Q-1

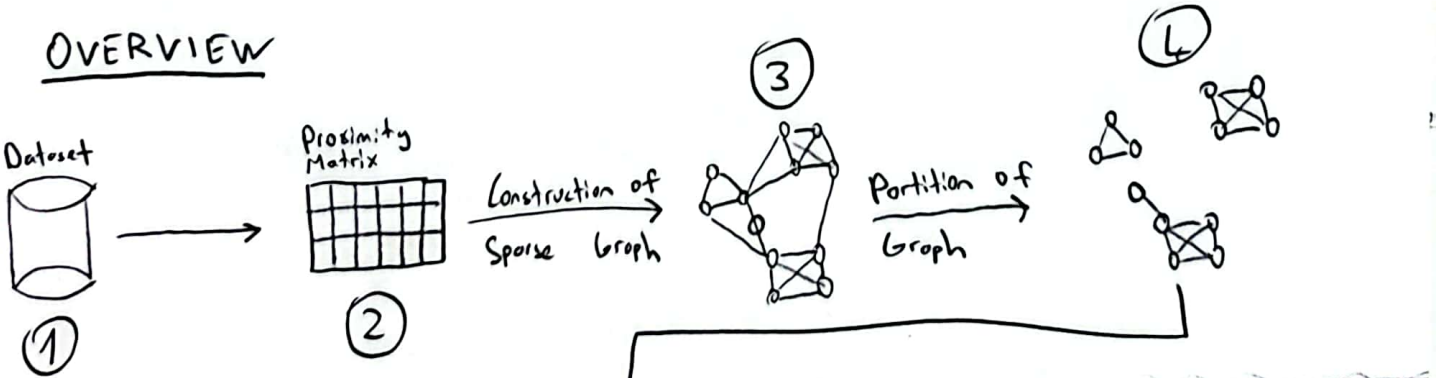
INTRODUCTION

" Chameleon is a hierarchical clustering algorithm that uses dynamic modeling to determine the similarity between pairs of clusters. Cluster similarity is assessed based on how well connected objects are within a cluster and the proximity of clusters. That is, two clusters are merged if their interconnectivity is high and they are close together. Thus, Chameleon does not depend on a static, user-supplied model and can automatically adapt to the internal characteristics of the clusters being merged. The merge process facilitates the discovery of natural and homogenious clusters and applies to all data types as long as a similarity function can be specified. " [1]

The dynamic model that is mentioned above is used to measure the similarity between clusters.

* Main properties are the relative closeness and relative interconnectivity of the cluster.
* Two clusters are combined if the resulting cluster shares these properties with the constituent clusters.
* The merging preserves self-similarity.

OVERVIEW



→ Sparse Graph using KNN

- p and q are connected if q is among the top k closest neighbors of p.
- Edge weights = similarity of two points.

→ If the graph was fully-connected instead of sparse, we cannot handle the computional cost of it.

The idea is, cut the weak connections to create "tight" clusters in step-3 in overview. Then, we are going to merge those tight clusters again to create clusters.

Relative Interconnectivity

Two clusters are combined if the points in the resulting cluster are almost as strongly connected as points in each of the original clusters.

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

$EC(C_i, C_j)$ = Sum of edges that connect clusters C_i and C_j

$EC(C_i)$ = Minimum sum of the cut edges that partition C_i to two roughly equal parts

Relative Closeness

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.

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

m_i and m_j size of the clusters

$\bar{S}_{EC(C_i, C_j)}$ = Average weight of edges that connects clusters.

$\bar{S}_{EC(C_i)}$ = Average weight of edges if we bisect cluster C_i

* Greater RI and greater RC is better.

Q-2 — Note: I strongly suggest you to read the answer of question-3. Because I showed time complexity of Chameleon algorithm there.

Advantages and Disadvantages

+ Chameleon is relatively robust to outliers and noises.
+ Efficient for low dimensional space.
+ It is flexible. It can discover clusters of various shapes and sizes.
- Time complexity of Chameleon can be high especially for large and high dimensional space.
- Initial partitioning is very important. The quality of the initial partitioning can significantly affect the final clustering result.
- The performance of the Chameleon is sensitive to the choice of parameters such as thresholds.

* Chameleon vs DBSCAN

DBSCAN is much more efficient than Chameleon. It has $O(n^2)$ in the worst case. However, it requires significant memory to store density-based data. (I explained time complexity of Chameleon in next question.)

DBSCAN is more flexible. It identifies arbitrary shape of clusters.
DBSCAN is more robust.
DBSCAN can automatically determine the number of clusters.

* Chameleon vs K-Means

K-Means has time complexity of $O(tnk)$ where t is the num of iterations, n is number of data points, k is the number of clusters. It is generally faster than Chameleon, (I explained the time complexity of Chameleon in next question) especially for large datasets.

In terms of memory, K-Means is also more efficient than Chameleon as it only needs to store the data points and cluster centers.

However, Chameleon is more scalable, more robust and more flexible. than K-means.

* Chameleon vs BIRCH

Both algorithms are hierarchical algorithms. BIRCH is designed to be memory efficient. It is more efficient in terms of memory

BIRCH is faster then Chameleon. Its complexity is $O(n)$.

Chameleon can be more flexible in terms of the shapes of the clusters it can discover.

Note => I tried to compare Chameleon with different type of algorithms.

Q-3 Time Complexity

We can calculate the time complexity by primaly considering two phases :

- The Graph Partitioning Phase (Step-2 in my pseudo code)

  The construction of k-nearest neighbor graph is $O(n^2)$ since in the worst case, each data point needs to be compared with every other data point to determine the nearest neighbors. The graph partitioning step can be done in $O(n \log n)$ using an efficient graph partitioning algorithms such as METIS

- Merge Phase (Step-4 in my pseudo code)

  The time complexity is calculated by two factors.

  - The time to compute the internal interconnectivity and internal closeness for each initial and intermediate cluster. This is done by bisecting the corresponding k-nearest neighbor sub-graph of the cluster. The worst-case complexity for this part is $O(nm)$ where n is the total data points, where m is the number of initial sub-clusters produced by the graph partitioning phase of the algorithm.

  - The time to select the most similar pair of clusters to merge This can be done using a heap-based priority-queue, and in the worst case, it takes $O(m^2 \log m)$

Therefore, the overall time complexity is $O(nm + n\log n + m^2 \log m)$

Note ⇒ This is optimized version of Chameleon.

PSEUDO - CODE

Input: Dataset D, Number of Clusters k, Number of Neorest Neighbors $k1$, Threshold t

Output: Set of Clusters

1. Initilioze C to on empty set
2. • Construct a k-nearest neighbor graph G from the dataset
   • Partition G into sub-graphs using a graph partitioning algorithm like METIS. The number of sub-graphs should be larger than the number of desired clusters.
3. • For each sub-graph in G:
      o Compute the Internal Connectivity and Closeness of sub-graph
      o Add the sub-graph to the set of clusters C
4. • While the number of clusters in C is greater than k:
      o For each pair of clusters (C_i, C_j) in C
         ⊗ Compute the interconnectivity and closeness between C_i and C_j
         ⊗ Compute the relative interconnectivity and relative closeness of C_i and C_j
      o Find the pair of clusters (C_i, C_j) that has the maximum relative interconnectivity and relative closeness.
      o If the maximum relative interconnectivity and relative closeness is greater than the threshold t, merge C_i and C_j.
5. Return set of clusters C.