A Neighborhood-Based Clustering by Means of the Triangle Inequality

Documentation Structure:

1. **Introduction**
2. **Algorithm Description**
    - Detailed explanation of the proposed algorithm with pseudocode and step-by-step steps for implementation.
3. **Implementation Details**
    - The core functions of the algorithm.
    - Libraries and tools used.
    - Explanation of key parameters (e.g., threshold, distance function).
4. **Experimental Results**

    Present findings using:
    - **Tables**: Displays accuracy and efficiency metrics for different datasets.
    - **Graphs**: Visualizes clusters and compare them with baseline methods.
5. **Conclusion and Next Steps**

1. Introduction

Grouping data into meaningful clusters is a crucial task in both artificial intelligence and data mining. Among the various clustering approaches, density-based algorithms are particularly significant as they rely on calculating the neighborhood of a given data point. However, these algorithms often face computational bottlenecks when dealing with high-dimensional data. To address this challenge, we propose a novel TI-k-Neighborhood-Index algorithm that leverages the triangle inequality to efficiently compute k-neighborhoods for all points in a dataset. Experimental results demonstrate that the Neighborhood-Based Clustering (NBC) algorithm, when supported by our index, outperforms NBC implementations that use established spatial indices like VA-file and R-tree, both in low and high-dimensional scenarios.

2. Algorithm Description

Proposed solution in pseudocode

Input:

- *Data points (D): A set of data points*
- *Distance function (dist): A distance function*
- *Threshold value for clustering (T): A threshold value for clustering*

Output:

- *Clusters (C): A set of clusters*

Algorithm:

1. Initialize an empty set of clusters C.

2. For each data point p in D:

a. Determine the neighborhood of p:

- $N(p) = \{q \in D \mid dist(p, q) \leq T\}$.

b. For each neighbor q in N(p):

- *Check if q satisfies triangle inequality with respect to other neighbors.*
- *If satisfied, assign q to the same cluster as p.*

3. Repeat 2a and b until all data points are clustered.

4. Return clusters C.

3. Implementation Details

Programming Language

- **Python**: An optimal choice due to its rich ecosystem of libraries for machine learning, clustering, and data visualization.

Libraries and Tools

1. **numpy**: For efficient numerical computations and distance calculations.
2. **scikit-learn**: Provides utility functions for clustering evaluation metrics and synthetic dataset generation.
3. **pandas**: Facilitates data manipulation and preprocessing.
4. **Matplotlib** and **seaborn**: For visualizing clustering results.

4. Experimental Results

Objective

Evaluates the performance of the neighborhood-based clustering algorithm using synthetic and real-world datasets.

Datasets

1. **Real-World Datasets**:
   - Iris dataset (available in scikit-learn).
   - Additional datasets from the UCI Machine Learning Repository as needed.

Experimental Steps

1. **Data Preparation**:
   - To load or generate datasets.
   - Preprocessing data (normalize, handle missing values).
2. **Algorithm Implementation**:
   - Developing the neighborhood-based clustering algorithm.
   - Using Python and libraries as described above.
3. **Performance Evaluation**:
   - To compare results with other clustering methods (e.g., K-Means, DBSCAN).
   - Using the following metrics:
     - **Adjusted Rand Index (ARI)**: Measures clustering similarity.
     - **Silhouette Score**: Evaluates the quality of clustering.
     - **Execution Time**: Measures computational efficiency as dataset size increases.
4. **Visualization**:
   - To plot clustering results using 2D or 3D scatter plots.
   - To highlight clusters and outliers.

5. **Analysis**:
    - To compare clustering accuracy and execution time across different datasets and thresholds.
    - To identify strengths and weaknesses of the algorithm.

5. <u>Conclusion</u>

- Algorithm performance on the chosen datasets.
- Benefits and limitations.
- Potential improvements (e.g., adaptive thresholds, parallelization for large datasets).

<u>Next Steps</u>

- Finalize the pseudocode into Python code.
- Design experiments with varied datasets.
- Document findings with comprehensive visuals and interpretations