contains object mostly from one true cluster.

Agglomerative Hierarchical Clustering
In this step, Chameleon merges clusters from previous step depends on self-similarity. Chameleon can be parameterized to merge more than one pair of clusters in a single step, and stop before

all objects have been merge into a single cluster.

About Self-similarity:

- Relative Closeness (RC)
  RC is the absolute closeness of two clusters normalized by the internal closeness of the clusters. Two clusters are combined only if the points in the resulting cluster are almost as close to each other as in each of the original clusters. Mathematically,

  $$RC = \frac{\tilde{S}_{EC}(C_i,C_j)}{\frac{m_i}{m_i+m_j}\tilde{S}_{EC}(C_i)+\frac{m_i}{m_i+m_j}\tilde{S}_{EC}(C_j)}$$

  Where:

  - m_i and m_j are the size of clusters Ci, Cj, respectively
  - $\tilde{S}_{EC}(C_i, C_j)$ is the average weight of the edges (of the k-nearest neighbor graph) that connect cluster Ci, Cj.
  - $\tilde{S}_{EC}(C_i)$ is the average weight of edges if we bisect cluster C_i
  - $\tilde{S}_{EC}(C_i)$ is the average weight of edges if we bisect cluster C_j

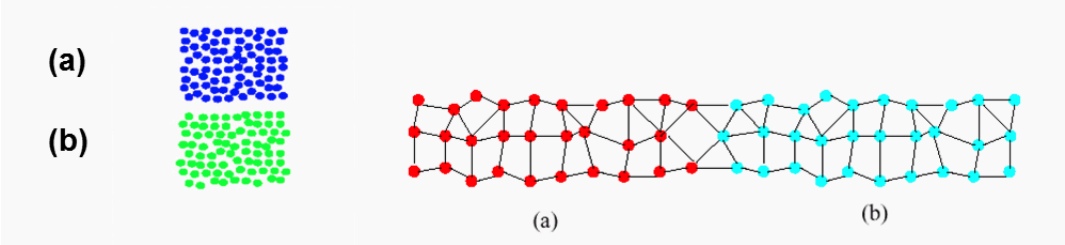


Figure 4: Closeness schemes will merge (a) and (b)

- Relative Interconnectivity (RI)
  RI is the absolute interconnectivity of two clusters normalized by the internal connectivity of the clusters. Two clusters are combined if the points in the resulting cluster are almost as strongly connected as points in each of the original clusters. Mathematically,

  $$RI = \frac{EC(C_i,C_j)}{\frac{1}{2}(EC(C_i)+EC(C_j))}$$

  Where:

  - $EC(Ci, Cj)$ is the sum of edges (of the k-nearest neighbor graph) that connect clusters Ci and Cj
  - $EC(C_i)$ is the minimum sum of the cut edges if we bisect cluster C_i

– $EC(C_j)$ is the minimum sum of the cut edges if we bisect cluster C_j

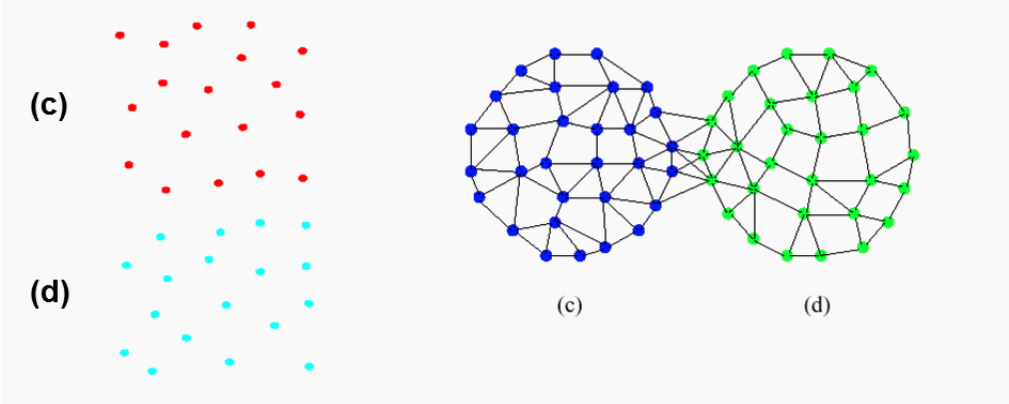


Figure 5: Average connectivity schemes will merge (c) and (d)

- Self-Similarity

  $Self - similarity = RC(C_i, C_j) * RI(C_i, C_j)^{\alpha}$

  α is user specified, typically > 1

2.3.3 Complexity

- Time complexity:
  - Build Graph sparsification: $O(m^2)$
  - Partitioning: $O(mp + m * log(m)$
  - Merge: $O(p^2 * log(p)$

  Where m is number of data points, p is number of partitions

- Space Complexity: $O(km)$, where
  - k: is the k in k-nearest neighbors
  - m: number of data points

2.3.4 Strengths and Limitations

Strengths

- Effective with cluster spatial data, even though noise and outliers are present.

- Can work with clusters are of different shapes, sizes, and density.

Limitations

- Chameleon has problems when the partitioning process does not produce subclusters, as is often the case for high-dimensional data.

2.4 Shared Nearest Neighbor Similarity Concept (SNN similarity)

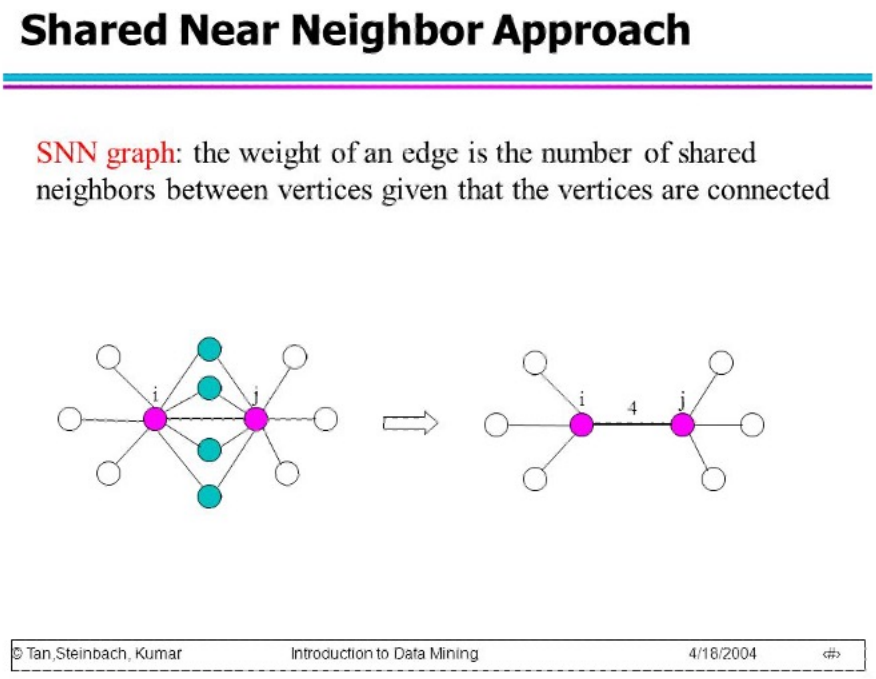


Figure 6: SNN Similarity

2.4.1 Algorithm

This is the algorithm to build a SNN similarity graph

1. identify the k-nearest neighbor of all the sample points.
2. if two points x and y are not in each other's k-nearest neighbor list, then
3. similarity (x, y)$\leftarrow$0
4. else
5. similarity (x, y) $\leftarrow$ the number of shared neighbors
6. end if

2.4.2 Benefits

This concept can solve the problems of Hierrachical clustering approach:

- In hierrachical clustering: Euclidean distance does not work well with the high dimensional dataset.
- Clustering using distance or similarity measure can work better with high dimensional data, like cosine measure. However, it does not completely solve the problems of similarities.
  For example, data points in high dimensional dataset usually have low similarities, so that the data point in different clusters are more likely same as the data point in the same cluster.

With SNN Similarity, it sovle these problems by:

- First, compute K-nearest Neighbor list, computed by original similarity measure
- Then, define a similarity measure based the the number of nearest neighbors that are shared between 2 points.

2.5 Javis-Patrick Clustering

Javis - Patrick clustering using the concept of the Shared Nearest Neighbor Similarity.

2.5.1 Algorithm

1. Compute SNN similarity graph.
2. Sparsify the SNN similarity graph by applying threshold.
3. Find the connected components (Clusters) of Sparsified SNN similarity Graph.

There are 3 inputs to the Algorithms:

- **Distance function**: to calculate the distance between any two given points.
- **k**: number of the closet neighbors to make note of, while forming clusters.
- **kmin**: the minimum number of shared neighbors for any two given points to be clustered.

2.5.2 Complexity

- Time complexity: $O(m^2)$
- Space complexity: $O(km)$

Where m is the number of data points, k is the K nearest neighbors.

2.5.3 Strengths and Limitations

Strengths:

- Perform well when the dataset are: noise, has outliers, clusters of different sizes, shapes, densities.
- Perform well with: high dimensional data
- Work efficiently in the case of finding clusters which have objects linked tightly to each others.

Limitations

- Jarvis-Patrick clustering defines a cluster as a connected component within the SNN similarity graph. Thus, whether a bunch of objects is split into two clusters or left collectively may rely on a single link. Hence, Jarvis-Patrick clustering is somewhat brittle i.e., it may split true clusters or join clusters that ought to be kept separate,
- In this algorithm, not all objects are clustered. However, these objects will be added to existing clusters, and in some cases, there is no requirement for a full clustering. When put next to other clustering algorithms, choosing the simplest and best values for the parameters may be challenging.

2.6 SNN Density-based Clustering

2.6.1 SNN density concept

We have discuss that Euclidean distance is meaningless in the high dimensional data space. And it also the same for Euclidean density. For better using of density, we are going to discuss a new concept of density: ***SNN density***.
SNN density is defined by using SNN similarity instead of Euclidean distance. And other concepts of density then follow the DBSCAN approach:

- **Core points**:
  A point is a core point if the number of points within a given neighborhood around the point, as determined by SNN similarity and a supplied parameter Eps exceeds a certain threshold $MinPts$, which also a supplied parameter.
- **Border points**:
  A border point is a point that is not a core point,i.e., there are not enough points in its neighborhood for it to be a core point, but it falls within the neighborhood of a core point.
- **Noise points**:
  A noise point is any point that is neither a core point nor a border point.

2.6.2 Algorithm

1. Compute SNN similarity graph.
2. Apply DBSCAN with user-specified parameters for Eps and $MinPts$

The algorithm automatically determines the number of clusters in the data. Note that not all points are clustered. The points that are discarded include noise and outliers, as well as points that are not strongly connected points to a group of points. SNN density-based clustering finds clusters in which the points are strongly related to one another.
Depending on the application, we might want to discard many of the points. For example, SNN density-based clustering is good for finding topics in groups of documents.

2.6.3 Complexity

- Time complexity: $O(m^2) + O(nlogn)$
  Equal to complexity of build SNN graph plus DBSCAN complexity.

2.6.4 Strengths and Limitations

Strengths

- The same as Javis - Patrick Clustering algorithm

Limitations

- Similar to javis-Patrick Clustering algorithm
- Because of using core points, SNN density-based adds considerable power and flexibility to this approach

3 Implementation

3.1 OPOSSUM clustering and CHAMELEON clustering

3.1.1 Environment and Library

To implement these 2 algorithms, we need an environment with:

- Environment: Python version 3.9
- Python Libraries: numpy, pandas, scikit-learn,networkx, METIS python wrapped version[5], and matplotlib.

3.1.2 Design Modules

With OPOSSUM, I have 2 module, while in CHAMELEON, we have 1 additional module to Merge subclusters.

1. create_metrics:
   This module is used to read the data, and create the knn metric (sparsified graph) for the process.
2. graph_partitioning:
   This module is used to: create a graph networkx, partitioning graph into subgraphs using METIS algorithms.
3. merge_partitions:
   This module is used to calculate the RC, RI, Self-Similarity between subgraphs, and merge subgraphs (subclusters) into single cluster.

3.1.3 Test with dataset

With these 2 algorithms, I have tested with 4 datasets: sample1.csv, sample2.csv, DE_target.csv, DE_edges.csv. All these 4 datasets will be discussed later.

3.2 Javis-Patrick clustering

3.2.1 Environment and Library

To implement Javis-Patrick algorithms, we need an environment with:

- Environment: Python version 3.9
- Python Libraries: numpy, pandas, scikit-learn, and matplotlib.

3.2.2 Design Modules

I have written this algorithm into 3 modules:

- snn_computation:
  This module is used to create SNN-similarity metrics from a KNN-metric.
- knn_metrics:
  This module is used to create knn-metrics from the dataset.
- create_graph:
  This module is used to run sparsified the SNN-similarity metric using a threshold parameter, and run the demonstration python notebook

3.2.3 Test with dataset

With this algorithm, I have tested with 2 datasets: DE_target, DE_edges.

3.3 SNN density-based clustering

3.3.1 Environment and Library

To implement SNN density-based algorithms, we need an environment with:

- Environment: Python version 3.9
- Python Libraries: numpy, pandas, scikit-learn, and matplotlib.

3.3.2 Design Modules

I have written this algorithm into 3 modules:

- snn_computation:
  This module is used to create SNN-similarity metrics from a KNN-metric.
- knn_metrics:
  This module is used to create knn-metrics from the dataset.
- execute:
  This module is used to run DBSCAN from the SNN-similarity metric, and run the demonstration python notebook.

3.3.3 Test with dataset

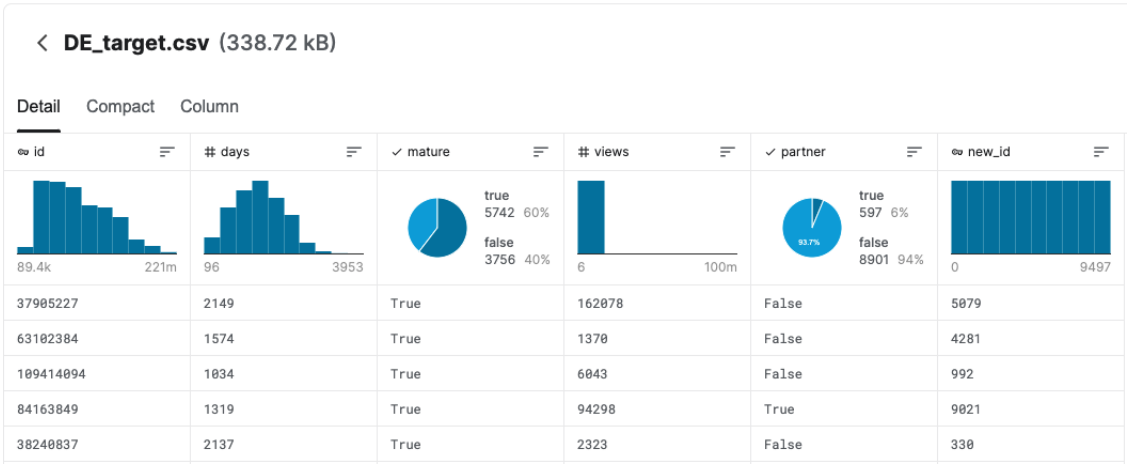
With this algorithm, I have tested with 2 datasets: DE_target, DE_edges.

4 Dataset

4.1 Twitch Dataset

[6] **Description**:
This dataset comes from Kaggle.
These datasets used for node classification and transfer learning are Twitch user-user networks of gamers who stream in a certain language. Nodes are the users themselves and the links are mutual friendships between them. Vertex features are extracted based on the games played and liked, location and streaming habits. Datasets share the same set of node features, this makes transfer learning across networks possible. These social networks were collected in May 2018. The supervised task related to these networks is binary node classification - one has to predict whether a streamer uses explicit language.

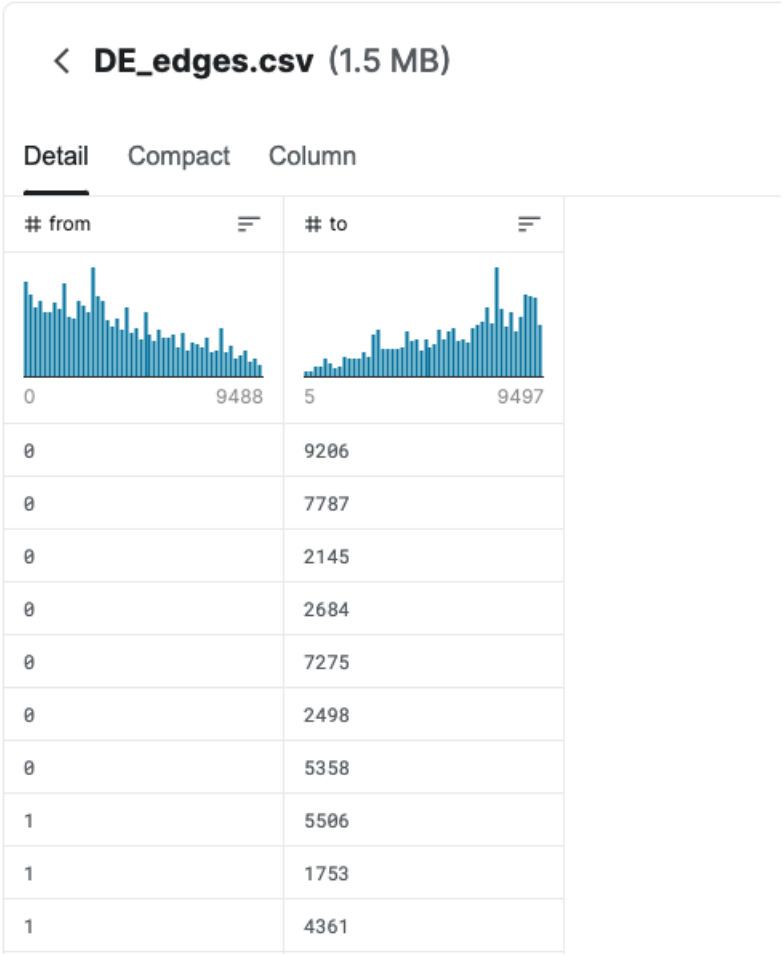
1. DE_target.csv:
   This is the twitch dataset of accounts in Germany. Each row of record is the statistic index of an account with attributes: id, days, mature, views, partner, new_id



id	days	mature	views	partner	new_id
37905227	2149	True	162078	False	5079
63102384	1574	True	1370	False	4281
109414094	1034	True	6043	False	992
84163849	1319	True	94298	True	9021
38240837	2137	True	2323	False	330

Figure 7: DE_target.csv

2. DE_edges.csv:
   This is the twitch dataset of accounts in Germany. Each row is a relationship between two 2 accounts. It means that an account has followed 1 account.



# from	# to
0	9206
0	7787
0	2145
0	2684
0	7275
0	2498
0	5358
1	5506
1	1753
1	4361

Figure 8: DE_edges.csv

We use this dataset to demonstrate the Javis-Patrick and SNN density-based Clustering.

4.2 Generated Dataset

This is the dataset that I have generated by coding. They are used to test the Chameleon algorithms. The idea about the shapes and density come from the paper of Chameleon Algorithm.

1. sample1.csv: have 5500 points

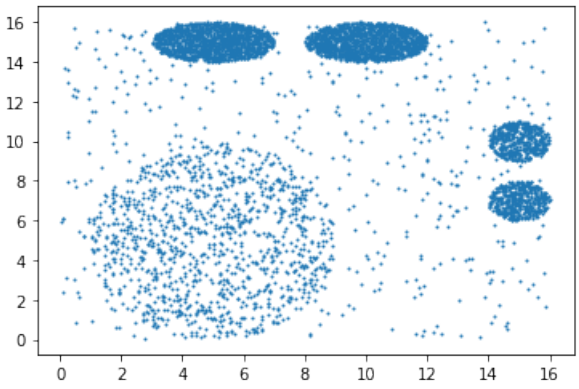


Figure 9: sample1.csv

2. sample2.csv: have 4000 points

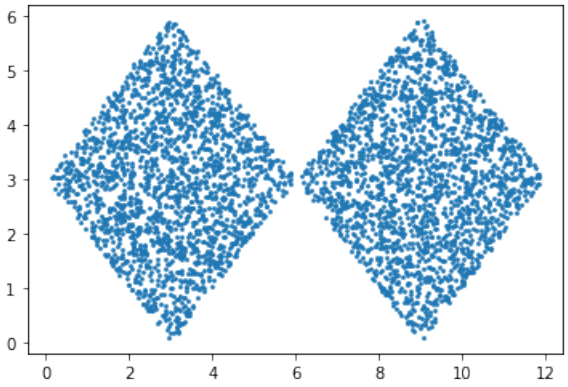


Figure 10: sample2.csv

5 Result and Evaluation

5.1 Evaluation methods

5.1.1 Visualize result

By using this step, we visualize the result before and after clustered the dataset. This is only useful when we have the dataset has low dimensional and can be visualized in 2D or 3D.

5.1.2 Silhoutte Coefficient

With the Unsupervised learning, the evaluation resulted clusters is based on 2 values:

- Cluster Cohesion: Determine how closely related are the objects in a cluster.

  $Cohesion(C_i) = \sum_{x \in C_i, y \in C_i} proximity(x, y)$

- Cluster Separation: Determine how distinct or well separated a cluster is from other cluster.

  $Separtion(C_i, C_j) = \sum_{x \in C_i, y \in C_j} proximity(x, y)$

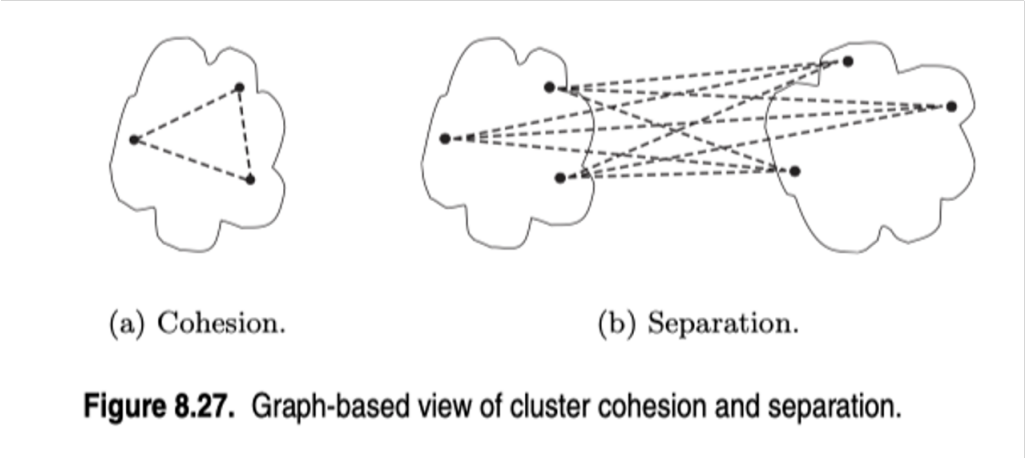


Figure 11: Cohesion and Separation

The Silhoutte Coefficient method combines both cohesion and separation an. The following steps explain how to compute Silhoutte coefficient:

(1) For the object x, calculate its average distance to all other objects in its cluster. We call this value $a(x)$, given by

$$a(x) = \frac{1}{|C_k| - 1} \sum_{y \in C_k, y \neq x} d(x, y). \tag{1}$$

(2) For the object x and any cluster not containing this object, calculate object's average distance to all the objects in the given cluster. Find the minimum value and call it b, given by

$$b(x) = \min_{j, j \neq k} \left[\frac{1}{|C_j|} \sum_{y \in C_j} d(x, y) \right]. \tag{2}$$

(3) For the object x, the Silhouette coefficient is

$$s(x) = \frac{b(x) - a(x)}{\max\{b(x), a(x)\}}. \tag{3}$$

(4) Repeat previous steps for all objects in cluster to get an average cluster's value.
(5) Afterwards sum cluster's Silhouettes and divide by number of clusters.

$$f_S(\mathbb{C}) = \frac{1}{|\mathbb{C}|} \sum_{k=1}^{|\mathbb{C}|} \left(\frac{1}{|C_k|} \sum_{x \in C_k} s(x) \right). \tag{4}$$

Figure 12: Silhoutte Coefficient steps

Let S is Silhoutte Coefficient score

- S is in range : [-1, 1]
- A clustering algorithms is good if : S not negative, S nearer to 1 – is better
- S $<$ 0: Wrong clusters
- S = 0 : Clusters overlap
- S = 1: Clusters are dense, well separated

5.2 Results

5.2.1 OPOSSUM clustering

- Sample1.csv

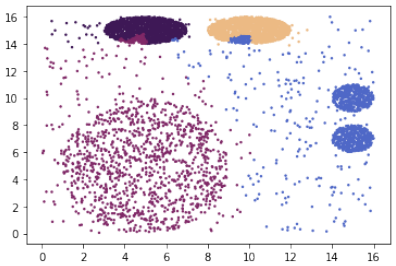


Figure 13: k=5,p=4

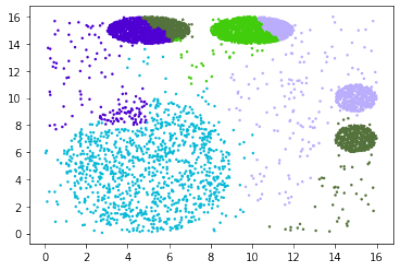


Figure 14: k=5,p=5

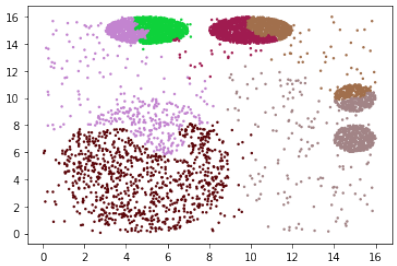


Figure 15: k=5,p=6

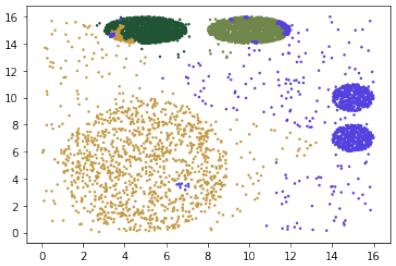


Figure 16: k=3,p=4

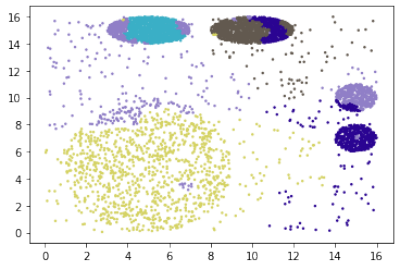


Figure 17: k=3,p=5

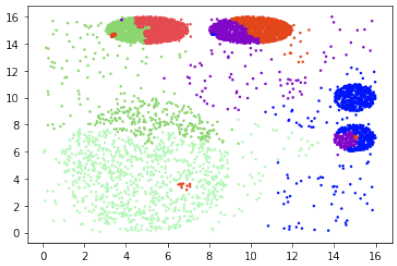


Figure 18: k=3,p=6

- Sample2.csv

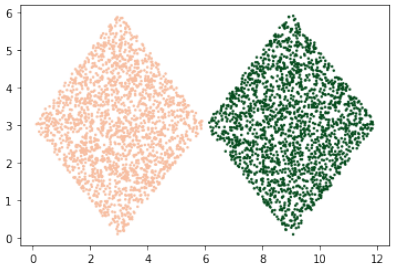


Figure 19: k=5,p=2

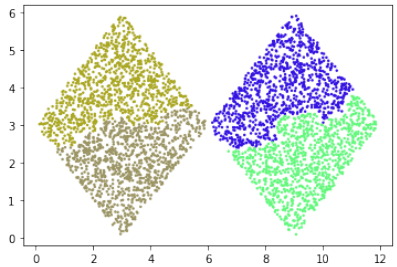


Figure 20: k=5,p=4

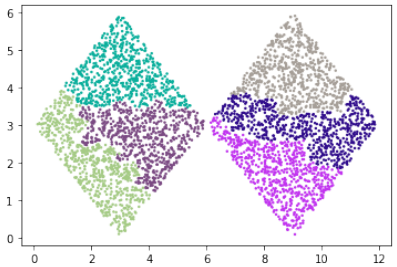


Figure 21: k=5,p=6

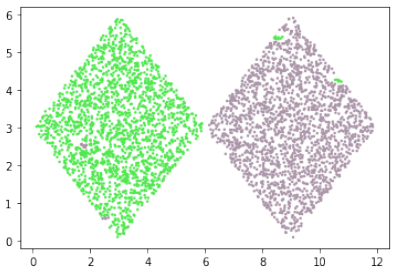


Figure 22: k=3,p=2

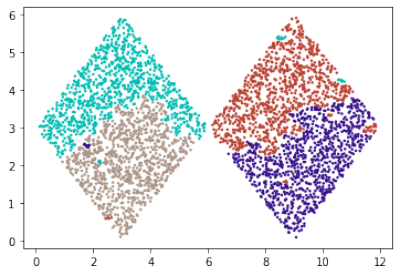


Figure 23: k=3,p=4

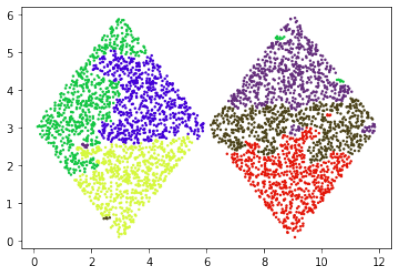


Figure 24: k=3,p=6

5.2.2 Chameleon clustering

- Sample1.csv

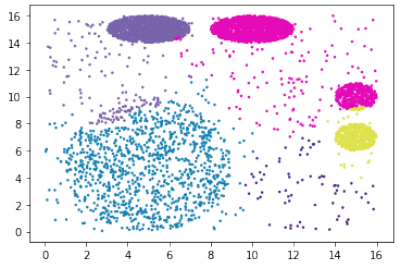


Figure 25: k=5,p=20, keep 5 clusters

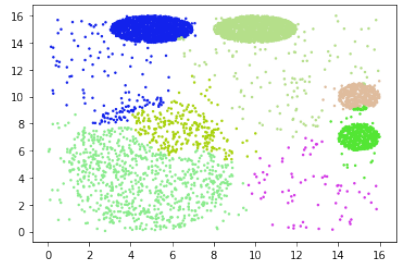


Figure 26: k=5,p=20, keep 7 clusters

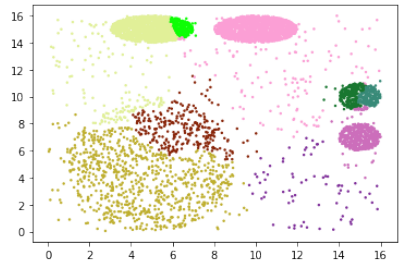


Figure 27: k=5,p=20,keep 9 clusters

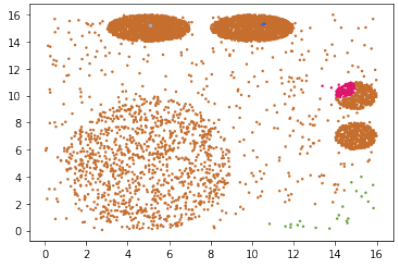


Figure 28: k=5,p=50, keep 5 clusters

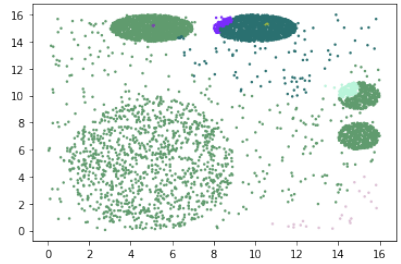


Figure 29: k=5,p=50, keep 7 clusters

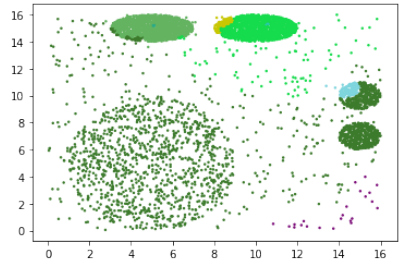


Figure 30: k=5,p=50,keep 9 clusters

- Sample2.csv

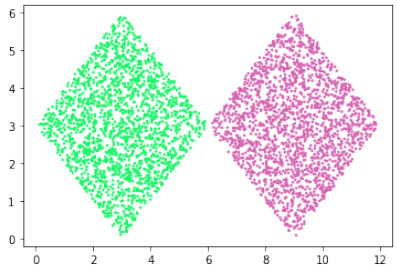


Figure 31: k=5,p=20, keep 2 clusters

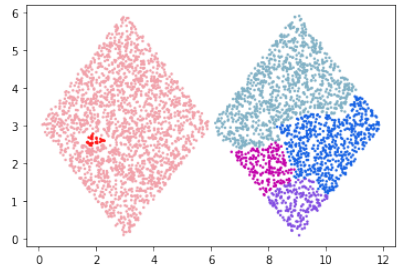


Figure 32: k=5,p=20, keep 6 clusters

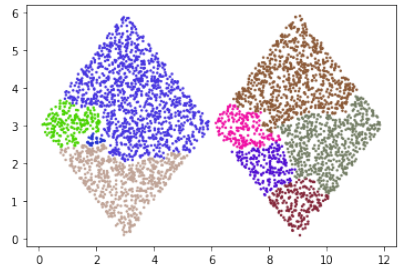


Figure 33: k=5,p=20,keep 9 clusters

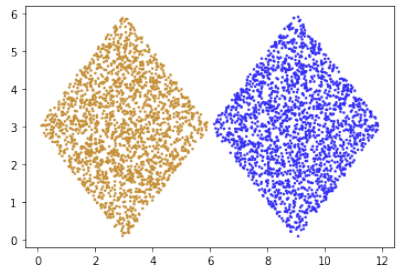


Figure 34: k=5,p=50, keep 2 clusters

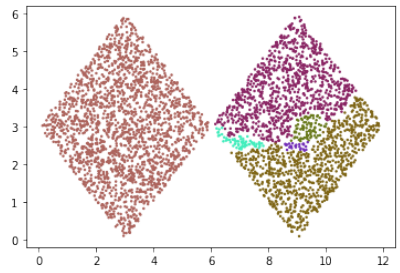


Figure 35: k=5,p=50, keep 6 clusters

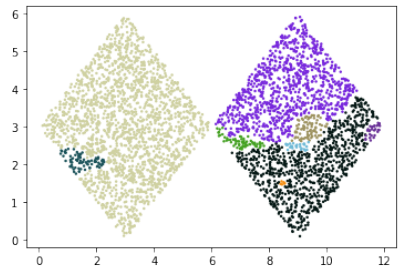


Figure 36: k=5,p=50,keep 9 clusters

5.2.3 Javis-Patrick clustering

- DE_edges.csv

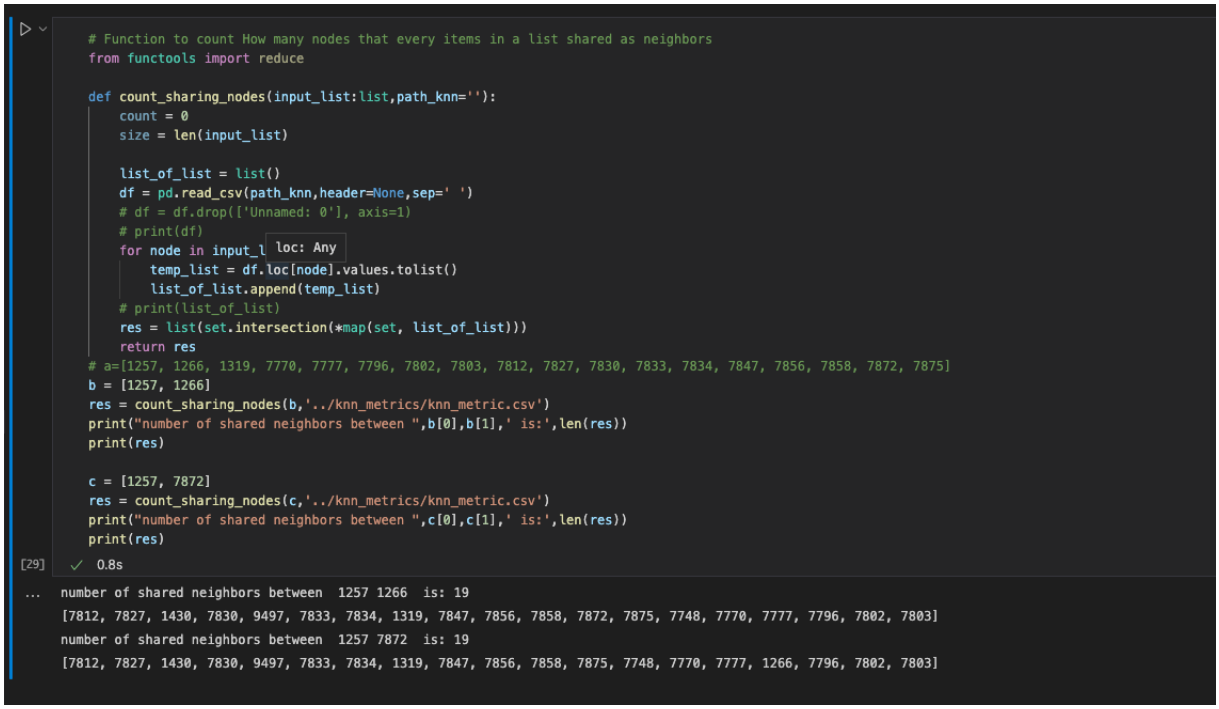
```
Threshold:  1 score: 0.022828595481381782 points: 4704 clusters: 509
Threshold:  2 score: 0.09639488660281718 points: 2014 clusters: 576
Threshold:  3 score: 0.20145674999047672 points: 838 clusters: 272
Threshold:  4 score: 0.34814728315897353 points: 426 clusters: 145
Threshold:  5 score: 0.46860608827862343 points: 293 clusters: 96
Threshold:  6 score: 0.5216897677942888 points: 248 clusters: 78
Threshold:  7 score: 0.5437769251459266 points: 227 clusters: 68
Threshold:  8 score: 0.5742705145674987 points: 210 clusters: 61
Threshold:  9 score: 0.5918509261529006 points: 196 clusters: 55
Threshold:  10 score: 0.6394588041227534 points: 181 clusters: 52
Threshold:  11 score: 0.6568614967365756 points: 167 clusters: 46
Threshold:  12 score: 0.6664600668535048 points: 157 clusters: 42
Threshold:  13 score: 0.691094855222317 points: 144 clusters: 38
Threshold:  14 score: 0.7189566755760081 points: 135 clusters: 36
Threshold:  15 score: 0.7550900773252771 points: 120 clusters: 32
Threshold:  16 score: 0.7200716907872788 points: 112 clusters: 30
Threshold:  17 score: 0.7691852630026038 points: 103 clusters: 27
Threshold:  18 score: 0.756106650555935 points: 93 clusters: 24
Threshold:  19 score: 0.7932184353004214 points: 72 clusters: 20
Number of labels needs to > 1. ERROR!!
Threshold:  20 score: -1 points: 0 clusters: 0
Number of labels needs to > 1. ERROR!!

Choose Threshold 19 where Silhoutte Score is highers, Clusters Output:
Cluster:  0
[1257, 1266, 1319, 7770, 7777, 7796, 7802, 7803, 7812, 7827, 7830, 7833
Cluster:  1
[1430, 7644]
```

```
Cluster:  2
[3341, 3407, 3454]
Cluster:  3
[3679, 3740]
Cluster:  4
[3811, 4826]
Cluster:  5
[3852, 4783]
Cluster:  6
[4407, 9497]
Cluster:  7
[5557, 5562]
Cluster:  8
[5792, 5816]
Cluster:  9
[6007, 6027, 6099, 6129, 8261, 8295]
Cluster:  10
[6110, 6211, 6292]
Cluster:  11
[6497, 6504, 6529]
Cluster:  12
[6582, 6594]
Cluster:  13
[6698, 6703, 6712, 6750, 6774, 6776, 8990]
Cluster:  14
[6823, 8059]
Cluster:  15
[7748, 7757]
Cluster:  16
[8340, 8357]
Cluster:  17
[8400, 8432, 8435, 8462, 8558]
Cluster:  18
[8864, 8878, 8905]
Cluster:  19
[9201, 9218]
```

To check for confirmation, we check the number of nearest neighbors between 2 ran nodes of a clusters.

- DE_target.csv

```
Threshold:  1 score: 0.00025288726176571173 points: 9431 clusters:
```



```
# Function to count How many nodes that every items in a list shared as neighbors
from functools import reduce

def count_sharing_nodes(input_list:list,path_knn=''):
    count = 0
    size = len(input_list)

    list_of_list = list()
    df = pd.read_csv(path_knn,header=None,sep=' ')
    # df = df.drop(['Unnamed: 0'], axis=1)
    # print(df)
    for node in input_l  loc: Any
        temp_list = df.loc[node].values.tolist()
        list_of_list.append(temp_list)
    # print(list_of_list)
    res = list(set.intersection(*map(set, list_of_list)))
    return res
# a=[1257, 1266, 1319, 7770, 7777, 7796, 7802, 7803, 7812, 7827, 7830, 7833, 7834, 7847, 7856, 7858, 7872, 7875]
b = [1257, 1266]
res = count_sharing_nodes(b,'../knn_metrics/knn_metric.csv')
print("number of shared neighbors between ",b[0],b[1],' is:',len(res))
print(res)

c = [1257, 7872]
res = count_sharing_nodes(c,'../knn_metrics/knn_metric.csv')
print("number of shared neighbors between ",c[0],c[1],' is:',len(res))
print(res)
```

```
[29]  ✓ 0.8s

...  number of shared neighbors between  1257 1266  is: 19
     [7812, 7827, 1430, 7830, 9497, 7833, 7834, 1319, 7847, 7856, 7858, 7872, 7875, 7748, 7770, 7777, 7796, 7802, 7803]
     number of shared neighbors between  1257 7872  is: 19
     [7812, 7827, 1430, 7830, 9497, 7833, 7834, 1319, 7847, 7856, 7858, 7875, 7748, 7770, 7777, 1266, 7796, 7802, 7803]
```

H

Figure 37: Check result of JP Clustering

```
Threshold:  2 score: 0.0007017305306636791 points: 9307 clusters: 52
Threshold:  3 score: 0.0015203117541109567 points: 8930 clusters: 87
Threshold:  4 score: 0.021437848776648605 points: 7441 clusters: 809
Threshold:  5 score: 0.06385603716084443 points: 5286 clusters: 1327
Threshold:  6 score: 0.11330858195586262 points: 3412 clusters: 1187
Threshold:  7 score: 0.16965004003471004 points: 2128 clusters: 855
Threshold:  8 score: 0.22526728663672282 points: 1437 clusters: 613
Threshold:  9 score: 0.28882393393749456 points: 1026 clusters: 452
Threshold:  10 score: 0.3479264453175878 points: 776 clusters: 349
Threshold:  11 score: 0.3981843388597246 points: 634 clusters: 289
Threshold:  12 score: 0.42673199355937963 points: 565 clusters: 259
Threshold:  13 score: 0.4488162992865269 points: 518 clusters: 239
Threshold:  14 score: 0.4673107326708206 points: 480 clusters: 222
Threshold:  15 score: 0.484636583005609 points: 453 clusters: 211
Threshold:  16 score: 0.5023286378636621 points: 422 clusters: 197
Threshold:  17 score: 0.5247756335524537 points: 387 clusters: 181
Threshold:  18 score: 0.5508002994924766 points: 345 clusters: 162
Threshold:  19 score: 0.566533607263124 points: 325 clusters: 153
Threshold:  20 score: 0.5837094291874803 points: 299 clusters: 140
Threshold:  21 score: 0.6037291640040736 points: 275 clusters: 129
Threshold:  22 score: 0.6240554481701632 points: 259 clusters: 123
```

```
Threshold:  23 score: 0.64724003338873 points: 238 clusters: 114
Threshold:  24 score: 0.6576380507761541 points: 226 clusters: 108
Threshold:  25 score: 0.6840517569036277 points: 199 clusters: 95
Threshold:  26 score: 0.6977437379437641 points: 188 clusters: 90
Threshold:  27 score: 0.7067049704459225 points: 180 clusters: 86
Threshold:  28 score: 0.7113775496893628 points: 177 clusters: 85
Threshold:  29 score: 0.7244225273813588 points: 164 clusters: 79
Threshold:  30 score: 0.7401085533006359 points: 151 clusters: 73
Threshold:  31 score: 0.7459061486096278 points: 145 clusters: 70
Threshold:  32 score: 0.7535794755258064 points: 137 clusters: 66
Threshold:  33 score: 0.7779518580300804 points: 115 clusters: 55
Threshold:  34 score: 0.7858587606678111 points: 110 clusters: 53
Threshold:  35 score: 0.8019881277369562 points: 103 clusters: 51
Threshold:  36 score: 0.8155266372423285 points: 92 clusters: 46
Threshold:  37 score: 0.830012919985965 points: 80 clusters: 40
Threshold:  38 score: 0.8506268383642117 points: 68 clusters: 34
Threshold:  39 score: 0.870303555148127 points: 56 clusters: 28
Threshold:  40 score: 0.881139981765902 points: 50 clusters: 25
Threshold:  41 score: 0.9016902833298268 points: 40 clusters: 20
Threshold:  42 score: 0.9107669814775855 points: 36 clusters: 18
Threshold:  43 score: 0.9249203777731024 points: 30 clusters: 15
Threshold:  44 score: 0.9349081281997336 points: 26 clusters: 13
Threshold:  45 score: 0.9520000000000003 points: 20 clusters: 10
Threshold:  46 score: 0.9520000000000003 points: 20 clusters: 10
Threshold:  47 score: 0.9657142857142856 points: 14 clusters: 7
Threshold:  48 score: 0.976 points: 10 clusters: 5
Threshold:  49 score: 0.98 points: 8 clusters: 4
```

Choose the threshold 45, that has high Silhoutte Coefficient score. The

```
Cluster:  0
      new_id  days  views
541     1192  2066  22812
4556    1189  2066  23459
Cluster:  1
      new_id  days  views
5236    1227  1546  21032
8922    1563  1546  22543
Cluster:  2
      new_id  days  views
1029    1670   744   2734
2711    1715   744   2630
```

```
Cluster:  3
      new_id  days  views
2825    7908  1117    235
9013    2893  1117    104
Cluster:  4
      new_id  days  views
68      8616  1359   1021
6144    3113  1359   1184
Cluster:  5
      new_id  days  views
4526    8500   959   1521
5148    3876   959   1535
Cluster:  6
      new_id  days  views
787     4062  1161   4973
3603    4190  1161   4836
Cluster:  7
      new_id  days  views
5080    6218  1515   4046
5347    4860  1515   4013
Cluster:  8
      new_id  days  views
1129    7636  1229   6408
4616    5354  1229   6312
Cluster:  9
      new_id  days  views
4046    6394   723  12255
7008    6369   723  15023
```

5.2.4 SNN Density-based clustering

- DE_edges.csv
  - epsilon = 0.65, and MinPts in range (3,20)

```
Silhoutte   Number of Clusters  Number of nodes
0.5196751942481924 19 129
0.5100750005371928 12 108
0.517969882717201 7 88
0.508209032278478 6 83
0.5390059075669844 6 78
0.506701325683917 5 71
0.506701325683917 5 71
```

```
0.5003226645849598 4 61
0.5040896358543417 3 51
0.5040896358543417 3 51
0.5277210884353741 2 35
Number of labels needs to > 1. ERROR!!
-1 1 22
Number of labels needs to > 1. ERROR!!
-1 1 20
Number of labels needs to > 1. ERROR!!
-1 1 20
Number of labels needs to > 1. ERROR!!
-1 1 20
Number of labels needs to > 1. ERROR!!
-1 1 20
Number of labels needs to > 1. ERROR!!
-1 1 20
```

– MinPts = 2, epsilon in range (0.05,1)

```
epsilon:    Silhoutte   clusters    points
0.05 : 0.7932184353004214 20 72
0.1 : 0.756106650555935 24 93
0.15000000000000002 : 0.7691852630026038 27 103
0.2 : 0.7200716907872788 30 112
0.25 : 0.7550900773252771 32 120
0.3 : 0.7189566755760081 36 135
0.35000000000000003 : 0.691094855222317 38 144
0.4 : 0.6664600668535048 42 157
0.45 : 0.6568614967365756 46 167
0.5 : 0.6394588041227534 52 181
0.55 : 0.5918509261529006 55 196
0.6000000000000001 : 0.5742705145674987 61 210
0.6500000000000001 : 0.5437769251459266 68 227
0.7000000000000001 : 0.5216897677942888 78 248
0.7500000000000001 : 0.46860608827862343 96 293
0.8 : 0.34814728315897353 145 426
0.8500000000000001 : 0.20145674999047672 272 838
0.9000000000000001 : 0.09639488660281718 576 2014
0.9500000000000001 : 0.022828595481381782 509 4704
```

```
Choose epsilon=0.05, which has highest Silhoutte Coefficient,
and also the same result in Javis Patrick Clustering.
The cluster output is
Cluster:  0
```

```
[1257, 1266, 1319, 7770, 7777, 7796, 7802, 7803, 7812, 7827, 7830,
7833, 7834, 7847, 7856, 7858, 7872, 7875]
Cluster:  1
[1430, 7644]
Cluster:  2
[3341, 3407, 3454]
Cluster:  3
[3679, 3740]
Cluster:  4
[3811, 4826]
Cluster:  5
[3852, 4783]
Cluster:  6
[4407, 9497]
Cluster:  7
[5557, 5562]
Cluster:  8
[5792, 5816]
Cluster:  9
[6007, 6027, 6099, 6129, 8261, 8295]
Cluster:  10
[6110, 6211, 6292]
Cluster:  11
[6497, 6504, 6529]
Cluster:  12
[6582, 6594]
Cluster:  13
[6698, 6703, 6712, 6750, 6774, 6776, 8990]
Cluster:  14
[6823, 8059]
Cluster:  15
[7748, 7757]
Cluster:  16
[8340, 8357]
Cluster:  17
[8400, 8432, 8435, 8462, 8558]
Cluster:  18
[8864, 8878, 8905]
Cluster:  19
[9201, 9218]
```

- DE_target.csv

```
The result when MinPts=2, Epsilon in range(0.05,1)
epsilon:    Silhoutte   clusters    points
0.05 : 0.976 5 10
0.1 : 0.9520000000000003 10 20
0.15000000000000002 : 0.9249203777731023 15 30
0.2 : 0.881139981765902 25 50
0.25 : 0.8506268383642116 34 68
0.3 : 0.8155266372423285 46 92
0.35000000000000003 : 0.7779518580300804 55 115
0.4 : 0.7401085533006359 73 151
0.45 : 0.7113775496893628 85 177
0.5 : 0.6840517569036277 95 199
0.55 : 0.64724003338873 114 238
0.6000000000000001 : 0.5837094291874803 140 299
0.6500000000000001 : 0.5508002994924766 162 345
0.7000000000000001 : 0.484636583005609 211 453
0.7500000000000001 : 0.44881629928652683 239 518
0.8 : 0.34792644531758776 349 776
0.8500000000000001 : 0.22526728663672282 613 1437
0.9000000000000001 : 0.06385603716084443 1327 5286
0.9500000000000001 : 0.0015203117541109567 87 8930

Choose epsilon=0.1, which has the high Silhoutte coefficient score, and

Cluster:  0
      new_id  days  views
541     1192  2066  22812
4556    1189  2066  23459
Cluster:  1
      new_id  days  views
5236    1227  1546  21032
8922    1563  1546  22543
Cluster:  2
      new_id  days  views
1029    1670   744   2734
2711    1715   744   2630
Cluster:  3
      new_id  days  views
2825    7908  1117    235
9013    2893  1117    104
Cluster:  4
      new_id  days  views
```

```
68      8616  1359   1021
6144    3113  1359   1184
Cluster:  5
      new_id  days  views
4526    8500   959   1521
5148    3876   959   1535
Cluster:  6
      new_id  days  views
787     4062  1161   4973
3603    4190  1161   4836
Cluster:  7
      new_id  days  views
5080    6218  1515   4046
5347    4860  1515   4013
Cluster:  8
      new_id  days  views
1129    7636  1229   6408
4616    5354  1229   6312
Cluster:  9
      new_id  days  views
4046    6394   723  12255
7008    6369   723  15023
```

5.3 Overall Conclusion

- Both 4 algorithms performances are the same as their Strengths and Limitations mentioned.
- Chameleon requires a lot of resources (time, sapce), it works not too good if dataset contain noises density similar to density of clusters.
- Javis-Patrick Clustering, SNN Densoty-based Clustering can detected clusters whose nodes are strongly connected. However, they reduces a lot of noise and outliers points. In addition, the resulted clusters are small.

6 Problems and Future Improvements

6.1 Problems

- The written source code was not optimized.
- The resource required for running these algorithms is a big problem. The computational cost is very high.
- I did not test more types of dataset.
- Although we implemented a method to evaluate result, the specified parameters in each algorithms is a problem.

6.2 Future Improvements

- Research some methods to optimize the calculation of KNN process, SNN process.
- Test more types of Social network data.
- Refactor code for optimization.

References

[1] M. S. Pang-Ning Tan, Vipin Kumar. Introduction to data mining. [June 2020]. [Online]. Available: https://www.pearson.com/us/higher-education/product/Tan-Introduction-to-Data-Mining/9780321321367.html

[2] K. Lab. Metis - serial graph partitioning and fill-reducing matrix ordering. [April 2020]. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[3] H. L. Benedikt Hufnagl. Ma graph-based clustering method with special focus on hyperspectral imaging. [30 May 2019]. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000326701931311X

[4] V. K. George Karypis, Eui-Hong (Sam) Han. Chameleon: A hierarchical clustering algorithm using dynamic modeling. [· September 1999]. [Online]. Available: https://www-users.cse.umn.edu/~hanxx023/dmclass/chameleon.pdf

[5] Python. Metis for python. [January 2022]. [Online]. Available: https://metis.readthedocs.io/en/latest/

[6] A. Garritano. Twitch social networks. [May 2018]. [Online]. Available: https://www.kaggle.com/andreagarritano/twitch-social-networks