University of Dhaka

Department of Computer Science and Engineering

CSE-4255 : Introduction to Data Mining and Warehousing Lab

Lab Report: Comparative Analysis of Clustering Algorithms (k-Means and DBSCAN)

Submitted By:

Name : Zisan Mahmud

Roll No : 23

Submitted On:

JULY 17, 2025

Submitted To:

Dr. Chowdhury Farhan Ahmed

Md. Mahmudur Rahman

Contents

1 ABSTRACT

This report presents a comparative analysis of k-Means and DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm. Total ten datasets are used and both algorithms are implemented on the datasets to determine the most efficient method. The differet accuracy measures (silhouette coefficient, within-cluster variation) of the two algorithms is also analyzed.

keywords: k-Means, DBSCAN.

2 INTRODUCTION

Clustering is a core technique in unsupervised learning that focuses on discovering the natural grouping or structure within a dataset without using predefined labels. The primary goal is to partition a set of data points into clusters, such that points within the same group exhibit high similarity, while those in different clusters are significantly dissimilar. This similarity is typically measured based on distance metrics in a multidimensional space, such as Euclidean or Manhattan distance.

In contrast to classification, which relies on prior knowledge or labeled data, clustering allows patterns to emerge directly from the data itself. This makes it a valuable approach in situations where labeling is costly, unavailable, or impractical. Applications of clustering can be found across numerous domains, including market segmentation, social network analysis, image processing, biological data analysis, and anomaly detection.

A good clustering algorithm should ideally generate compact and well-separated groups, be robust to noise, and scale well with increasing data volume and dimensionality. However, different algorithms approach this task using different definitions of a "cluster"—some based on density, others on centroid distance or connectivity. As such, the choice of algorithm often depends on the shape and distribution of the data.

In this context, two widely used clustering algorithms—k-Means and DBSCAN—present contrasting approaches. While k-Means relies on partitioning data around centroids with fixed cluster counts, DBSCAN forms clus-

ters based on data density, offering greater flexibility in detecting irregular shapes and handling noise. Understanding the fundamental differences between these two approaches is essential for selecting the right method in practical applications.

k-Means (Centroid-Based Technique)

Given a dataset D comprising n objects situated in a Euclidean space, partitioning methods aim to divide the dataset into k disjoint clusters $C_1, C_2, \ldots, C_k$, such that $C_i \subset D$ and $C_i \cap C_j = \emptyset$ for all $i \neq j$. The quality of such a partitioning is evaluated using an objective function that seeks to ensure that objects within the same cluster are highly similar, whereas those in different clusters are significantly dissimilar. This implies achieving high intra-cluster similarity and low inter-cluster similarity.

In centroid-based methods, each cluster C_i is represented by a centroid c_i, which ideally lies at the center of the cluster. The centroid can be defined either as the mean or medoid of the objects within the cluster. The dissimilarity between an object $p \in C_i$ and the cluster centroid c_i is commonly measured using the Euclidean distance, denoted $\text{dist}(p, c_i)$.

A common measure to evaluate the compactness of clusters is the **sum of squared error (SSE)**, given by:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} \text{dist}(p, c_i)^2 \tag{1}$$

Here, E reflects the total within-cluster variation across all clusters. The objective is to minimize this value to obtain tightly packed clusters that are well-separated.

Finding the optimal clustering that minimizes E is computationally intensive. The problem is NP-hard, even for $k = 2$ clusters in general Euclidean space. When both k (number of clusters) and d (data dimensionality) are fixed, the problem can be solved in $O(nd^{k+1} \log n)$ time. Due to the high computational cost of finding exact solutions, heuristic or greedy approaches are typically adopted in practice.

One of the most widely used heuristic techniques for clustering is the k-means algorithm. It represents each cluster by its mean, and proceeds as follows:

1. Randomly select k objects from D as the initial cluster centers.
2. Repeat until convergence:
   (a) Assign each object in D to the cluster whose mean is nearest to the object (based on Euclidean distance).
   (b) Recalculate the mean of each cluster using the current members of the cluster.

This iterative process, called *iterative relocation*, continues until the cluster assignments no longer change between iterations. The algorithm aims to gradually reduce the value of E by updating the cluster centers and reassigning the points accordingly.

The k-means algorithm does not guarantee convergence to the global optimum. It may instead settle at a local optimum, and the final outcome can be influenced by the initial selection of cluster centers. To address this issue, it is standard practice to execute the algorithm multiple times with different initializations and select the best result.

The time complexity of the k-means algorithm is $O(nkt)$, where n is the number of data points, k is the number of clusters, and t is the number of iterations. Since k and t are generally much smaller than n, the algorithm is considered efficient and scalable for large datasets.

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that identifies clusters as areas of high point density in a data space. Unlike partition-based methods, DBSCAN does not require the number of clusters to be specified beforehand and can discover clusters of arbitrary shape while effectively identifying noise points.

The algorithm depends on two user-defined parameters: ε (epsilon), which defines the radius of a neighborhood around a point, and *MinPts*, the minimum number of points required in the ε-neighborhood for a point to be considered a core point. Based on these parameters, points are categorized as core points, border points, or noise. A core point has at least *MinPts* neighbors within ε. A border point is within the ε-neighborhood of a core point but has fewer than *MinPts* neighbors itself. Points that are neither core nor border are labeled as noise.

DBSCAN builds clusters by identifying core points and expanding them through density reachability. A point p is directly density-reachable from a point q if p is within the ε-neighborhood of q and q is a core point. A point is density-reachable from another point if there exists a sequence of directly density-reachable steps connecting them. Two points are density-connected if they are both density-reachable from a common core point. Clusters are defined as maximal sets of density-connected points.

The algorithm starts with all points marked as unvisited. It selects a random unvisited point and checks its neighborhood. If the point is a core point, a new cluster is formed and expanded by recursively collecting all density-reachable points. If it is not a core point, it is temporarily labeled as noise. The process continues until all points are visited and assigned to a cluster or identified as noise.

DBSCAN is efficient for large datasets, especially when using spatial indexing structures like R-trees, with a time complexity of $O(n \log n)$. Without such indexing, the complexity is $O(n^2)$. Its ability to detect clusters of arbitrary shape and to distinguish outliers makes DBSCAN a powerful tool for density-based clustering.

3 IMPLEMENTATION DETAIL

This section outlines the implementation specifics for both the k-Means and DBSCAN.

3.1 K-Means Clustering Implementation

The K-Means clustering algorithm was developed from first principles in Python, with an emphasis on modularity and reproducibility. The implementation comprises several key stages:

- **Data Preprocessing:**
  - **Numerical Feature Extraction:** Only numerical features are retained for clustering, while categorical features are systematically detected and excluded.
  - **Missing Value Handling:** Rows containing missing values are removed to ensure data integrity.
  - **Feature Scaling:** Standard normalization (z-score) is applied to all features, promoting uniformity in the clustering process.
- **Centroid Initialization:**
  - **Random Initialization:** Centroids are selected randomly within the bounds of the data.
  - **K-Means++ Initialization:** Centroids are chosen using a probabilistic approach based on squared distances, which enhances convergence and cluster quality.
- **Iterative Clustering Process:**
  - **Cluster Assignment:** Each data point is assigned to the nearest centroid using the Euclidean distance metric.
  - **Centroid Update:** Centroids are recalculated as the mean of all points assigned to each cluster.
  - **Convergence Check:** The algorithm terminates when the movement of centroids falls below a specified threshold (**convergence_threshold**) or when the maximum number of iterations (**max_iterations**) is reached.

- **Cluster Evaluation:**
  - **Inertia Calculation:** The sum of squared distances from each point to its assigned centroid (**inertia**) is computed to assess cluster compactness.
  - **Elbow Method:** Inertia values are calculated for varying numbers of clusters (k), and the optimal k is identified by locating the point of maximum curvature (the "elbow").
  - **Silhouette Score:** The mean silhouette score is computed to evaluate cluster separation, based on intra-cluster and nearest-cluster distances.
- **Operational Flexibility:**
  - **Fit, Predict, and Fit-Predict:** The implementation supports separate and combined operations for model fitting and cluster prediction.
  - **Reproducibility:** All random operations are controlled via a fixed seed (**random_state**) to ensure consistent results across runs.

This structured approach ensures that the K-Means implementation is robust, interpretable, and suitable for integration into broader data mining workflows.

3.2 DBSCAN Clustering Implementation

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm was implemented from scratch in Python, emphasizing modularity and parameter flexibility. The implementation consists of the following key stages:

- **Data Preprocessing:**
  - **Numerical Feature Extraction:** Only numerical features are retained for clustering; categorical features are systematically detected and excluded.
  - **Missing Value Handling:** Rows containing missing values are removed to maintain data integrity.

  - **Feature Scaling:** Standard normalization (z-score) is applied to all features, ensuring uniformity in distance calculations.

- **Core DBSCAN Algorithm:**
  - **Parameterization:** The algorithm is governed by two main parameters: **epsilon** (maximum neighborhood radius) and **min_samples** (minimum points required to form a dense region).
  - **Neighbor Identification:** For each point, neighbors within the **epsilon** radius are identified using the Euclidean distance metric.
  - **Core Point Detection:** Points with at least **min_samples** neighbors are designated as core points.
  - **Cluster Expansion:** Clusters are formed by recursively expanding from core points, aggregating all density-connected points.
  - **Noise Handling:** Points not assigned to any cluster are labeled as noise (**-1**).

- **Cluster Evaluation:**
  - **Silhouette Score:** The mean silhouette score is computed for non-noise points to assess cluster separation and quality.
  - **Noise Ratio:** The proportion of points classified as noise is calculated to evaluate clustering effectiveness.

- **Parameter Optimization:**
  - **Automatic Range Selection:** Suitable ranges for **epsilon** and **min_samples** are determined using k-distance graphs and feature dimensionality.
  - **Grid Search:** Multiple parameter combinations are evaluated, and results are ranked by a combined score that weights silhouette score and penalizes excessive noise.

- **Operational Flexibility:**
  - **Fit and Fit-Predict:** The implementation supports both fitting the model and direct cluster label prediction.
  - **Result Reporting:** Outputs include cluster labels, core sample indices, number of clusters, and noise count for comprehensive analysis.

4 Hardware Specification

This experiment were run on a Laptop with 2.30 GHz Intel(R) Core(TM) i7 - 11800H (11th Gen) processor and 16Gbyte Ram. The operating system was Windows 11.

5 EXPERIMENTAL RESULT

Here I present the experimental result of the implementation of k-Means and DBSCAN.

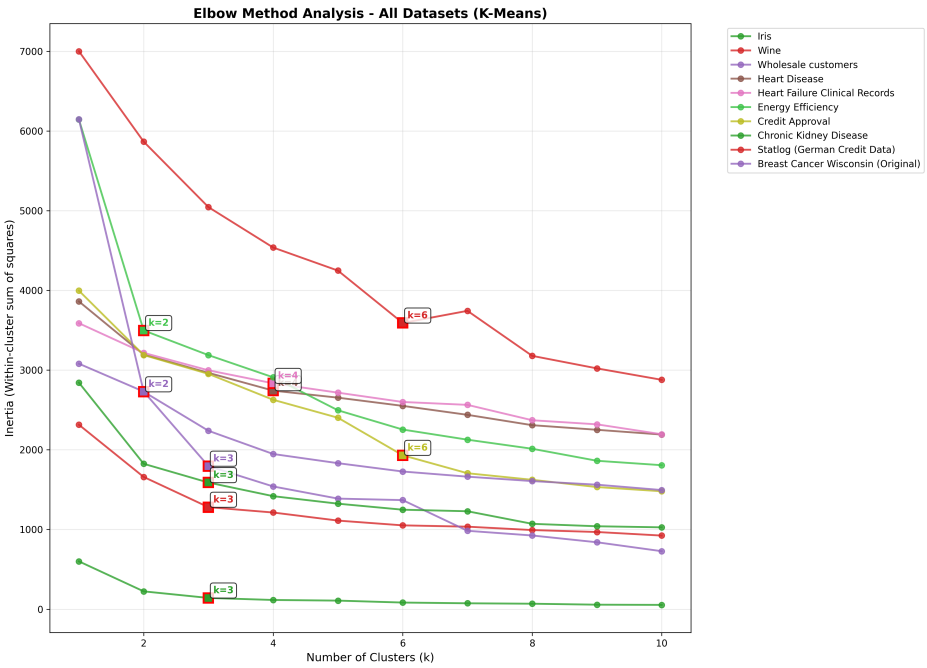
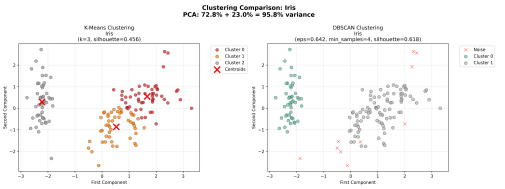
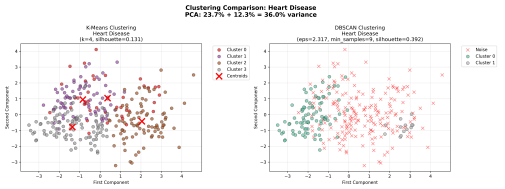


Figure 1: Elbow Method for All Datasets

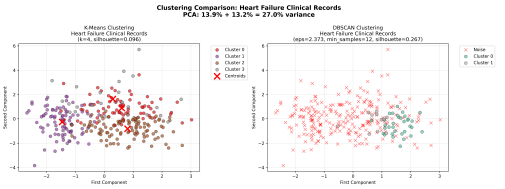


(a) Iris Dataset

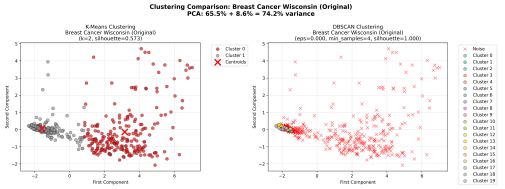


(b) Heart Disease Dataset

Figure 2: Clustering Visualizations (Part 1)

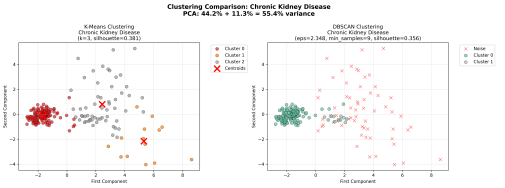


(a) Heart Failure Clinical Records

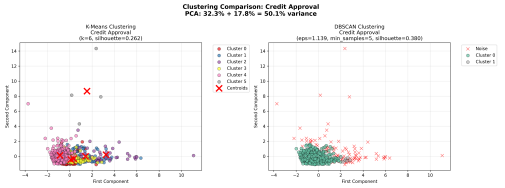


(b) Breast Cancer Wisconsin Dataset

Figure 3: Clustering Visualizations (Part 2)

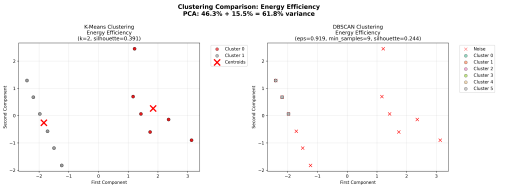


(a) Chronic Kidney Disease Dataset

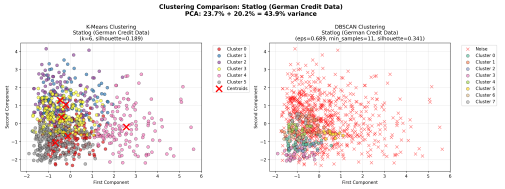


(b) Credit Approval Dataset

Figure 4: Clustering Visualizations (Part 3)

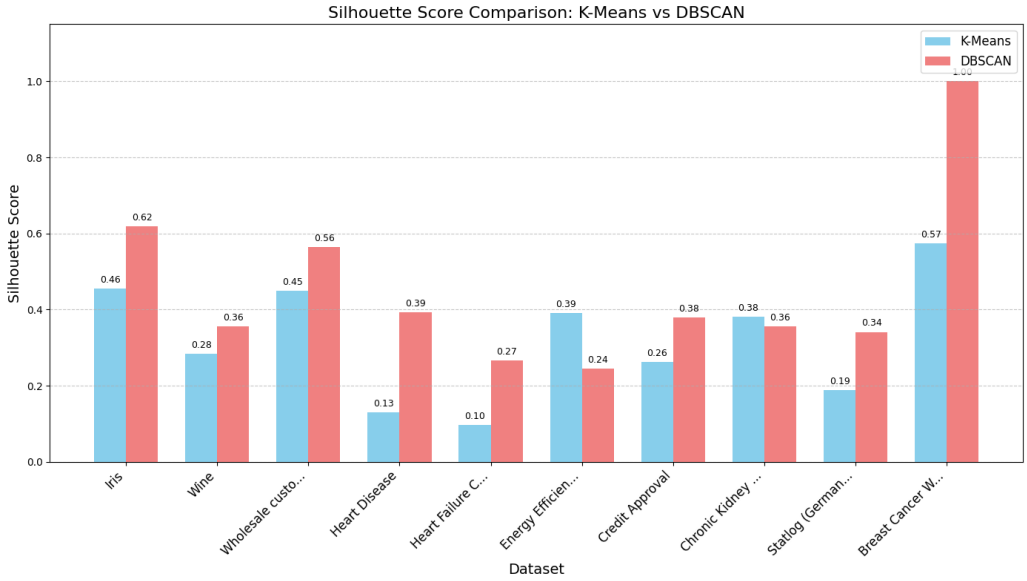


(a) Energy Efficiency Dataset



(b) Statlog German Credit Dataset

Figure 5: Clustering Visualizations (Part 4)



(a) Sihouette Comparison Graph

6 Conclusion

This comparative study between K-Means and DBSCAN clustering algorithms across multiple benchmark datasets reveals distinct strengths and limitations for each method. DBSCAN consistently outperforms K-Means in terms of silhouette score on the majority of datasets (8 out of 10), demonstrating its ability to detect clusters of arbitrary shapes and handle noise effectively. The presence of noise points identified by DBSCAN further highlights its robustness to outliers, which is a notable advantage over K-Means, where every point is forcibly assigned to a cluster regardless of its fit.

K-Means, while computationally efficient and straightforward, tends to perform better only when clusters are well-separated, roughly spherical, and of similar size, as evidenced by its higher silhouette scores on the Energy Efficiency and Chronic Kidney Disease datasets. Its requirement to prespecify the number of clusters and sensitivity to initialization remain inherent drawbacks.

Overall, DBSCAN emerges as the preferred clustering technique for diverse, real-world datasets exhibiting complex cluster structures and noise. However, careful tuning of its parameters ε and *min_samples* is crucial for optimal performance. K-Means remains a viable choice for simpler datasets

or when cluster count is known and clusters approximate spherical shapes.

Hence, the choice of clustering algorithm should be guided by the data characteristics and the specific application requirements, with DBSCAN recommended for datasets with irregular cluster shapes and noise, and K-Means suited for well-defined, spherical clusters.

References

[1] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu,
A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,
Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD), pp. 226–231, 1996.
Institute for Computer Science, University of Munich.
Emails: {ester | kriegel | sander | xwxu}@informatik.uni-muenchen.de