Clustering Algorithms – Analysis and Implementation

Gideon Peters
Concordia University
Montreal, Canada
gi_peter@live.concordia.ca

Bhoomiben Bhatt
Concordia University
Montreal, Canada
bhattbhoomi0111@gmail.com

Anjolaoluwa Lasekan
Concordia University
Montreal, Canada
anjolaoluwa.lasekan@mail.concordia.ca

Masum Mewaz
Concordia University
Montreal, Canada
MA_Newaz@live.concordia.ca

1 Introduction

Clustering is an unsupervised machine learning technique that partitions a dataset into groups, or clusters, based on the similarity of data points [31, 56, 74]. Unlike supervised learning, which relies on labeled data, clustering discovers underlying patterns in data without prior knowledge of class labels [74]. The primary goal of clustering is to maximize intra-cluster similarity while minimizing inter-cluster similarity, ensuring that points within the same cluster are more alike than those in different clusters [66]. This makes clustering a valuable tool for exploratory data analysis [15], pattern recognition [60], and data-driven decision-making [5].

Clustering is widely applied across various domains. In marketing, businesses use clustering for customer segmentation, grouping consumers based on purchasing behavior to tailor targeted campaigns [59]. In cybersecurity, anomaly detection algorithms leverage clustering to identify unusual patterns indicative of fraud or cyber threats [1]. In computer vision, clustering helps in image segmentation by grouping pixels with similar features [50]. Similarly, in bioinformatics, clustering is used for gene expression analysis to uncover genetic similarities among biological samples [17].

Several clustering algorithms exist, each with unique strengths and weaknesses. Partition-based methods, such as k-means, divide data into k clusters by iteratively minimizing intra-cluster variance [64]. Hierarchical clustering constructs a nested hierarchy of clusters [21]. Density-based methods, like DBSCAN, identify clusters as regions of high-density points, making them robust to noise and outliers [6]. Choosing the right clustering algorithm depends on factors such as data distribution, scalability, and computational complexity [65].

Our report provides a structured exploration of clustering algorithms, detailing their base algorithms, computational efficiency, distance metrics, and evaluation methods. We also examine specific considerations for k-means, such as initialization strategies, convergence properties, and limitations. Additionally, we discuss various clustering evaluation metrics and their applicability to different algorithms.

2 Computation of the Dissimilarity Coefficients

In this section, we examine dissimilarity coefficients, focusing on the Euclidean distance and its weighted variant. The Weighted Euclidean Distance incorporates feature-specific weights to reflect the importance of features, while normalization ensures that features with varying scales contribute equally. These steps are crucial for accurate and meaningful clustering results in real-world applications.

In clustering, *dissimilarity coefficients* are quantitative measures used to determine how different two data points are from each other [62]. These coefficients play a crucial role in clustering algorithms, where the goal is to group similar data points together while keeping dissimilar ones apart. By defining a numerical measure of dissimilarity, clustering methods can effectively segment data into meaningful clusters based on the underlying patterns in the dataset [58].

Several types of dissimilarity coefficients exist, each suited for different types of data and clustering approaches [44, 49]. Examples are:

- **Euclidean distance:** Determines the proximity between observations by drawing a straight line between pairs of observations [68].
- **Manhattan Distance:** Calculates the absolute differences between feature values, useful when dealing with high-dimensional spaces with sparse data.
- **Cosine Dissimilarity:** Measures the angular difference between vectors, often used in text and document clustering where magnitude is less relevant than direction.
- **Jaccard Dissimilarity:** Used primarily for categorical and binary data, measuring the proportion of non-overlapping features.
- **Minkowski Distance:** A generalization of Euclidean and Manhattan distances, with flexibility based on the choice of the exponent parameter.

$$d(x, y) = \sqrt{\sum_{j=1}^{d} (x_j - y_j)^2} \tag{1}$$

Of all these methods, the Euclidean distance is the most common and widely used metric [49]. Equation 1 shows the mathematical formula for this. Where:

- x_j and y_j are the feature values of two data points across d dimensions.

In many real-world applications, different features may have varying degrees of importance. For instance, in medical diagnosis, blood pressure and heart rate may carry more significance than height when clustering patient data [12, 25, 40]. To account for this, the ***Weighted Euclidean Distance*** modifies the standard Euclidean distance by assigning a weight to each feature:

$$d_w(x, y) = \sqrt{\sum_{j=1}^{d} w_j (x_j - y_j)^2} \tag{2}$$

Equation 2 shows the mathematical representation for the weighted Euclidean distance. Where:

- w_j is the weight for feature j. Higher w_j values give greater influence to that feature in distance calculations. Proper weighting helps emphasize *more relevant* features while reducing noise from *less significant* ones.

This variation of the Euclidean distance is very important [26, 53]. By incorporating weights, it refines the similarity measurement to better reflect underlying relationships in the data as opposed to the non-weighted version in Equation 1 that treats all features equally, which is not the case in practice.

Weights should be carefully chosen based on domain expertise or through iterative experimentation [14, 57, 79]. For example, in a customer segmentation problem, you might assign a higher weight to "annual income" compared to "age" if income is considered more relevant for clustering. Weighting allows us to emphasize the importance of certain features, leading to clusters that are more aligned with the specific goals of an analysis.

Consequently, ***normalization*** is also an important step. Normalization is particularly important when dealing with datasets where features have varying scales or units of measurement [69]. It is usually done when preprocessing data for analyses involving distance metrics [52], especially the Weighted Euclidean Distance. Without proper normalization, features with larger magnitudes (e.g., income in dollars vs. age in years) can disproportionately influence the distance calculation, leading to biased clustering results. For example, if one feature ranges from 0 to 100,000 and another from 0 to 10, the former will dominate the distance metric, rendering the latter almost irrelevant.

Normalization ensures that each feature contributes appropriately, reflecting its true importance and leading to more accurate and meaningful analytical outcomes. There are various methods to achieve this, the most common are:

- **Min-Max Scaling**: Rescales values to $[0, 1]$, ensuring no feature overpowers others. It scales the data to a fixed range but is sensitive to outliers [52].
- **Z-score Standardization**: Centers data with mean 0 and standard deviation 1, making distances comparable and it less sensitive to outliers [28].
- **Robust Scaling**: Uses median and interquartile range, reducing the influence of outliers. It is robust to outliers [52].

Existing literature emphasizes the importance of choosing a normalization method that is appropriate for the distribution of the data being considered [52]. For example, robust scaling is recommended if the data contains outliers.

In this section, we have discussed the role of dissimilarity coefficients in clustering, focusing on the Euclidean distance and its weighted variant. The Weighted Euclidean Distance allows for the incorporation of feature-specific weights, ensuring that more relevant features have a greater influence on clustering outcomes.

However, the effectiveness of these distance metrics depends on proper normalization, which addresses issues arising from features with varying scales or units. Techniques like Min-Max Scaling, Z-score Standardization, and Robust Scaling ensure that all features contribute equally, preventing bias in clustering results. Together, weighted distance measures and normalization enhance the accuracy and interpretability of clustering algorithms, enabling more meaningful insights from real-world datasets.

3 Algorithm Descriptions and Complexity Analysis

This section provides a detailed description and complexity analysis of various clustering algorithms. Each algorithm is categorized based on its underlying methodology. The focus is on the *base versions* of these algorithms, with refinements to be explored in Part II.

3.1 Density-Based Clustering

Density-based clustering algorithms group data points based on the density of their surrounding points [36]. These methods identify regions with high data point density as clusters and separate them from regions with low density, often handling noise and outliers effectively. Unlike partition-based methods like k-means, which rely on a predefined number of clusters, density-based methods identify dense regions in the data space and classify sparse regions as noise. There are various density-based clustering algorithms, including DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), and DENCLUE (Density-Based Clustering Using Kernel Density Estimation). In this section, we highlight DBSCAN and OPTICS along with their algorithm descriptions and their detailed complexity analysis.

3.1.1 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Description: DBSCAN is one of the most widely used density-based clustering algorithms [19]. It was developed to cluster data with arbitrary shapes while handling noise in both spatial and high-dimensional non-spatial databases [35].

It defines clusters based on two parameters:

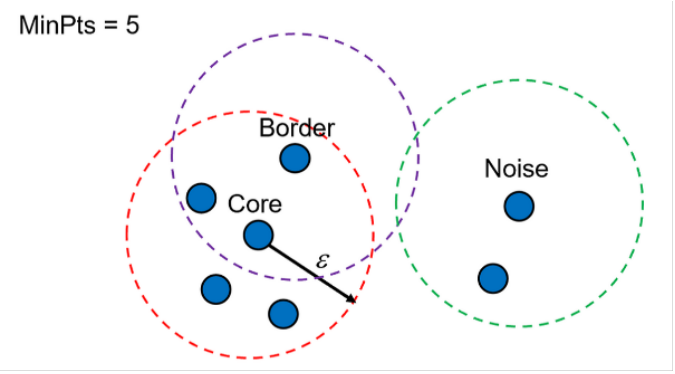
- ε (epsilon): The radius within which points are considered neighbors.
- MinPts (minimum points): The minimum number of points required to form a dense region (i.e., a cluster).

The algorithm identifies core points, border points, and noise points.

A point is considered a **core point** if it has at least MinPts within its ε-neighborhood. If a point is within ε of a core point, it is considered **density reachable**. A **boundary point** is a point that lies within the ε-neighborhood of a core point but does not have enough neighbors to be a core point itself. A noise point is a point not within any cluster. Clusters are formed by connecting density-reachable points, while points that do not belong to any cluster are classified as **noise** [6].

Figure 1 demonstrates these fundamental concepts by clustering data points based on density. The parameter **MinPts = 5** defines the minimum number of points required to form a dense region within a radius ϵ. **Core points** are those with at least `MinPts` neighbors within ϵ, forming the dense core of a cluster. **Border points** lie within the ϵ-neighborhood of a core point but do not

Figure 1: Visual description of DBSCAN clustering



have enough neighbors to qualify as core points. Points that do not meet the density criteria and are not reachable from any core point are classified as **noise points**. Thus visualizing DBSCAN's ability to identify clusters of arbitrary shapes while distinguishing and isolating noise.

Algorithm: Algorithm 1 highlights the base DBSCAN algorithm in detail. It requires the following parameters:

- **Inputs:**
  - DataSet: Dataset of points
  - ε: Radius (neighborhood distance threshold)
  - MinPts: Minimum number of points required to form a dense region
- **Output:**
  - Clusters of points with core points, boundary points, and noise labeled separately

The DBSCAN algorithm begins by initializing the cluster counter C to zero (Line 1). It then iterates through each unvisited point P in the dataset D. For each point P, it is marked as visited, and its neighbors within a radius ε are retrieved using the `regionQuery(`P`,` ε`)` function. If the number of neighbors is less than `MinPts`, P is classified as **NOISE**. Otherwise, a new cluster C is created, and P is added to this cluster. The algorithm then expands the cluster by calling the `expandCluster()` function.

The `expandCluster()` function assigns P to the current cluster C and iterates through its neighboring points. For each neighboring point P', if P' has not been visited, it is marked as visited, and its neighbors are retrieved using `regionQuery(`P'`,` ε`)`. If P' has at least `MinPts` neighbors, it is labeled as a **CORE_POINT**, and its neighbors are added to the current neighborhood for further processing. If P' does not meet the `MinPts` threshold, it is classified as a **BOUNDARY_POINT**. If P' has not yet been assigned to any cluster, it is added to the current cluster C. This process continues iteratively until all points in the neighborhood are processed.

The algorithm terminates when all points in the dataset have been visited. The final output consists of clusters containing **core points** and **boundary points**, while **noise points** remain unassigned to any cluster.

Complexity Analysis:

Algorithm 1 DBSCAN Algorithm [32, 35]

Require: Dataset D, distance threshold ε, minimum points $MinPts$
Ensure: Clusters

```
1:  C ← 0                                              ▷ Initialize cluster counter
2:  for each unvisited point P in dataset D do
3:      Mark P as visited
4:      NeighborPts ← REGIONQUERY(P, ε)
5:      if size of NeighborPts < MinPts then
6:          Mark P as NOISE
7:      else
8:          C ← next cluster
9:          EXPANDCLUSTER(P, NeighborPts, C, ε, MinPts)
10: Return clusters
11: function EXPANDCLUSTER(P, NeighborPts, C, ε, MinPts)
12:     Add P to cluster C
13:     for each point P' in NeighborPts do
14:         if P' is not visited then
15:             Mark P' as visited
16:             NeighborPts' ← REGIONQUERY(P', ε)
17:             if size of NeighborPts' ≥ MinPts then
18:                 NeighborPts ← NeighborPts ∪ NeighborPts'
19:         if P' is not yet a member of any cluster then
20:             Add P' to cluster C
21: function REGIONQUERY(P, ε)
22:     Return all points within P's ε-neighborhood (including P)
```

(1) **Finding Neighbors:** For each point P, the algorithm retrieves all points within the ε-radius using the `regionQuery(`P`,` ε`)` function. In a naïve approach, this involves computing pairwise Euclidean distances between P and all other points in the dataset D. For N points, this results in:

$$\sum_{i=1}^{N} O(N) = O(N^2) \tag{3}$$

Thus, without optimization, the neighbor search contributes $O(N^2)$ complexity.
However, spatial indexing structures like **k-d trees** or **ball trees** can significantly improve efficiency. In low-dimensional spaces, a single range query using a k-d tree takes $O(\log N)$ time. Since this search is performed for each of the N points, the total complexity reduces to:

$$O(N \log N) \tag{4}$$

In high-dimensional spaces, however, the **curse of dimensionality** weakens the efficiency of spatial indexing structures. The search time per query approaches $O(N)$, leading to an overall complexity of:

$$O(N^2) \tag{5}$$

(2) **Cluster Expansion:** Once a core point is found, the `expandCluster()` function is called. This function iterates through the neighbors of the core point and recursively expands the cluster. In the worst case, if all points belong to a single dense cluster, every point is visited once and expanded. Each point

contributes $O(1)$ operations, and for N points, this results in:

$$O(N) \tag{6}$$

Since every point is assigned to a cluster only once, this step does not add extra complexity beyond the neighbor search step.

(3) **Final Complexity Calculation:** The overall complexity of DBSCAN depends on the approach used for neighbor search:
- **Brute-force approach:** If pairwise distance calculations are used, the total complexity is dominated by the neighbor search step:

$$O(N^2) + O(N) = O(N^2) \tag{7}$$

- **Optimized approach (using k-d trees):** If spatial indexing is used in low-dimensional spaces, the complexity improves significantly [74]:

$$O(N \log N) + O(N) = O(N \log N) \tag{8}$$

However, in high-dimensional cases, spatial indexing loses efficiency, reverting the complexity to:

$$O(N^2) \tag{9}$$

Advantages:

- DBSCAN is particularly effective in identifying and isolating noise or outliers. It does not force every point into a cluster, unlike other algorithms that may misclassify noise as part of a cluster [67, 70].
- Unlike algorithms like K-Means, DBSCAN does not require the number of clusters to be specified in advance. This makes it flexible and adaptable to different datasets [20].
- DBSCAN excels at finding clusters of various shapes, not just spherical, making it suitable for complex, non-linear data patterns.
- With optimizations like the use of spatial indexing structures (e.g., KD-trees), DBSCAN can efficiently handle large datasets [20].

Disadvantages:

- DBSCAN's performance heavily depends on the selection of its key parameters, epsilon (ε) and minPts. Incorrect parameter choices can result in poor clustering or excessive noise detection [6].
- DBSCAN struggles when clusters have varying densities, as a single set of parameters might not work well for all clusters in the dataset [45].
- In high-dimensional spaces, DBSCAN may lose its effectiveness due to the "curse of dimensionality," as the distance between points becomes less distinguishable.
- While DBSCAN is efficient for moderately sized datasets, its computational complexity increases with the number of points and dimensions, especially without proper indexing.

3.1.2 OPTICS (Ordering Points to Identify the Clustering Structure)

Description: OPTICS is a density-based clustering algorithm that extends DBSCAN by overcoming its sensitivity to parameter selection. Instead of forming explicit clusters [4, 19]. Unlike DBSCAN, which directly forms clusters, OPTICS creates an ordered list of data points along with their *reachability distances*, allowing hierarchical cluster extraction. This makes it useful for detecting clusters of varying densities.

OPTICS works by expanding clusters in a manner similar to DBSCAN but maintains a priority queue to determine the next best point to process. It does not assign explicit cluster labels during execution but provides a reachability plot that helps identify cluster structures. The key idea behind OPTICS is to create a *"reachability plot,"* which is a visualization of the density-based clustering structure. The algorithm computes two values for each point:

- **Core Distance**: The smallest distance ϵ' such that the ϵ'-neighborhood of the point contains at least $MinPts$ points. If the point is not a core point, the core distance is undefined.
- **Reachability Distance**: The minimum distance at which a point can be reached from another core point. This distance is used to order the points in the reachability plot.
  - If point is within Core distance, then it's reachability distance is same as core distance.
  - If point is far away from ε distance, then it's reachability distance is Euclidian distance between those two points.

Figure 2: Visual description of OPTICS clustering [24]

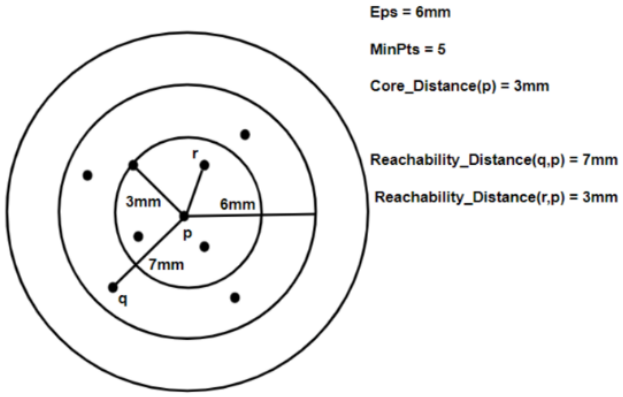


Figure 2 illustrates the OPTICS clustering process with a visual representation of core and reachability distances. The parameters used are $Eps = 6mm$ and $MinPts = 5$. The core distance of point p is $3mm$, indicating the smallest radius required for p to be a core point. The reachability distances from p to points q and r are $7mm$ and $3mm$, respectively, reflecting the density-based relationships between the points. This diagram highlights the concept of density-reachability and the ordering of points in OPTICS.

Point p is a core point because its core distance ($3mm$) is less than Eps ($6mm$), and it has at least $MinPts$ (5) points within its neighborhood. This demonstrates how OPTICS identifies dense regions in the dataset. The reachability distance from p to q is $7mm$, which is greater than Eps, suggesting that q lies outside the dense region around p and may belong to a different cluster or be an outlier. In contrast, the reachability distance from p to

r is $3mm$, indicating that r is part of the same dense region as p. The varying reachability distances highlight how OPTICS can distinguish between points in dense regions (e.g., r) and those in sparser regions (e.g., q). This is particularly useful for datasets with varying densities, where traditional clustering algorithms might struggle [19].

In OPTICS clustering, clusters are represented as valleys in the reachability plot [4], where deeper valleys signify denser clusters. Points within a cluster have minimal reachability distances to their nearest neighbors. The optimal epsilon value can be determined from the lowest points of these valleys.

Figure 3: Example of a reachability plot for OPTICS

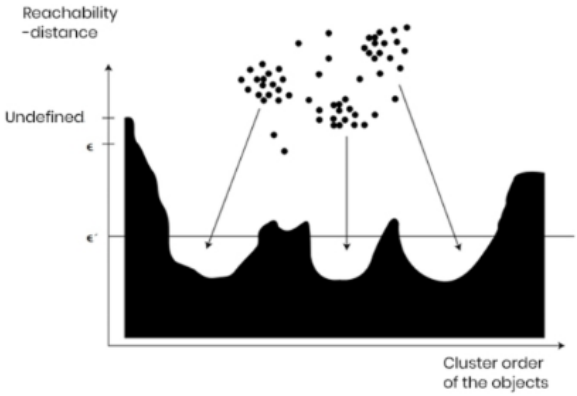


Figure 3 is an example of a reachability plot, the *undefined* reachability distances often correspond to noise or outliers, which are not part of any cluster, whereas points with defined reachability distances are part of dense regions and contribute to the formation of clusters. The cluster order of objects refers to the sequence in which points are processed and added to the reachability plot. This order reflects the density-based structure of the data, with points in dense regions appearing earlier in the sequence and points in sparser regions or outliers appearing later. This ordering is crucial for generating the reachability plot, which visually represents the clustering structure of the dataset.

Algorithm: Algorithm 2 highlights the base OPTICS algorithm in detail. It requires the following parameters:

- **Inputs:**
  - DataSet: Dataset containing n points
  - ε: Maximum neighborhood radius for core distance computation
  - MinPts: Minimum number of neighbors required to form a dense region
- **Output:**
  - Ordered List of Points with their Reachability Distance

The OPTICS algorithm operates as follows: Initially, each point within the dataset *DataSet* is assigned an undefined `accessibilityMeasure`, ensuring that no prior reachability information influences the process. The algorithm then iterates over unprocessed elements, selecting an element E and determining its `NeighborSet` within the given radius ε. Once identified, E is marked as `PROCESSED` and appended to `sortedList`, ensuring that all processed elements are stored in order of discovery.

The key step in the process is evaluating whether E qualifies as a core point using the function `coreMeasure(`E`, `ε`, `$MinPts$`)`. If E satisfies the core criterion (i.e., it has at least $MinPts$ neighbors within ε), the `UPDATE` function is invoked. This function employs a priority queue, referred to as `Queue`, which dynamically organizes points based on their reachability distance. The `UPDATE` function modifies the reachability distances of neighboring points and inserts them into `Queue` for further evaluation, ensuring that points in dense regions are processed earlier.

The `UPDATE` function refines the accessibility relationships among neighboring points by computing the `coreDist` of E and adjusting the accessibility of each neighbor within `NeighborSet`. For every neighbor that remains `UNPROCESSED`, the algorithm calculates a new accessibility measure, `newAccessMeasure`, as the greater value between `coreDist` and the distance from E to the neighbor. If the neighbor has no previously assigned `accessibilityMeasure`, it is inserted into `Queue` with `newAccessMeasure`. However, if the neighbor is already in `Queue` with a higher reachability distance, its `accessibilityMeasure` is updated, and `Queue` is reorganized to reflect the improved distance. This ensures that the nearest and most densely connected points are processed first, leading to an ordered `sortedList` that captures the cluster hierarchy.

The final output consists of an ordered list of points with their reachability distances, which can be visualized in a reachability plot to reveal the underlying clustering structure. This hierarchical representation allows for flexible and adaptive cluster extraction without requiring a predefined number of clusters, making OPTICS highly effective for discovering arbitrarily shaped and nested clusters.

Complexity Analysis:

(1) **Neighbor Search:** For each point P in the dataset *DataSet*, the algorithm retrieves its neighbors within radius ε using the `findNeighbors(`P`, `ε`)` function. In a naïve approach, this involves computing pairwise Euclidean distances between P and all other points in *DataSet*. For N points, this results in:

$$\sum_{i=1}^{N} O(N) = O(N^2) \tag{10}$$

Thus, without optimization, the neighbor search contributes $O(N^2)$ complexity.
However, spatial indexing structures like **k-d trees** or **ball trees** can significantly improve efficiency. In low-dimensional spaces, a single range query using a k-d tree takes $O(\log N)$ time. Since this search is performed for each of the N points, the total complexity reduces to:

$$O(N \log N) \tag{11}$$

In high-dimensional spaces, the **curse of dimensionality** weakens the efficiency of spatial indexing structures. The

Algorithm 2 OPTICS Algorithm

Require: Dataset *DataSet*, radius ε, minimum points *MinPts*
Ensure: Ordered list of points with reachability distances

for each element E in *DataSet* **do**
  E.accessibilityMeasure ← UNDEFINED
for each unprocessed element E in *DataSet* **do**
  $NeighborSet \leftarrow$ findNeighbors(E, ε)
  Mark E as PROCESSED
  Append E to *sortedList*
  if coreMeasure$(E, \varepsilon, MinPts) \neq$ UNDEFINED **then**
    Queue ← empty priority structure
    UPDATE($NeighborSet$, E, *Queue*, ε, *MinPts*)
    for each *nextElement* in *Queue* **do**
      $NeighborSet' \leftarrow$ findNeighbors$(nextElement, \varepsilon)$
      Mark *nextElement* as PROCESSED
      Append *nextElement* to *sortedList*
      if coreMeasure$(nextElement, \varepsilon, MinPts) \neq$ UNDEFINED **then**
        UPDATE($NeighborSet'$, *nextElement*, *Queue*, ε, *MinPts*)
Return *sortedList*
function UPDATE($NeighborSet$, E, *Queue*, ε, *MinPts*)
  $coreDist \leftarrow$ coreMeasure$(E, \varepsilon, MinPts)$
  for each *neighbor* in $NeighborSet$ **do**
    if *neighbor* is UNPROCESSED **then**
      $newAccessMeasure \leftarrow \max(coreDist,$ distance$(E, neighbor))$
      if *neighbor*.accessibilityMeasure == UNDEFINED **then**
        neighbor.accessibilityMeasure ← *newAccessMeasure*
        Queue.insert(*neighbor*, *newAccessMeasure*)
      else
        if *newAccessMeasure* < *neighbor*.accessibilityMeasure **then**
          neighbor.accessibilityMeasure ← *newAccessMeasure*
          Queue.promote(*neighbor*, *newAccessMeasure*)

search time per query approaches $O(N)$, leading to an overall complexity of:

$$O(N^2) \tag{12}$$

(2) **Priority Queue Operations:** The algorithm uses a priority queue (e.g., a min-heap) to manage the order in which points are processed. Each insertion or update operation in the priority queue takes $O(\log N)$ time. In the worst case, every point is inserted into the queue once, and its accessibility measure may be updated multiple times. Assuming each point is updated $O(1)$ times on average, the total complexity for priority queue operations is:

$$O(N \log N) \tag{13}$$

(3) **Cluster Expansion:** The `UPDATE` function iterates through the neighbors of each core point and updates their accessibility measures. In the worst case, if all points belong to a single dense cluster, every point is visited once, and its neighbors are processed. Each point contributes $O(1)$ operations, and for N points, this results in:

$$O(N) \tag{14}$$

(4) **Final Complexity Calculation:** The overall complexity of the OPTICS algorithm depends on the approach used for neighbor search:

- **Brute-force approach:** If pairwise distance calculations are used, the total complexity is dominated by the neighbor search step:

$$O(N^2) + O(N \log N) + O(N) = O(N^2) \tag{15}$$

- **Optimized approach (using k-d trees):** If spatial indexing is used in low-dimensional spaces, the complexity improves significantly:

$$O(N \log N) + O(N \log N) + O(N) = O(N \log N) \tag{16}$$

However, in high-dimensional cases, spatial indexing loses efficiency, reverting the complexity to:

$$O(N^2) \tag{17}$$

Advantages:

- OPTICS (Ordering Points to Identify the Clustering Structure) is more robust than DBSCAN when dealing with clusters of varying densities. It can identify clusters that have different densities within the same dataset [20].
- Similar to DBSCAN, OPTICS does not require the number of clusters to be specified in advance, which enhances its flexibility for various types of data.
- OPTICS provides a reachability plot that visually captures the clustering structure of the data, helping to detect clusters and subclusters more effectively [20].
- Like DBSCAN, OPTICS can detect clusters of arbitrary shape, making it suitable for datasets where traditional methods (like K-Means) fail.

Disadvantages:

- Although OPTICS can handle varying densities better than DBSCAN, its performance still depends on the selection of parameters, particularly the minPts value. Poor parameter choices can lead to suboptimal clustering.
- The reachability plot produced by OPTICS can be difficult to interpret, especially for large datasets or datasets with complex structures, requiring extra effort to extract useful insights.
- OPTICS can be computationally expensive, particularly for large datasets, as it requires pairwise distance computations, which may slow down the clustering process.
- Like DBSCAN, OPTICS struggles with high-dimensional data due to the *curse of dimensionality*, where distance metrics lose their effectiveness as the number of dimensions increases.

3.2 Distribution-Based Clustering

Distribution-based clustering algorithms group data points based on their probability distribution within the data space [55]. These methods assume that the data is generated from a mixture of probability distributions, and they aim to identify the underlying distributions to form clusters. Each cluster is modeled by a distinct distribution, and the goal is to estimate the parameters of these distributions to group similar data points. A popular example of this approach is the Gaussian Mixture Model (GMM), which assumes that the data points are generated from a combination of Gaussian distributions [55]. Unlike partition-based methods such as k-means, distribution-based methods provide probabilistic membership assignments, allowing data points to belong to multiple clusters with varying degrees of confidence. This makes them particularly useful for capturing complex data structures and overlapping clusters [20]. In this section, we focus on the Gaussian Mixture Model, including its algorithmic description, and a detailed complexity analysis.

3.2.1 Gaussian Mixture Model (GMM)

Description. A **Gaussian Mixture Model (GMM)** is a probabilistic model that assumes data points are generated from a mixture of several Gaussian distributions with unknown parameters [82]. This model effectively captures subpopulations within a larger population. For instance, when analyzing human height, it is common to model height as a normal distribution for each gender, with average heights of roughly 5'10" for males and 5'5" for females. If gender information is not provided, the overall height distribution would resemble a combination of two normal distributions, each with distinct means and variances. A model that incorporates this assumption is known as a Gaussian Mixture Model (GMM) [10].

Algorithm. Algorithm 3 highlights the base GMM algorithm in detail. It requires the following parameters:

- **Inputs**:
  - DataSet: Dataset containing n points.
  - K: Number of clusters.
  - ϵ: Convergence threshold for log-likelihood.
- **Output**:
  - Cluster assignments for each data point.
  - Model parameters $\{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$: Mixing coefficients, means, and covariance matrices for each cluster.

The steps for using GMM in clustering are as follows [82]:

- **Initialize**: Specify the number of clusters K and initialize parameters π_k, μ_k, and Σ_k.
- **E-step**: Compute posterior probabilities γ_{ik} for each data point x_i and cluster k.
- **M-step**: Update parameters π_k, μ_k, and Σ_k using γ_{ik}.
- **Check Convergence**: Repeat E-step and M-step until the change in log-likelihood $\ell(\theta)$ is below a threshold ϵ.
- **Assign Clusters**: Assign each x_i to the cluster k with the highest γ_{ik}.

For each of these steps, some calculations are important and to be used in the algorithm, they are as follows:

1. Posterior Probability (E-step). The posterior probability γ_{ik} represents the probability that data point x_i belongs to cluster k. It is computed using Bayes' theorem:

$$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$$

- π_k: Mixing coefficient (prior probability) for cluster k. It represents the weight of cluster k in the mixture.
- $\mathcal{N}(x_i \mid \mu_k, \Sigma_k)$: Gaussian probability density function (PDF) for cluster k, evaluated at x_i.
- $\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i \mid \mu_j, \Sigma_j)$: Normalization term ensuring that $\sum_{k=1}^{K} \gamma_{ik} = 1$.

2. Gaussian Probability Density Function(PDF). The Gaussian PDF for a data point x_i given cluster parameters μ_k (mean) and Σ_k (covariance matrix) is defined as:

$$\mathcal{N}(x_i \mid \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x_i - \mu_k)^T \Sigma_k^{-1}(x_i - \mu_k)\right)$$

- d: Dimensionality of the data.
- $|\Sigma_k|$: Determinant of the covariance matrix Σ_k.
- Σ_k^{-1}: Inverse of the covariance matrix Σ_k.
- $(x_i - \mu_k)^T \Sigma_k^{-1}(x_i - \mu_k)$: Mahalanobis distance between x_i and μ_k, which measures the distance between x_i and the mean μ_k, weighted by the covariance Σ_k.

3. Mixing Coefficients Update (M-step). The mixing coefficients π_k are updated as the average posterior probability for cluster k:

$$\pi_k = \frac{1}{n} \sum_{i=1}^{n} \gamma_{ik}$$

- n: Total number of data points.
- γ_{ik}: Posterior probability of data point x_i belonging to cluster k.
- π_k: Updated mixing coefficient for cluster k, representing the proportion of data points assigned to cluster k.

4. Means Update (M-step). The mean μ_k for cluster k is updated as the weighted average of all data points, where the weights are the posterior probabilities γ_{ik}:

$$\mu_k = \frac{\sum_{i=1}^{n} \gamma_{ik} x_i}{\sum_{i=1}^{n} \gamma_{ik}}$$

- x_i: Data point.
- γ_{ik}: Posterior probability of x_i belonging to cluster k.
- μ_k: Updated mean for cluster k, representing the center of the cluster.

5. Covariance Matrices Update (M-step). The covariance matrix Σ_k for cluster k is updated as the weighted covariance of all data points, where the weights are the posterior probabilities γ_{ik}:

$$\Sigma_k = \frac{\sum_{i=1}^{n} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{n} \gamma_{ik}}$$

- $(x_i - \mu_k)(x_i - \mu_k)^T$: Outer product of the deviation of x_i from the mean μ_k, representing the scatter of data points around the mean.

- γ_{ik}: Posterior probability of x_i belonging to cluster k.
- Σ_k: Updated covariance matrix for cluster k, representing the shape and orientation of the cluster.

6. Log-Likelihood. The log-likelihood measures how well the model fits the data and is used to check for convergence:

$$\ell(\theta) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x_i \mid \mu_k, \Sigma_k) \right)$$

- θ: Set of all model parameters $\{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$.
- $\sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x_i \mid \mu_k, \Sigma_k)$: Probability density of x_i under the mixture model.
- $\ell(\theta)$: Log-likelihood of the data given the model parameters, used to monitor convergence.

Algorithm 3 Gaussian Mixture Model (GMM)

Require: Dataset $X = \{x_1, x_2, \ldots, x_n\}$, number of clusters K, convergence threshold ϵ

Ensure: Cluster assignments, mixture parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$

Initialize mixture parameters:
  Mixing coefficients $\pi_k = \frac{1}{K}$ for $k = 1, 2, \ldots, K$
  Means μ_k randomly selected from X
  Covariance matrices Σ_k set to identity matrices
repeat
  E-step (Expectation):
  Compute posterior probabilities γ_{ik} for each data point x_i and cluster k:

$$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$$

  where $\mathcal{N}(x_i \mid \mu_k, \Sigma_k)$ is the Gaussian probability density function.
  M-step (Maximization):
  Update mixture parameters:
    Mixing coefficients:

$$\pi_k = \frac{1}{n} \sum_{i=1}^{n} \gamma_{ik}$$

    Means:

$$\mu_k = \frac{\sum_{i=1}^{n} \gamma_{ik} x_i}{\sum_{i=1}^{n} \gamma_{ik}}$$

    Covariance matrices:

$$\Sigma_k = \frac{\sum_{i=1}^{n} \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{n} \gamma_{ik}}$$

  Compute the log-likelihood:

$$\ell(\theta) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x_i \mid \mu_k, \Sigma_k) \right)$$

until Change in log-likelihood $|\ell(\theta_{\text{new}}) - \ell(\theta_{\text{old}})| < \epsilon$
return Cluster assignments γ_{ik}, mixture parameters θ

Complexity Analysis:

(1) **Initialization:** The initialization step involves setting the mixing coefficients π_k, means μ_k, and covariance matrices Σ_k. This step has a complexity of:

$$O(K \cdot d^2) \tag{18}$$

where K is the number of clusters and d is the dimensionality of the data. This is because each covariance matrix Σ_k is a $d \times d$ matrix.

(2) **E-step (Expectation):** In the E-step, the posterior probabilities γ_{ik} are computed for each data point x_i and cluster k. This involves evaluating the Gaussian probability density function $\mathcal{N}(x_i \mid \mu_k, \Sigma_k)$ for all n data points and K clusters. The complexity of this step is:

$$O(n \cdot K \cdot d^2) \tag{19}$$

This is because computing the Gaussian probability density function requires $O(d^2)$ operations due to the inversion and determinant of the covariance matrix Σ_k.

(3) **M-step (Maximization):** In the M-step, the mixture parameters π_k, μ_k, and Σ_k are updated. The complexity of this step is:

$$O(n \cdot K \cdot d^2) \tag{20}$$

This is because updating the means μ_k and covariance matrices Σ_k involves summing over all n data points and performing matrix operations for each of the K clusters.

(4) **Log-Likelihood Computation:** The log-likelihood $\ell(\theta)$ is computed at each iteration to check for convergence. This involves evaluating the Gaussian probability density function for all n data points and K clusters, resulting in a complexity of:

$$O(n \cdot K \cdot d^2) \tag{21}$$

(5) **Overall Complexity:** The overall complexity of the GMM algorithm depends on the number of iterations T required for convergence. Since each iteration consists of the E-step, M-step, and log-likelihood computation, the total complexity is:

$$O(T \cdot n \cdot K \cdot d^2) \tag{22}$$

Here:
- T: Number of iterations until convergence.
- n: Number of data points.
- K: Number of clusters.
- d: Dimensionality of the data.

The complexity is dominated by the E-step and M-step, both of which scale linearly with the number of data points n and the number of clusters K, and quadratically with the dimensionality d. The number of iterations T depends on the convergence threshold ϵ and the initialization of the parameters. In practice, T is often small for well-behaved datasets [42, 83]. For high-dimensional data (large d), the $O(d^2)$ term can become a bottleneck, making GMM computationally expensive.

Advantages:

- Unlike methods like K-Means, which assume spherical clusters, distribution-based clustering can manage more complex data shapes [20].

- Probabilistic Interpretations: It provides a clear statistical framework, assigning probabilities rather than rigid cluster assignments.
- The method can be adapted for different types of data through various probability distributions.

Disadvantages:

- These methods depend on the assumption that data follows a specific distribution, which, if incorrect, can reduce the accuracy of the clustering.
- Estimating distribution parameters, especially with large datasets, can be time-consuming.
- Sensitive to initialization.
- Assumes Gaussian distributions, which may not fit all data.
- Outliers can distort the model and negatively impact clustering results.

3.3 Centroid/Medoid-Based Clustering

Centroid/Medoid-based clustering algorithms are a class of partitioning methods that group data points into clusters by identifying central representatives for each cluster [71]. These representatives are either *centroids* (geometric centers) or *medoids* (actual data points), which serve as prototypes for their respective clusters [2, 39]. The goal of these algorithms is to minimize the dissimilarity between data points within a cluster and their corresponding central representative, thereby creating compact and well-separated clusters.

The most widely used centroid-based algorithm is *k-means*, which iteratively assigns data points to the nearest centroid and updates the centroids as the mean of the assigned points. While k-means is computationally efficient, it is sensitive to outliers and initialization [22], as centroids are influenced by all points in the cluster, including extreme values [34, 43]. In contrast, *k-Medoids* (also known as Partitioning Around Medoids or PAM) selects actual data points as medoids, making it more robust to noise and outliers. Medoids are chosen to minimize the total dissimilarity within clusters, providing a more interpretable and stable clustering solution.

Both k-means and k-Medoids require the number of clusters k to be specified in advance and rely on iterative optimization to converge to a local minimum [34, 43, 71]. However, they differ in their computational complexity and suitability for different types of data [27]. In this section, we provide a detailed description of the k-means and k-Medoids algorithms, including their steps, advantages, limitations, and computational complexity.

3.3.1 k-Means Clustering

Description. The primary use of this algorithm is to partition the dataset into k clusters [2, 39]. It minimizes the variance within each cluster, ensuring that data points in the same cluster exhibit high similarity while those in different clusters are distinct [2, 8, 39]. The algorithm operates iteratively to refine cluster assignments until an optimal solution is achieved.

Algorithm. The k-means clustering algorithm is described in Algorithm 4, it follows these key steps [22]:

(1) **Initialization:** Randomly select K data points from the dataset $\mathcal{D}$ as the initial cluster centroids:
$$C^{(0)} = \{c_1^{(0)}, c_2^{(0)}, \ldots, c_K^{(0)}\}$$
(2) **Cluster Assignment:** Assign each data point x_i to the nearest cluster centroid:
$$C_i = \arg\min_j \|x_i - c_j^{(t)}\|^2$$
where C_i represents the cluster assignment for point x_i, and $c_j^{(t)}$ is the centroid of cluster j at iteration t.
(3) **Centroid Update:** Recalculate each cluster centroid as the mean of all points assigned to it:
$$c_j^{(t+1)} = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$
where $|S_j|$ denotes the number of data points assigned to cluster j.
(4) **Convergence Check:** Repeat the assignment and update steps until the centroids converge, i.e., when:
$$\sum_{j=1}^{K} \|c_j^{(t+1)} - c_j^{(t)}\| < \epsilon$$
or until a predefined stopping criterion, such as a maximum number of iterations, is reached.

For example,

Customer	Annual Spending	Number of Visits
A	500	5
B	700	8
C	2000	25
D	1500	20
E	400	6

Table 1: Customer Spending and Visits

Let us take K1 = (500,5) and K2 = (2000,25). Now using Euclidean distance formula:

$$\sqrt{(X_0 - X_c)^2 + (Y_0 - Y_c)^2} \tag{23}$$

Here X_0 Observed value of annual spending. X_c is centroid value of annual spending. Similarly, Y_0 is observed value of visits and Y_c is centroid value of number of visits.

Euclidean distance for customer B:

$$k_1 = \sqrt{(700 - 500)^2 + (8 - 5)^2} = \sqrt{40009} \approx 200.02$$

$$k_2 = \sqrt{(700 - 2000)^2 + (8 - 25)^2} = \sqrt{1690289} \approx 1300.11$$

Since $k_1 < k_2$, customer B will be assigned to cluster k_1.

$$K_1 = \{A, B\}, \quad K_2 = \{C\}$$

Update centroid value using the mean formula:

$$\text{New Centroid of } K_1 = \left(\frac{X_1 + X_2}{2}, \frac{Y_1 + Y_2}{2}\right)$$

Algorithm 4 K-Means Clustering [22]

Input:
- A dataset $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$, where each x_i is a feature vector.
- K - The number of clusters to form.

Output:
- A set of K cluster centroids $C = \{c_1, c_2, \ldots, c_K\}$.
- A partition of $\mathcal{D}$ into K clusters.

Initialization Phase:
- Randomly select K data points from $\mathcal{D}$ as initial centroids $C^{(0)} = \{c_1^{(0)}, c_2^{(0)}, \ldots, c_K^{(0)}\}$.

Iterative Process:

repeat

  Cluster Assignment:

  for each data point $x_i \in \mathcal{D}$ **do**

    Assign x_i to the nearest centroid:

$$\text{Cluster}(x_i) = \arg\min_j \|x_i - c_j^{(t)}\|^2$$

  Centroid Update:

  for each cluster $j = 1, 2, \ldots, K$ **do**

    Compute the new centroid $c_j^{(t+1)}$ as the mean of all points assigned to cluster j:

$$c_j^{(t+1)} = \frac{1}{|\mathcal{S}_j|} \sum_{x_i \in \mathcal{S}_j} x_i$$

    Where $\mathcal{S}_j$ is the set of points assigned to cluster j.

  Check for Convergence:

  if $\|c_j^{(t+1)} - c_j^{(t)}\| < \epsilon$ for all $j = 1, 2, \ldots, K$ **then**

    Convergence reached. Exit loop.

until convergence

$$K_1 = \left(\frac{500+700}{2}, \frac{5+8}{2}\right) = \left(\frac{1200}{2}, \frac{13}{2}\right) = (600, 6.5)$$

Repeat for D and E

customer D:

$$k_1 = \sqrt{(1500-600)^2 + (20-6.5)^2} = \sqrt{810182.25} \approx 1100$$

$$k_2 = \sqrt{(1500-2000)^2 + (20-25)^2} = \sqrt{250025} \approx 500$$

Here k1>k2 so D will be joined to cluster k2.

$$K_1 = \{A, B\}, \quad K_2 = \{C, D\}$$

$$\text{New Centroid of } K_2 = \left(\frac{1500+2000}{2}, \frac{25+20}{2}\right) = (1750, 22.5)$$

customer E:

$$k_1 = \sqrt{(400-600)^2 + (20-6.5)^2} = \sqrt{40182.25} \approx 200.46$$

$$k_2 = \sqrt{(400-1750)^2 + (6-22.5)^2} = \sqrt{1822772.25} \approx 1350.10$$

Here k1<k2 so E will be joined to cluster k1.

$$K_1 = \{A, B, E\}, \quad K_2 = \{C, D\}$$

$$\text{New Centroid of } K_1 = \left(\frac{600+400}{2}, \frac{6.5+6}{2}\right) = (500, 6.25)$$

You'll get clusters $K1 = \{A, B, E\}$ and $K2 = \{C, D\}$.

Complexity Analysis. The time complexity of the K-Means clustering algorithm depends on the following factors:

- n: The number of data points in the dataset.
- K: The number of clusters.
- d: The dimensionality of the feature vectors.
- T: The number of iterations until convergence.

(1) **Initialization Phase:**
- Randomly selecting K initial centroids from n data points takes $O(K)$ time.

(2) **Cluster Assignment:**
- For each data point, compute the distance to all K centroids.
- Distance computation for one point takes $O(K \cdot d)$ time.
- For n data points, this step takes $O(n \cdot K \cdot d)$ time per iteration.

(3) **Centroid Update:**
- For each cluster, compute the mean of all assigned points.
- This involves summing d-dimensional vectors for all points in the cluster.
- For K clusters, this step takes $O(n \cdot d)$ time per iteration.

(4) **Convergence Check:**
- Compare the new centroids with the previous centroids.
- This step takes $O(K \cdot d)$ time per iteration.

Overall Time Complexity: The algorithm runs for T iterations, and the dominant step is the cluster assignment, which is repeated in each iteration. Therefore, the overall time complexity is:

$$O(T \cdot n \cdot K \cdot d)$$

Advantages:
- K-Means is easy to implement and understand, making it a good starting point for clustering tasks.
- The algorithm is computationally efficient, especially for large datasets, as it quickly converges to a solution.
- It can handle large datasets effectively, which is beneficial when dealing with big data.
- The algorithm usually requires only a few iterations to reach a stable solution, making it fast.

Disadvantages:
- The number of clusters (k) must be set before running the algorithm, which can be challenging if the optimal k is unknown [22].
- The outcome of the clustering can depend on the initial placement of centroids, which may lead to suboptimal results.
- K-Means works best when clusters are spherical and evenly sized, which makes it less effective for irregularly shaped clusters.
- The algorithm can be affected by outliers, as they can pull centroids away from the actual center of a cluster.
- K-Means relies on Euclidean distance for measuring similarity, which may not be suitable for all data types, particularly categorical data.

3.3.2 k-Medoids Clustering

Description: The K-Medoids algorithm is a clustering technique designed to improve upon K-Means, particularly in handling outliers [33]. Unlike K-Means, which relies on centroids (mean of data points), K-Medoids selects actual data points (medoids) as cluster representatives. This approach enhances robustness, ensuring clusters are less affected by extreme values or noise [33]. The algorithm minimizes the sum of dissimilarities between points labeled to be in a cluster and the medoid, making it more suitable for datasets with noise or outliers [34, 51]. K-Medoids is particularly effective in applications such as bioinformatics, image segmentation, and market segmentation, where data integrity and outlier resistance are critical [54].

Algorithm. The k-Medoids clustering process follows the iterative refinement approach described in Algorithm 5. The core steps are:

(1) **Initialization:** Select k data points randomly from the dataset $\mathcal{X}$ as initial medoids. Let the set of medoids be represented as: $\mathcal{M} = \{m_1, m_2, \ldots, m_k\}$ where m_j is the medoid of cluster j.
(2) **Cluster Assignment:** Assign each data point $x_i \in \mathcal{X}$ to the nearest medoid:

$$\text{Cluster}(x_i) = \arg\min_{m_j \in \mathcal{M}} d(x_i, m_j)$$

where:
- $d(x_i, m_j)$ is the distance between the data point x_i and the medoid m_j.
- The function arg min selects the medoid m_j that minimizes the distance to x_i.

(3) **Compute Total Cost:** The total cost C of the clustering is calculated as the sum of distances between each data point and its assigned medoid:

$$C = \sum_{i=1}^{n} \min_{m_j \in \mathcal{M}} d(x_i, m_j)$$

where:
- C represents the overall clustering cost.
- The summation runs over all data points x_i in the dataset.
- The minimum distance is taken for each x_i to its closest medoid.

(4) **Medoid Update:** Iterate through each medoid m_j and attempt to swap it with a non-medoid $x_h \in \mathcal{X} \setminus \mathcal{M}$. For each swap, compute the new total cost:

$$C_{\text{new}} = \sum_{i=1}^{n} \min_{m \in \mathcal{M}_{\text{new}}} d(x_i, m)$$

where:
- $\mathcal{M}_{\text{new}}$ represents the updated set of medoids after swapping m_j with x_h.
- C_{new} is the new total clustering cost.

If $C_{\text{new}} < C$, update the medoids $\mathcal{M}$ and set $C = C_{\text{new}}$.
(5) **Convergence Check:** If no medoid swaps lead to a decrease in cost C, the algorithm terminates.

Algorithm 5 K-Medoids Clustering [22]

Input: Dataset $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, Number of clusters k.
Output: k clusters with medoids.
Initialization:
Randomly select k data points from $\mathcal{X}$ as initial medoids $\mathcal{M} = \{m_1, m_2, \ldots, m_k\}$.
Repeat until convergence:
Cluster Assignment:
for each data point $x_i \in \mathcal{X}$ **do**
    Assign x_i to the nearest medoid:

$$\text{Cluster}(x_i) = \arg\min_{m_j \in \mathcal{M}} \|x_i - m_j\|$$

Compute Total Cost:
Calculate the total cost C as the sum of distances between each data point and its assigned medoid:

$$C = \sum_{i=1}^{n} \min_{m_j \in \mathcal{M}} \|x_i - m_j\|$$

Medoid Update:
for each medoid $m_j \in \mathcal{M}$ **do**
    for each non-medoid $x_h \in \mathcal{X} \setminus \mathcal{M}$ **do**
        Swap m_j with x_h to form a new set of medoids $\mathcal{M}_{\text{new}}$.
        Compute the new cost C_{new} for $\mathcal{M}_{\text{new}}$:

$$C_{\text{new}} = \sum_{i=1}^{n} \min_{m \in \mathcal{M}_{\text{new}}} \|x_i - m\|$$

        if $C_{\text{new}} < C$ **then**
            Update $\mathcal{M} = \mathcal{M}_{\text{new}}$.
            Update $C = C_{\text{new}}$.
Check for Convergence:
if No improvement in total cost C **then**
    Convergence reached. Exit loop.
Return final clusters and medoids.

Let's understand with an example, Calculate distance using

$$d_{\text{Manhattan}} = |X_1 - X_2| + |Y_1 - Y_2| \tag{24}$$

Customer	X	Y
A	8	2
B	3	5
C	4	7
D	8	4
E	5	5

Table 2: Coordinates of Customers

- Correct pair of medoids are (A,B), (A,C), (A,E), (D,B), (D,E).

Complexity Analysis: The K-Medoids algorithm consists of three main steps: **Cluster Assignment**, **Total Cost Computation**, and **Medoid Update**. The complexity of each step is analyzed as follows:

Customer	X	Y	K1	K2	Cost
A	8	2	0	8	0
B	3	5	8	0	0
C	4	7	9	3	3
D	8	4	2	6	2
E	5	5	6	2	2

Table 3: A and B medoids. Cost = 7. K1=A,D, K2=B,C,E

Customer	X	Y	K1	K2	Cost
A	8	2	0	9	0
B	3	5	8	3	3
C	4	7	9	0	0
D	8	4	2	7	2
E	5	5	6	3	3

Table 4: A and C as medoids. Cost = 7

Customer	X	Y	K1	K2	Cost
A	8	2	0	6	0
B	3	5	8	2	2
C	4	7	9	3	3
D	8	4	2	4	2
E	5	5	6	0	0

Table 5: A and E as medoids. Cost = 7

Customer	X	Y	K1	K2	Cost
A	8	2	8	2	2
B	3	5	0	6	0
C	4	7	3	7	3
D	8	4	6	0	0
E	5	5	2	4	2

Table 6: D and B as medoids. Cost = 7

(1) **Cluster Assignment:**
- For each data point $x_i \in \mathcal{X}$, the algorithm computes the distance to each of the k medoids to find the nearest one.
- The distance computation for a single data point is $O(k)$.
- Since there are n data points, the total complexity for this step is $O(nk)$.

(2) **Total Cost Computation:**
- After assigning all data points to clusters, the algorithm calculates the total cost C by summing the distances of each data point to its nearest medoid.
- This step involves iterating over all n data points and summing their distances, resulting in a complexity of $O(n)$.

(3) **Medoid Update:**
- For each of the k medoids, the algorithm considers swapping it with each of the $n - k$ non-medoid points.
- For each swap, the algorithm computes the new cost C_{new}, which involves recalculating the distances for all n data points to the new medoids.
- The complexity for a single swap is $O(nk)$, as each of the n data points must be compared to the k medoids.
- Since there are $k(n-k)$ possible swaps, the total complexity for this step is $O(k(n-k) \cdot nk) = O(k^2n(n-k))$.

Customer	X	Y	K1	K2	Cost
A	8	2	9	2	2
B	3	5	3	6	3
C	4	7	0	3	0
D	8	4	3	0	0
E	5	5	3	4	3

Table 7: D and C as medoids. Cost=8. Reject swap

Customer	X	Y	K1	K2	Cost
A	8	2	2	6	2
B	3	5	6	2	2
C	4	7	7	3	3
D	8	4	0	4	0
E	5	5	4	0	0

Table 8: D and E as medoids. cost = 7

Overall Complexity:
- The algorithm iterates until convergence, which typically requires t iterations.
- Combining the complexities of all steps, the overall complexity of the K-Medoids algorithm is:

$$O(t \cdot (nk + n + k^2n(n-k))) = O(tk^2n(n-k))$$

- In the worst case, where $k \ll n$, the complexity simplifies to $O(tk^2n^2)$.

The K-Medoids algorithm has a polynomial time complexity, making it computationally expensive for large datasets or a high number of clusters. However, it is more robust to outliers compared to K-Means due to its use of medoids instead of centroids.

Advantages:
- Unlike K-Means, which can be significantly affected by outliers, K-Medoids selects actual data points as cluster centers. This makes the algorithm more resistant to outliers or noisy data, leading to more reliable results in datasets with extreme values.
- K-Medoids allows the use of various distance metrics, not just Euclidean distance. This flexibility makes it suitable for a broader range of data types, including categorical and non-metric data, where Euclidean distance may not be applicable.
- Since K-Medoids chooses actual data points as centroids (medoids), the cluster centers are more interpretable and meaningful in the context of the data. This can be helpful for understanding the characteristics of the clusters.
- K-Medoids can perform better than K-Means in cases where the clusters are not spherical or equally sized. This makes it a good choice for data with more complex cluster shapes.

Disadvantages:
- K-Medoids generally requires more computational power compared to K-Means, especially for large datasets. This is because the algorithm must calculate distances between all data points to find the optimal medoids, which can be time-consuming.
- Similar to K-Means, K-Medoids requires the number of clusters (k) to be specified beforehand. Identifying the optimal

number of clusters can be challenging, particularly when there is no clear distinction between the groups in the data.
- The algorithm's time complexity makes it less scalable for very large datasets. As the number of data points increases, the performance of K-Medoids can degrade significantly.
- K-Medoids, while less sensitive than K-Means, can still be affected by the initial selection of medoids. Poor initialization may lead to suboptimal clustering results.

3.4 Exemplar-Based Clustering

Exemplar-based clustering algorithms identify specific data points, known as exemplars, that serve as representative centers for clusters [37]. Unlike traditional methods that compute abstract centroids (e.g., k-means), these algorithms select actual data points as cluster centers, enhancing interpretability and robustness [13].

Key Characteristics:

- **Selection of Exemplars:** The algorithm evaluates all data points to determine which ones are most representative of the underlying data distribution.
- **Cluster Formation:** Once exemplars are chosen, each remaining data point is assigned to the cluster of the nearest exemplar, effectively grouping similar points together.

Advantages:

- **Interpretability:** Using actual data points as cluster centers makes the results more understandable, as each exemplar is a concrete example from the dataset.
- **Automatic Determination of Cluster Number:** Many exemplar-based methods, such as Affinity Propagation, do not require pre-specifying the number of clusters; the algorithm determines this based on the data.

Challenges:

- **Computational Complexity:** Evaluating all possible pairs of data points can be computationally intensive, especially for large datasets.
- **Parameter Sensitivity:** The performance of these algorithms can be sensitive to parameters like similarity measures and preference values, necessitating careful tuning.

Overall, exemplar-based clustering provides a powerful approach for identifying representative data points, facilitating more interpretable and potentially more accurate clustering outcomes.

3.4.1 Affinity Propagation Clustering Algorithm

Affinity propagation is a clustering algorithm in which every data point is considered a potential exemplar (i.e., cluster center) [16]. It uses a similarity matrix S of size $n \times n$, where each entry $S(i,k)$ reflects how well point k might serve as the exemplar for point i. In many applications, one sets [41]:

$$S(i,k) = -\|x_i - x_k\|^2 \tag{25}$$

and places 'preference' values on diagonal $S(k,k)$ to control how likely k is to become a cluster center [38].

In affinity propagation, two types of messages are exchanged: the **responsibility** $r(i,k)$, which captures how strongly i might favor k over other exemplars, and the **availability** $a(i,k)$, which indicates how appropriate k is for i given the competition among other data points. The responsibility update is defined as [80]:

$$r(i,k) = s(i,k) \; - \; \max_{k' \neq k} \big[a(i,k') + s(i,k') \big] \tag{26}$$

Equation (26) highlights that $r(i,k)$ becomes larger if $s(i,k)$ is high and no other potential exemplar k' provides a bigger sum. Next, for the availability update when $i \neq k$:

$$a(i,k) = \min \Big\{ 0, \; r(k,k) + \sum_{i' \notin \{i,k\}} \max\big(0, \; r(i',k)\big) \Big\} \tag{27}$$

And for the self-availability when $i = k$:

$$a(k,k) = \sum_{i' \neq k} \max\big(0, \; r(i',k)\big) \tag{28}$$

Equations (27) and (28) effectively ensure that if a point k has strong self-responsibility $r(k,k)$, and/or many positive responsibilities from other points, then $a(i,k)$ can become positive, encouraging k to serve as an exemplar [78]. A damping factor $\lambda \in [0,1]$ is usually applied after each new calculation of r or a, blending old and new values to reduce oscillations.

Eventually, once the messages stabilize or a maximum iteration count is reached, each point i is assigned to whichever k maximizes the sum $\big[a(i,k) + r(i,k)\big]$:

$$\text{Exemplar}(i) = \arg\max_{k} \big[a(i,k) + r(i,k) \big] \tag{29}$$

Points k for which $\text{Exemplar}(k) = k$ become true exemplars. This final assignment step completes the clustering: each point is linked to exactly one exemplar, and the resulting sets define the clusters [77].

Similarity Matrix:

Suppose we have the set of points:

$$X = \{ (1,2), (2,2), (3,2), (8,8), (9,8), (10,8), (1,10), (2,10), (3,10), (5,5) \}.$$

From Equation (25), we compute:

$$s(i,k) \; = \; -\|x_i - x_k\|^2 \quad \text{for } i \neq k,$$

and on the diagonal $S(k,k)$, we place our preference p, which we set to the median of all off-diagonal values (here, -53). Hence, the resulting 10×10 matrix S is:

i/k	1	2	3	4	5	6	7	8	9	10
1	-53	-1	-4	-85	-100	-117	-64	-65	-68	-25
2	-1	-53	-1	-72	-85	-100	-65	-64	-65	-18
3	-4	-1	-53	-61	-72	-85	-68	-65	-64	-13
4	-85	-72	-61	-53	-1	-4	-53	-40	-29	-18
5	-100	-85	-72	-1	-53	-1	-68	-53	-40	-25
6	-117	-100	-85	-4	-1	-53	-85	-68	-53	-34
7	-64	-65	-68	-53	-68	-85	-53	-1	-4	-41
8	-65	-64	-65	-40	-53	-68	-1	-53	-1	-34
9	-68	-65	-64	-29	-40	-53	-4	-1	-53	-29
10	-25	-18	-13	-18	-25	-34	-41	-34	-29	-53

Table 9: Example similarity matrix (diagonal = median preference, -53).

Time Complexity: Each iteration updates responsibilities and availabilities for every pair (i,k), giving $O(n^2)$ per iteration. If we run for T iterations, the time complexity is $O(n^2 T)$.

Algorithm 6 Affinity Propagation

Require: Similarity matrix $S \in \mathbb{R}^{n \times n}$, maximum iterations max_iter, damping factor λ, stability iterations stability_iter
Ensure: Clusters $\{C_1, C_2, \ldots, C_k\}$
Initialize $R \leftarrow 0_{(n,n)}$, $A \leftarrow 0_{(n,n)}$, stable_count $\leftarrow 0$
exemplars_prev $\leftarrow \emptyset$
for iteration = 1 to max_iter **do**
  Update Responsibilities:
  for each $i, k \in \{1, 2, \ldots, n\}$ **do**
    $R_{\text{new}}[i,k] \leftarrow (1 - \lambda)\left(S[i,k] - \max_{k' \neq k}(A[i,k'] + S[i,k'])\right) + \lambda R[i,k]$
  Update Availabilities:
  for each $i, k \in \{1, 2, \ldots, n\}$ **do**
    if $i \neq k$ **then**
      $A_{\text{new}}[i,k] \leftarrow (1 - \lambda) \min\left(0, R[k,k] + \sum_{i' \notin \{i,k\}} \max(0, R[i',k])\right) + \lambda A[i,k]$
    else
      $A_{\text{new}}[k,k] \leftarrow (1 - \lambda) \sum_{i' \neq k} \max(0, R[i',k]) + \lambda A[k,k]$
  $R \leftarrow R_{\text{new}}$, $A \leftarrow A_{\text{new}}$
  Check Convergence:
  current_exemplars $\leftarrow \arg\max_k(A[i,k] + R[i,k])$ for each i
  if current_exemplars = exemplars_prev **then**
    stable_count $\leftarrow$ stable_count + 1
  else
    stable_count $\leftarrow 0$
  exemplars_prev $\leftarrow$ current_exemplars
  if stable_count = stability_iter **then**
    break
Assign Clusters:
clusters $\leftarrow \{\}$
for each $k \in$ current_exemplars **do**
  clusters$[k] \leftarrow \{k\}$
for each $i \notin$ current_exemplars **do**
  best$_k \leftarrow \arg\max_k(A[i,k] + R[i,k])$
  clusters[best$_k$].append(i)
  return clusters

Space Complexity: We store the $n \times n$ similarity matrix S, and the responsibility and availability matrices are also $n \times n$. Thus, the space complexity is $O(n^2)$.

3.4.2 Mean-Shift Clustering Algorithm

Mean shift is a nonparametric clustering method that locates cluster centers by seeking high-density regions in the feature space. It does not require specifying the number of clusters in advance; instead, clusters emerge as data points congregate around modes (local density maxima) of the estimated probability density [3, 7]. At a high level, each data point is iteratively shifted towards areas of higher data density (a process of *mode seeking*) until convergence [84]. This approach can discover clusters of arbitrary shape since it finds clusters by following the underlying data distribution rather than imposing a fixed shape [3].

Algorithm. Algorithm 7 describes the Mean-shift algorithm. Given a set of data points, $\{x_i\}_{i=1}^N \subset \mathbb{R}^d$ and a window radius (bandwidth) r, mean shift starts by placing a window of radius r around an initial point. At each iteration, the mean shift update moves the center of this window to the centroid of the data points inside it. In other words, if

$$S_y = \left\{ x_i \mid \|x_i - y\| \leq r \right\} \tag{30}$$

is the neighborhood of the current center y, then y is updated as:

$$y_{\text{new}} = \frac{1}{|S_y|} \sum_{x_i \in S_y} x_i \tag{31}$$

(i.e., shifted to the average of the neighbors). The displacement,

$$\Delta y = \frac{1}{|S_y|} \sum_{x_i \in S_y} x_i \; - \; y \tag{32}$$

is called the *mean shift vector* [11], which points in the direction of maximum increase in the local density (proportional to the density gradient at y). By iteratively recalculating the neighborhood and shifting y, each point climbs the density gradient and eventually converges to a local maximum of density (a cluster mode) [7, 84]. All points that converge to the same mode are assigned to the same cluster, so the final number of clusters is determined by the number of distinct modes found [3].

Algorithm 7 Mean Shift Clustering

Require: Points $\{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^d$, bandwidth r, threshold ε
Ensure: Cluster centers $\{m_1, m_2, \ldots, m_k\}$, cluster assignments $\{c_1, c_2, \ldots, c_n\}$
Initialize modes $\{m_1, m_2, \ldots, m_n\} \leftarrow \{x_1, x_2, \ldots, x_n\}$
for each point $x_i \in \{x_1, x_2, \ldots, x_n\}$ **do**
  $y \leftarrow x_i$ ▷ Initialize mean at x_i
  repeat
    $S_y \leftarrow \{x_j \mid \|x_j - y\| \leq r\}$ ▷ Collect neighbors within radius r (Eq. 30)
    $y_{\text{new}} \leftarrow \frac{1}{|S_y|} \sum_{x_j \in S_y} x_j$ ▷ Compute new mean (Eq. 31)
    $\Delta y \leftarrow y_{\text{new}} - y$ ▷ Compute mean shift vector (Eq. 32)
    $y \leftarrow y_{\text{new}}$ ▷ Update mean
  until $\|\Delta y\| < \varepsilon$ ▷ Convergence check
  $m_i \leftarrow y$ ▷ Assign final mode for x_i
Merge modes $\{m_1, m_2, \ldots, m_n\}$ into distinct clusters $\{m_1, m_2, \ldots, m_k\}$ ▷ Merge close modes
Assign each x_i to the cluster of its mode m_i ▷ Cluster assignment
  return Cluster centers $\{m_1, m_2, \ldots, m_k\}$, cluster assignments $\{c_1, c_2, \ldots, c_n\}$

Time Complexity: The worst-case time complexity for the Mean Shift algorithm is $O(n^2 T)$, where n is the number of data points and T is the number of steps required for convergence [72]. The

quadratic complexity arises from the need to recompute distances or kernel weights between each pair of points at every step.
Space Complexity: Mean Shift does not require much additional memory beyond storing the dataset and the result. The space complexity is $O(n\,d)$, where n is the number of points and d is the dimensionality of each point.

3.5 Hierarchical-Based Clustering

Hierarchical clustering is an unsupervised machine learning technique that systematically organizes data into a multilevel hierarchy of clusters. This method is particularly useful for discovering hidden patterns or groupings in data without prior knowledge of the number of clusters.

Types of Hierarchical Clustering:

(1) **Agglomerative (Bottom-Up) Approach:**
- **Process:** Starts with each data point as an individual cluster and iteratively merges the closest pairs of clusters until all points are grouped into a single cluster or a specified number of clusters is reached.
- **Characteristics:** This approach is more common in practice and is known for its straightforward implementation.

(2) **Divisive (Top-Down) Approach:**
- **Process:** Begins with the entire dataset as one cluster and recursively splits it into smaller clusters until each data point is in its own cluster or the desired clustering structure is achieved.
- **Characteristics:** This method is less commonly used due to its computational complexity but can be effective in certain scenarios.

Key Components:

- **Distance Metrics:** To determine the similarity between data points or clusters, various distance measures can be used, such as Euclidean distance, Manhattan distance, or cosine similarity. The choice of metric influences the shape and cohesion of the clusters.
- **Linkage Criteria:** These criteria define how the distance between clusters is calculated during the merging or splitting process. Common linkage methods include:
  - **Single Linkage:** Distance between the closest points of two clusters.
  - **Complete Linkage:** Distance between the farthest points of two clusters.
  - **Average Linkage:** Average distance between all pairs of points in two clusters.
  - **Ward's Method:** Minimizes the variance within clusters by considering the sum of squared differences.

Output Representation:

The result of hierarchical clustering is typically visualized using a dendrogram—a tree-like diagram that illustrates the sequence of merges or splits and the relative distances between clusters. The dendrogram provides insights into the data's structure, allowing users to select a level at which to cut the tree to form a desired number of clusters.

Advantages:

- **No Need to Predefine Number of Clusters:** The hierarchical nature allows exploration of the data at various levels of granularity without specifying the number of clusters upfront.
- **Versatility:** Applicable to a wide range of data types and capable of capturing complex cluster shapes.

Limitations:

- **Computational Complexity:** Especially for large datasets, hierarchical clustering can be computationally intensive in both time and memory.
- **Sensitivity to Noise and Outliers:** The presence of noisy data or outliers can significantly affect the resulting cluster structure.

3.5.1 BIRCH Hierarchical Clustering

Description. Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is a hierarchical clustering algorithm designed to efficiently handle large datasets by incrementally and dynamically clustering incoming data points [81]. It builds a hierarchical data structure known as the Clustering Feature (CF) tree to summarize the dataset, allowing for effective clustering with a single scan of the data and additional scans if refinement is necessary.

Algorithm. The BIRCH algorithm is designed to efficiently cluster large datasets by incrementally building a hierarchical data structure called the Clustering Feature (CF) tree. It is described in Algorithm 8, it operates in two main phases:

(1) **CF Tree Construction**: A compact representation of the dataset is built by inserting data points into a CF tree, which maintains summary statistics of clusters.
(2) **Clustering of Leaf Nodes**: The leaf entries of the CF tree are further processed using hierarchical clustering to generate the final clusters.

Phase 1: Building the CF Tree.

- Initialize an empty CF tree with a predefined threshold T (maximum radius or diameter of a cluster) and branching factor B (maximum number of child nodes per internal node).
- For each incoming data point:
  - Traverse the CF tree to find the closest leaf node that can accommodate the data point.
  - If the leaf node can absorb the new data point without exceeding T, update its clustering feature (CF) statistics.
  - Otherwise, split the leaf node into two and adjust the CF tree structure accordingly.
- This process continues until all data points are inserted into the CF tree.

Phase 2: Clustering the Leaf Nodes.

- Extract the leaf entries from the CF tree, each representing a micro-cluster.
- Apply an agglomerative hierarchical clustering algorithm to group the micro-clusters into the final set of clusters.
- Compute the final cluster centroids based on the aggregated clustering features of the merged groups.

This two-phase approach ensures that large datasets can be efficiently processed by summarizing the data in the CF tree before applying hierarchical clustering [81].

Algorithm 8 Clustering Feature (CF) Tree Construction and Clustering

Require: Threshold T, branching factor B, dataset $D = \{x_1, x_2, \ldots, x_n\}$
Ensure: CF tree, cluster centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$

Phase 1: Building the CF Tree
Initialize an empty CF tree with threshold T and branching factor B
for each data point $x_i \in D$ **do**
    Traverse CF tree to find the closest leaf node L for x_i
    if L can absorb x_i without violating T **then**
        Update L's clustering feature (CF)
    else
        Split L into two new leaves
        Adjust the CF tree structure to accommodate the split
Phase 2: Clustering the Leaves
Extract all leaf entries $\{L_1, L_2, \ldots, L_m\}$ from the CF tree
Apply agglomerative clustering to $\{L_1, L_2, \ldots, L_m\}$ to form final clusters
Compute cluster centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$ from the final clusters
return CF tree, cluster centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$

Complexity Analysis:

1. **Phase 1: CF Tree Construction**
   - **Insertion Operations** Each data point is inserted into the CF tree through a traversal process. Given that the CF tree is typically balanced and shallow, the insertion operation for each point is efficient. Consequently, the overall time complexity for inserting n data points is approximately $O(n)$.
2. **Phase 2: Global Clustering**
   - **Sub-cluster Clustering** After constructing the CF tree, the leaf nodes (sub-clusters) are clustered using a global clustering algorithm. The time complexity of this phase depends on the specific algorithm chosen and the number of sub-clusters. For instance, if agglomerative clustering is used, the complexity is typically $O(m^2 \log m)$, where m is the number of sub-clusters. However, since m is generally much smaller than n, this phase is computationally manageable.

- **Overall Time Complexity :** Considering both phases, the BIRCH algorithm operates with a time complexity of $O(n)$ for the CF tree construction and an additional $O(m^2 \log m)$ for the global clustering phase. Given that m is significantly smaller than n, the overall complexity is effectively linear with respect to the number of data points, making BIRCH highly efficient for large datasets.
- **Space Complexity:** The space complexity depends on the size of the CF tree, which is influenced by the threshold and branching factor. Typically, the CF tree is much smaller than the original dataset, making BIRCH space-efficient.

Advantages:

- **Scalability:** BIRCH is designed to handle large datasets efficiently by building a compact CF tree that summarizes the data.
- **Incremental Clustering:** It can incorporate new data points without the need to reprocess the entire dataset, making it suitable for dynamic datasets.
- **Versatility:** The CF tree can serve as a preprocessing step for other clustering algorithms, enhancing their performance on large datasets.

Limitations:

- **Sensitivity to Order of Data:** The clustering result can vary depending on the order in which data points are processed.
- **Threshold Selection:** Choosing an appropriate threshold is crucial; too small a threshold may lead to excessive splitting, while too large a threshold may result in poor clustering quality.

BIRCH's ability to efficiently cluster large datasets with a single scan makes it a valuable tool in data mining and exploratory data analysis.

3.5.2 Agglomerative Hierarchical Clustering

Description: Agglomerative Hierarchical Clustering (AHC) is a bottom-up clustering method that builds a hierarchy of clusters by progressively merging smaller clusters into larger ones based on their similarity [9]. This approach is widely used in data analysis to uncover the underlying structure of data without requiring a predefined number of clusters.

Algorithm. Algorithm 9 describes the AHC algorithm. It comprises of three phases:

Phase 1: Initialization.

- Treat each data point as an individual cluster.
- Compute a pairwise distance matrix between all data points based on a chosen similarity metric (e.g., Euclidean distance).

Phase 2: Iterative Merging.

- Identify the two closest clusters based on a predefined linkage criterion (e.g., single linkage, complete linkage, average linkage).
- Merge the selected clusters into a new cluster.
- Update the distance matrix to reflect the new cluster's similarity with the remaining clusters.
- Repeat the process until the desired number of clusters is reached or all points are merged into a single cluster.

Phase 3: Dendrogram Construction and Cluster Assignment.

- Record the merging sequence to construct a dendrogram, representing the hierarchical relationships between clusters.
- Depending on the application, select an appropriate cut-off level in the dendrogram to determine the final cluster assignments.

This hierarchical approach provides a flexible and interpretable clustering structure, making it useful for exploratory data analysis and domain-specific applications [9].

Algorithm 9 Hierarchical Agglomerative Clustering (HAC)

Require: Dataset $D = \{x_1, x_2, \ldots, x_n\}$, number of clusters k, linkage criterion
Ensure: Dendrogram, cluster labels $C = \{c_1, c_2, \ldots, c_n\}$
Initialize $C \leftarrow \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\}$ ▷ Each point as a cluster
Initialize distance matrix Dist $\leftarrow$ pairwise distances between clusters
while $|C| > k$ **do**
  Find $(c_a, c_b) \leftarrow_{c_i, c_j \in C} \text{Dist}(c_i, c_j)$ ▷ Closest clusters
  Merge c_a and c_b into new cluster c_{new}
  Update $C \leftarrow C \setminus \{c_a, c_b\} \cup \{c_{\text{new}}\}$
  Update Dist for c_{new} using linkage criterion
Generate dendrogram from merge history
Assign cluster labels C to each $x_i \in D$
return Dendrogram, cluster labels C

Complexity Analysis.

1. **Initialization**
   - **Distance Matrix Computation** Calculating the pairwise distances between all n data points requires $O(n^2)$ time.
2. **Iterative Merging**
   - **Identifying Closest Clusters** At each iteration, finding the pair of clusters with the smallest distance can be done in $O(1)$ time if a priority queue (min-heap) is used.
   - **Merging Clusters** Merging two clusters is an $O(1)$ operation.
   - **Updating Distances** Updating the distance matrix to reflect the distances between the new cluster and the remaining clusters depends on the linkage criterion:
     - **Single Linkage**: Updating distances can be done in $O(n)$ time per iteration.
     - **Complete and Average Linkage**: These may require $O(n)$ time per iteration.
     - **Ward's Method**: This often requires more complex computations, potentially increasing the time per iteration.

- **Overall Time Complexity:** Since there are $n - 1$ merging steps, the overall time complexity is typically $O(n^2 \log n)$, especially when efficient data structures like priority queues are used. However, for certain linkage criteria and naive implementations, the complexity can be $O(n^3)$ item **Optimizations:** SLINK Algorithm: For single-linkage clustering, the SLINK algorithm reduces the time complexity to $O(n^2)$ [47]. Approximations and Constraints: Constraining merges to neighboring nodes on a graph or using approximate methods can lead to faster computations. For instance, using a k-nearest neighbor graph can achieve $O(n \log n)$ time complexity [63].
- **Space Complexity:** The algorithm requires $O(n^2)$ space to store the distance matrix, which can be a limitation for large datasets.

Advantages.

- **Flexibility:** AHC does not require a predefined number of clusters and can adapt to the data's inherent structure.
- **Deterministic:** The results are reproducible and do not depend on initializations, unlike some other clustering methods.

Limitations.

- **Computationally Intensive:** The time and space complexities make AHC less suitable for very large datasets.
- **Sensitivity to Noise:** Outliers can significantly affect the merging process, leading to less meaningful clusters.

Agglomerative Hierarchical Clustering is a powerful tool for exploratory data analysis, especially when the underlying number of clusters is unknown. Its hierarchical nature provides a comprehensive view of the data's structure, which can be invaluable in various applications.

4 Metrics

The evaluation of cluster quality is crucial to assess the performance of the algorithm. This section describes key clustering evaluation metrics, including their mathematical definitions, interpretations, and computational complexities. In this section, we present various clustering evaluation metrics, highlighting their definitions, interpretations, and computational complexities. The choice of metric depends on the clustering method and the characteristics of the data set.

4.1 Silhouette Score

Definition: The Silhouette Score measures how well each point fits into its assigned cluster compared to the nearest other cluster [61]. For each point i, the silhouette score is given by:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{33}$$

where:

- $a(i)$ is the average intra-cluster distance (distance from i to all other points in the same cluster).
- $b(i)$ is the lowest average inter-cluster distance (distance from i to points in the nearest other cluster).

The overall silhouette score is the average of $s(i)$ over all points.
Interpretation:

- $s(i) \approx 1$ indicates that i is well-clustered.
- $s(i) \approx 0$ indicates that i lies on the decision boundary between clusters.
- $s(i) < 0$ suggests that i is likely misclassified.

Complexity: Computing $a(i)$ and $b(i)$ requires pairwise distances, making the complexity:

- $O(n^2)$ for all-pairs computation.
- $O(n)$ with efficient nearest-neighbor search methods.

4.2 Davies-Bouldin Index

Definition: The Davies-Bouldin Index (DBI) measures cluster compactness and separation. It is defined as:

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right) \tag{34}$$

where:

- k is the number of clusters.
- σ_i is the average distance of points in cluster i to its centroid c_i.
- $d(c_i, c_j)$ is the distance between centroids of clusters i and j.

Interpretation: Lower DBI values indicate better clustering.

Complexity: Computing intra-cluster dispersion and inter-cluster distances yields:

- $O(n)$ for intra-cluster distances.
- $O(k^2)$ for inter-cluster comparisons.
- Overall complexity: $O(n + k^2)$.

4.3 Calinski-Harabasz Index (Variance Ratio Criterion)

Definition: Also known as the Variance Ratio Criterion, this index is given by [73]:

$$CH = \frac{\text{trace}(B_k)}{\text{trace}(W_k)} \times \frac{n-k}{k-1} \tag{35}$$

where:

- B_k is the between-cluster dispersion matrix.
- W_k is the within-cluster dispersion matrix.
- n is the number of data points.
- k is the number of clusters.

Interpretation: Higher values indicate better clustering quality [73].

Complexity:

- $O(n)$ for centroid computation.
- $O(kd)$ for variance calculations.
- Overall complexity: $O(n)$.

4.4 Adjusted Rand Index (ARI)

Definition: ARI compares the agreement between clustering assignments and ground truth labels [75, 76]. It is given by:

$$ARI = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2} \right] / \binom{n}{2}}{\frac{1}{2} \left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2} \right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2} \right] / \binom{n}{2}} \tag{36}$$

where:

- n_{ij} is the number of points in both cluster i and ground truth class j.
- a_i is the number of points in cluster i.
- b_j is the number of points in ground truth class j.
- n is the total number of points.

Interpretation:

- $ARI = 1$ indicates perfect clustering.
- $ARI = 0$ corresponds to random clustering.

Complexity: Computation involves contingency tables:

- $O(n + k^2)$ using efficient contingency table calculations.

4.5 Mutual Information (MI)

Definition: MI measures how much knowing the clustering labels tells us about ground truth classes:

$$MI(X, Y) = \sum_{i=1}^{k} \sum_{j=1}^{m} P(i, j) \log \frac{P(i, j)}{P(i)P(j)} \tag{37}$$

where:

- $P(i)$ and $P(j)$ are marginal probabilities of clusters and classes.
- $P(i, j)$ is the joint probability of a data point being in both cluster i and class j.

Interpretation: Higher values indicate better agreement with ground truth.

Complexity:

- $O(n + k^2)$ using contingency tables.

4.6 Split

Definition: Cluster *split* is the minimum inter-cluster distance:

$$\text{Split} = \min_{i \neq j} d(c_i, c_j) \tag{38}$$

where $d(c_i, c_j)$ is the distance between centroids of clusters i and j.

Interpretation: Higher values indicate well-separated clusters.

Complexity:

- $O(k^2)$ for pairwise centroid distance calculations.

4.7 Diameter

Definition: Cluster *diameter* is the largest intra-cluster distance:

$$\text{Diameter} = \max_{x,y \in C} d(x, y) \tag{39}$$

where $d(x, y)$ is the Euclidean distance between points x and y in the same cluster.

Interpretation: Smaller diameters indicate compact clusters.

Complexity:

- $O(n^2)$ for exact pairwise distance computations.
- $O(n)$ with approximate methods.

5 K-Means Accuracy

As discussed in Section 3.3, K-means clustering aims to partition data into k clusters by minimizing intra-cluster variance. However, assessing the *accuracy* of k-means is challenging since clustering is an unsupervised learning task. While several references suggest using the **Silhouette Score** for evaluation [29, 30, 48], this metric does not directly assess accuracy.

5.1 Why the Silhouette Score Does Not Assess Accuracy

The **Silhouette Score** measures the compactness and separation of clusters but does not compare them to any ground truth labels [61]. It is defined for each point i as:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{40}$$

where:

- $a(i)$ is the average intra-cluster distance (within-cluster cohesion).
- $b(i)$ is the lowest average inter-cluster distance (nearest-cluster separation).

While a high silhouette score suggests well-separated and compact clusters [29], it does not indicate whether the clustering matches any *ground truth labels*. Thus, an exact k-means clustering (i.e., the one that optimally minimizes intra-cluster variance) may still yield a poor silhouette score if the natural data structure does not conform to k-means assumptions.

5.2 Exact K-Means Clustering with a Bad Silhouette Score

Even when k-means clustering finds an optimal solution by minimizing the variance within clusters, it can still produce a low silhouette score [46, 76]. This happens because the silhouette score measures how well-separated clusters are, and it may not align with the results of k-means in certain situations.

One reason for this mismatch is the shape of the clusters. K-means works best when clusters are round and evenly sized. However, if the clusters are long, curved, or irregularly shaped, k-means may group points in a way that minimizes variance but does not reflect the true structure of the data. For example, in a dataset with two crescent-shaped clusters (like the "two moons" dataset), k-means might split the crescents into parts that are not naturally separated. This can cause many points to be equally close to both clusters, resulting in a low silhouette score.

Another reason is the closeness or overlap of clusters. If clusters are near each other or overlap, the distances between points in different clusters can be very similar to the distances within clusters. This makes it hard for the silhouette score to distinguish between well-separated and poorly separated clusters. Additionally, if clusters have high internal variability, the silhouette score may decrease because the differences between intra-cluster and inter-cluster distances become less clear.

In summary, a low silhouette score does not always mean that k-means has failed. It can simply indicate that the data does not fit the assumptions of k-means or the silhouette score. This highlights the importance of choosing the right clustering method and evaluation metric for the type of data being analyzed.

5.3 Example: 10 Points with Exact K-Means and a Bad Silhouette Score

Consider the following dataset:

$$X = \{(1,2), (2,2), (3,2), (8,8), (9,8), (10,8), (1,10), (2,10), (3,10), (5,5)\}$$

If we set $k = 3$, k-means may produce the following clusters:

- Cluster 1: $\{(1,2), (2,2), (3,2)\}$
- Cluster 2: $\{(8,8), (9,8), (10,8)\}$
- Cluster 3: $\{(1,10), (2,10), (3,10), (5,5)\}$

- The centroid of Cluster 3 will be skewed due to the outlier $(5,5)$. - Points near the boundary (e.g., $(5,5)$) may have similar distances to multiple centroids, leading to a low silhouette score. - Even though k-means has minimized intra-cluster variance correctly, the structure of the dataset causes a misleading silhouette interpretation.

This example illustrates that silhouette score alone is not a sufficient measure of accuracy in k-means clustering, especially when clusters are not well-separated or when they have varying densities.

6 K-Means Improvement

Although k-means clustering is widely used due to its simplicity and efficiency, several optimization techniques have been proposed to further speed up the algorithm. One such method, introduced by Elkan [18], leverages the **triangle inequality** to reduce the number of distance calculations during the iterative updates.

6.1 Elkan's Triangle Inequality Optimization

Standard k-means: As shown in Algorithm 4 of Section 3.1, at each iteration, for each data point x_i, k-means computes the distance to all k centroids to determine the closest one. This results in a complexity of $O(nkd)$ per iteration, where n is the number of points, k is the number of clusters, and d is the number of dimensions.

Elkan's Optimization: Instead of computing all distances explicitly, Elkan's method uses the triangle inequality to *prune unnecessary distance calculations* [18]. The core idea is:

$$d(x, C_a) \geq d(C_a, C_b) - d(x, C_b) \tag{41}$$

where $d(x, C_a)$ is the distance from point x to centroid C_a, and $d(C_a, C_b)$ is the distance between centroids C_a and C_b. If:

$$d(x, C_b) \leq \frac{1}{2} d(C_a, C_b), \tag{42}$$

then x cannot be closer to C_a than to C_b, eliminating the need to compute $d(x, C_a)$.

For example, consider the following dataset with three points and two centroids:

$$X = \{(2,2), (4,4), (8,8)\}, \quad C_1 = (3,3), \quad C_2 = (7,7)$$

Step 1: Compute centroid distances

$$d(C_1, C_2) = \sqrt{(7-3)^2 + (7-3)^2} = \sqrt{16+16} = 4\sqrt{2}$$

Step 2: Apply the triangle inequality for pruning

- For $x = (2,2)$, assume we have already computed $d(x, C_1) = \sqrt{2}$. - We check if $d(x, C_1) \leq \frac{1}{2} d(C_1, C_2)$:

$$\sqrt{2} \leq \frac{1}{2}(4\sqrt{2}) = 2\sqrt{2}$$

Since the condition holds, we do not need to compute $d(x, C_2)$ explicitly.

This method significantly reduces the number of distance computations.

6.2 Complexity of Elkan's Algorithm

Standard k-means as seen in Algorithm 4, has a per-iteration complexity of:

$$O(nkd) \tag{43}$$

Elkan's algorithm reduces the number of distance calculations by maintaining:

- **Upper bounds** $u(x)$ on distances from each point to its assigned centroid.
- **Lower bounds** $l(x, C)$ on distances from the point to all other centroids.

In each iteration, these bounds are updated, avoiding redundant computations. The worst-case complexity remains $O(nkd)$, but in practice, the pruning effect significantly reduces the number of calculations, often bringing it closer to:

$$O(nk'd), \tag{44}$$

where $k' \ll k$ is the average number of centroids each point needs to check.

6.3 Impact of Triangle Inequality Tests on Complexity

- In the early iterations, most distances still need to be computed, so the performance gain is minimal.
- As centroids stabilize, the number of required distance computations drops dramatically, leading to significant speedups.
- For well-separated clusters, each point only needs to check a few centroids, making the effective complexity close to $O(nd)$ in later iterations.

Empirical results show that Elkan's method reduces distance computations by **70-95%** in real-world datasets [18], making it the fastest exact k-means variant in most practical cases.

Elkan's optimization dramatically improves k-means performance by leveraging the triangle inequality [18, 23]. While its theoretical worst-case complexity remains the same as standard k-means, its practical runtime is significantly reduced due to the pruning of unnecessary computations.

7 Metrics Appropriateness

Each clustering algorithm optimizes a different objective and, as a result, is best evaluated by certain metrics that align with its underlying assumptions. However, no single metric is universally suitable, and alternative metrics can highlight weaknesses in the clustering performance.

7.1 Metric Selection by Clustering Algorithm

Table 10 summarizes the most appropriate metric for evaluating each clustering algorithm based on its optimization objective, along with a complementary metric that addresses potential weaknesses. Some of the key findings include: k-Means is best evaluated using the Silhouette Score, with ARI as an alternative; k-Medoids favors the Davies-Bouldin Index; GMM is best assessed with BIC; DBSCAN and OPTICS rely on ARI and reachability plots, respectively; Affinity Propagation and Mean-Shift both benefit from the Silhouette Score; BIRCH is best evaluated using the Calinski-Harabasz Index; and Agglomerative Hierarchical Clustering is best measured with the Cophenetic Correlation.

7.2 Justification for Metric Choices

- **k-Means and k-Medoids:** Since these methods partition data based on centroids or medoids, compactness and separation (Silhouette Score, DBI) are the best evaluation criteria. When true labels exist, ARI helps assess clustering correctness [75].
- **GMM:** As a probabilistic model, GMM is best evaluated using likelihood-based metrics such as BIC. External metrics like AMI can be used for supervised validation.
- **DBSCAN and OPTICS:** These density-based methods handle arbitrary shapes and noise. ARI is useful when labels are available, while silhouette score can assess separation. However, reachability plots provide deeper insights into OPTICS clustering.
- **Affinity Propagation and Mean-Shift:** These methods determine cluster count automatically. Silhouette Score is useful for internal validation, while MI can measure agreement with known labels.
- **BIRCH and Hierarchical Clustering:** Since BIRCH builds a tree structure, CH Index is well suited for assessing clustering quality. Hierarchical clustering benefits from Cophenetic Correlation as it measures the quality of hierarchy preservation.

Choosing the right evaluation metric is crucial to properly assess clustering performance. Metrics like Silhouette Score, ARI, and DBI align well with specific clustering objectives, while alternative metrics address weaknesses such as noise handling and varying cluster shapes. This selection ensures a balanced and meaningful clustering assessment.

8 Conclusion

In Part I of this project, we conducted an in-depth study of clustering algorithms, examining their theoretical foundations, computational complexities, and evaluation metrics. This analysis provided valuable insights into the strengths and weaknesses of different clustering techniques and the importance of choosing the right evaluation method.

One of the key findings is that no single clustering algorithm performs optimally for all types of datasets. K-means is efficient and widely used, but it assumes spherical clusters and struggles with non-convex shapes. Density-based methods such as DBSCAN and OPTICS overcome this limitation by identifying arbitrarily shaped clusters, but they require careful parameter tuning and may fail in datasets with varying densities. Hierarchical clustering methods like BIRCH and Agglomerative Clustering provide hierarchical structures useful for analysis, though they typically have higher computational costs. Gaussian Mixture Models (GMM) introduce probabilistic clustering, making them more flexible, but their performance depends on selecting an appropriate number of components.

Computational complexity plays a significant role in algorithm selection, particularly for large datasets. Standard k-means has a time complexity of $O(nkdT)$, which can be significantly improved using optimizations such as Elkan's triangle inequality method. Density-based methods, while effective in many scenarios, require nearest-neighbor searches, leading to worst-case complexities of

Algorithm	Best Metric	Alternative Metric	Justification
k-Means	Silhouette Score	Adjusted Rand Index (ARI)	k-means minimizes intra-cluster variance, making silhouette score a natural fit. ARI is useful when ground truth labels are available.
k-Medoids	Davies-Bouldin Index (DBI)	Silhouette Score	k-Medoids selects actual points as centers, and DBI effectively measures compactness and separation.
Gaussian Mixture Model (GMM)	Bayesian Information Criterion (BIC)	Adjusted Mutual Information (AMI)	GMM optimizes likelihood, making BIC the best choice. AMI is useful for supervised evaluation.
DBSCAN	Adjusted Rand Index (ARI)	Silhouette Score	DBSCAN identifies arbitrary-shaped clusters, so ARI is useful for label-based validation. Silhouette Score helps assess cluster separation.
OPTICS	Reachability Plot	Davies-Bouldin Index (DBI)	OPTICS produces hierarchical clusters, making reachability plots the best evaluation tool. DBI assesses compactness.
Affinity Propagation	Silhouette Score	Mutual Information (MI)	Since AP determines clusters automatically, silhouette score helps validate separation, and MI compares assignments to ground truth.
Mean-Shift	Silhouette Score	Davies-Bouldin Index (DBI)	Mean-Shift finds density peaks, making silhouette score useful. DBI evaluates separation balance.
BIRCH	Calinski-Harabasz Index	Silhouette Score	BIRCH merges micro-clusters hierarchically, so CH Index is best for evaluation. Silhouette Score gives additional insights.
Agglomerative Hierarchical Clustering	Cophenetic Correlation	Silhouette Score	Cophenetic Correlation measures how well hierarchical clustering preserves distances. Silhouette Score assesses individual cluster quality.

Table 10: Best and Alternative Metrics for Clustering Algorithms

$O(n^2)$, though spatial indexing can improve efficiency. Hierarchical clustering methods generally exhibit $O(n^2)$ complexity, making them impractical for very large datasets unless optimized.

A major challenge in clustering is evaluating the quality of results, as clustering is inherently unsupervised and lacks a universal accuracy measure. Internal validation metrics such as the Silhouette Score assess compactness and separation but do not confirm whether clusters match an expected structure. External validation metrics like the Adjusted Rand Index (ARI) and Mutual Information (MI) are effective when ground truth labels exist but are not applicable in purely unsupervised settings. Other metrics, including the Davies-Bouldin Index (DBI) and Calinski-Harabasz Index (CH), provide insight into intra-cluster variance and inter-cluster separation but can be biased in certain conditions.

Furthermore, our analysis of k-means accuracy revealed that an exact k-means clustering can still yield a poor silhouette score, particularly when clusters are not well-separated or when the data contains elongated or overlapping structures. This highlights the limitations of relying solely on silhouette-based evaluations and reinforces the need for multiple evaluation criteria.

Overall, this phase of the project provided a deep understanding of clustering methodologies, their mathematical principles, and their evaluation techniques. The study emphasized that clustering performance depends not only on the algorithm itself but also on data characteristics, preprocessing techniques, and the choice of evaluation metrics. In Part II, we will focus on implementing these clustering algorithms on real datasets, conducting experiments to compare their performance, and analyzing their practical applicability in different scenarios.

References

[1] Charu C Aggarwal and Charu C Aggarwal. 2017. *An introduction to outlier analysis*. Springer.

[2] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. 2020. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics* 9, 8 (2020), 1295.

[3] Saket Anand, Sushil Mittal, Oncel Tuzel, and Peter Meer. 2013. Semi-supervised kernel mean shift clustering. *IEEE transactions on pattern analysis and machine intelligence* 36, 6 (2013), 1201–1215.

[4] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering points to identify the clustering structure. *ACM Sigmod record* 28, 2 (1999), 49–60.

[5] Deepak Arunachalam and Niraj Kumar. 2018. Benefit-based consumer segmentation and performance evaluation of clustering approaches: An evidence of data-driven decision-making. *Expert Systems with Applications* 111 (2018), 11–34.

[6] Panthadeep Bhattacharjee and Pinaki Mitra. 2021. A survey of density based clustering algorithms. *Frontiers of Computer Science* 15 (2021), 1–27.

[7] Shukui Bo and Yongju Jing. 2012. Image clustering using mean shift algorithm. In *2012 Fourth International Conference on Computational Intelligence and Communication Networks*. IEEE, 327–330.

[8] Hans-Hermann Bock. 2007. Clustering methods: a history of k-means algorithms. *Selected contributions in data analysis and classification* (2007), 161–172.

[9] Athman Bouguettaya, Qi Yu, Xumin Liu, Xiangmin Zhou, and Andy Song. 2015. Efficient agglomerative hierarchical clustering. *Expert Systems with Applications* 42, 5 (2015), 2785–2797.

[10] Brilliant. 2025. Gaussian Mixture Model. https://brilliant.org/wiki/gaussian-mixture-model/

[11] Dorin Comaniciu and Peter Meer. 1999. Mean shift analysis and applications. In *Proceedings of the seventh IEEE international conference on computer vision*, Vol. 2. IEEE, 1197–1203.

[12] Caitlin E Coombes, Xin Liu, Zachary B Abrams, Kevin R Coombes, and Guy Brock. 2021. Simulation-derived best practices for clustering clinical data. *Journal of biomedical informatics* 118 (2021), 103788.

[13] Ian Davidson, Michael Livanos, Antoine Gourru, Peter Walker, and Julien Velcin SS Ravi. 2024. An Exemplars-Based Approach for Explainable Clustering: Complexity and Efficient Approximation Algorithms. In *Proceedings of the 2024 SIAM International Conference on Data Mining (SDM)*. SIAM, 46–54.

[14] Renato Cordeiro De Amorim. 2016. A survey on feature weighting based k-means algorithms. *Journal of Classification* 33 (2016), 210–242.

[15] Christian Döring, Marie-Jeanne Lesot, and Rudolf Kruse. 2006. Data analysis with fuzzy clustering methods. *Computational Statistics & Data Analysis* 51, 1 (2006), 192–214.

[16] Delbert Dueck. 2009. *Affinity propagation: clustering data by passing messages*. Ph. D. Dissertation.

[17] Michael B Eisen, Paul T Spellman, Patrick O Brown, and David Botstein. 1998. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences* 95, 25 (1998), 14863–14868.

[18] Charles Elkan. 2003. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th international conference on Machine Learning (ICML-03)*. 147–153.

[19] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, Vol. 96. 226–231.

[20] Sabhia Firdaus and Md Ashraf Uddin. 2015. A survey on clustering algorithms and complexity analysis. *International Journal of Computer Science Issues (IJCSI)* 12, 2 (2015), 62.

[21] Michele Forina, Carla Armanino, and V Raggio. 2002. Clustering with dendrograms on interpretation variables. *Analytica Chimica Acta* 454, 1 (2002), 13–19.

[22] Glenn Fung. 2001. A comprehensive overview of basic clustering algorithms. (2001).

[23] Guojun Gan, Chaoqun Ma, and Jianhong Wu. 2020. *Data clustering: theory, algorithms, and applications*. SIAM.

[24] GeeksforGeeks. 2021. ML | OPTICS Clustering Explanation. https://www.geeksforgeeks.org/ml-optics-clustering-explanation/

[25] Sunila Godara, Rishipal Singh, and Sanjeev Kumar. 2017. Proposed density based clustering with weighted Euclidean distance. *International Journals of Advanced Research in Computer Science and Software Engineering* 7, 6 (2017), 409–412.

[26] Michael J Greenacre and Patrick JF Groenen. 2016. Weighted euclidean biplots. *Journal of Classification* 33 (2016), 442–459.

[27] Sandhya Harikumar and PV Surya. 2015. K-medoid clustering for heterogeneous datasets. *Procedia Computer Science* 70 (2015), 226–237.

[28] Henderi Henderi, Tri Wahyuningsih, and Efana Rahwanto. 2021. Comparison of Min-Max normalization and Z-Score Normalization in the K-nearest neighbor (kNN) Algorithm to Test the Accuracy of Types of Breast Cancer. *International Journal of Informatics and Information Systems* 4, 1 (2021), 13–20.

[29] Tin Tin Hmwe, Nwet Yin Tun Thein, and Khin Mar Cho. 2020. Improving clustering quality using silhouette score. *J. Comput. Appl. Res* 1 (2020), 58–62.

[30] F Mohamed Ilyas and S Silvia Priscila. 2024. An optimized clustering quality analysis in k-means cluster using silhouette scores. In *Explainable AI Applications for Human Behavior Analysis*. IGI Global, 49–63.

[31] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. 1999. Data clustering: a review. *ACM computing surveys (CSUR)* 31, 3 (1999), 264–323.

[32] Amin Karami and Ronnie Johansson. 2014. Choosing DBSCAN parameters automatically using differential evolution. *International Journal of Computer Applications* 91, 7 (2014), 1–11.

[33] Leonard Kaufman and Peter J Rousseeuw. 2009. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons.

[34] Noor Kamal Kaur, Usvir Kaur, and Dheerendra Singh. 2014. K-Medoid clustering algorithm-a review. *Int. J. Comput. Appl. Technol* 1, 1 (2014), 42–45.

[35] Kamran Khan, Saif Ur Rehman, Kamran Aziz, Simon Fong, and Sababady Sarasvady. 2014. DBSCAN: Past, present and future. In *The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014)*. IEEE, 232–238.

[36] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. 2011. Density-based clustering. *Wiley interdisciplinary reviews: data mining and knowledge discovery* 1, 3 (2011), 231–240.

[37] Danial Lashkari and Polina Golland. 2007. Convex clustering with exemplar-based models. *Advances in neural information processing systems* 20 (2007).

[38] Lanlan Li, SY Chen, Qiu Guan, Xiaoyan Du, and ZZ Hu. 2009. Point cloud simplification based on an affinity propagation clustering algorithm. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, Vol. 3. IEEE, 163–167.

[39] Youguo Li and Haiyan Wu. 2012. A clustering method based on K-means algorithm. *Physics Procedia* 25 (2012), 1104–1109.

[40] Pinyan Liu, Han Yuan, Yilin Ning, Bibhas Chakraborty, Nan Liu, and Marco Aurélio Peres. 2024. A modified and weighted Gower distance-based clustering analysis for mixed type data: a simulation and empirical analyses. *BMC Medical Research Methodology* 24, 1 (2024), 305.

[41] Xiang Liu and Jingting Xu. 2018. Based on multiple time series affinity propagation algorithm. In *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*. IEEE, 1500–1503.

[42] Jinwen Ma, Lei Xu, and Michael I Jordan. 2000. Asymptotic convergence rate of the EM algorithm for Gaussian mixtures. *Neural Computation* 12, 12 (2000), 2881–2907.

[43] Tagaram Soni Madhulatha. 2011. Comparison between k-means and k-medoids clustering algorithms. In *International Conference on Advances in Computing and Information Technology*. Springer, 472–481.

[44] T Soni Madhulatha. 2012. An overview on clustering methods. *arXiv preprint arXiv:1205.1117* (2012).

[45] Amandeep Kaur Mann and Navneet Kaur. 2013. Survey paper on clustering techniques. *International journal of science, engineering and technology research* 2, 4 (2013), 0803–0806.

[46] Miles McCrory and Spencer A Thomas. 2024. Cluster Metric Sensitivity to Irrelevant Features. *arXiv preprint arXiv:2402.12008* (2024).

[47] Daniel Müllner. 2011. Modern Hierarchical, Agglomerative Clustering Algorithms. https://arxiv.org/abs/1109.2378

[48] Godwin Ogbuabor and FN Ugwoke. 2018. Clustering algorithm for a healthcare dataset using silhouette score value. *Int. J. Comput. Sci. Inf. Technol* 10, 2 (2018), 27–37.

[49] Shraddha Pandit and Suchita Gupta. 2011. A comparative study on distance measuring approaches for clustering. *International journal of research in computer science* 2, 1 (2011), 29.

[50] Thrasyvoulos N Pappas and Nikil S Jayant. 1989. An adaptive clustering algorithm for image segmentation. In *International Conference on Acoustics, Speech, and Signal Processing,*. IEEE, 1667–1670.

[51] Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for K-medoids clustering. *Expert systems with applications* 36, 2 (2009), 3336–3341.

[52] SGOPAL Patro and Kishore Kumar Sahu. 2015. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462* (2015).

[53] R Venkata Rao and Dinesh Singh. 2012. Weighted Euclidean distance based approach as a multiple attribute decision making method for plant or facility layout design selection. *International Journal of Industrial Engineering Computations* 3, 3 (2012), 365–382.

[54] Alan P Reynolds, Graeme Richards, Beatriz de la Iglesia, and Victor J Rayward-Smith. 2006. Clustering rules: a comparison of partitioning and hierarchical clustering algorithms. *Journal of Mathematical Modelling and Algorithms* 5 (2006), 475–504.

[55] Douglas A Reynolds et al. 2009. Gaussian mixture models. *Encyclopedia of biometrics* 741, 659-663 (2009), 3.

[56] Mayra Z Rodriguez, Cesar H Comin, Dalcimar Casanova, Odemir M Bruno, Diego R Amancio, Luciano da F Costa, and Francisco A Rodrigues. 2019. Clustering algorithms: A comparative approach. *PloS one* 14, 1 (2019), e0210236.

[57] Arkajyoti Saha and Swagatam Das. 2017. Feature-weighted clustering with inner product induced norm based dissimilarity measures: an optimization perspective. *Machine Learning* 106 (2017), 951–992.

[58] Bhaba R Sarker and Khan M Saiful Islam. 1999. Relative performances of similarity and dissimilarity measures. *Computers & industrial engineering* 37, 4 (1999), 769–807.

[59] Peiman Alipour Sarvari, Alp Ustundag, and Hidayet Takci. 2016. Performance evaluation of different customer segmentation approaches based on RFM and demographics analysis. *Kybernetes* 45, 7 (2016), 1129–1157.

[60] Friedhelm Schwenker and Edmondo Trentin. 2014. Pattern classification and clustering: A review of partially supervised learning approaches. *Pattern Recognition Letters* 37 (2014), 4–14.

[61] Ketan Rajshekhar Shahapure and Charles Nicholas. 2020. Cluster quality analysis using silhouette score. In *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*. IEEE, 747–748.

[62] Ali Seyed Shirkhorshidi, Saeed Aghabozorgi, and Teh Ying Wah. 2015. A comparison study on similarity and dissimilarity measures in clustering continuous data. *PloS one* 10, 12 (2015), e0144059.

[63] Sami Sieranoja and Pasi Fränti. 2025. Fast agglomerative clustering using approximate traveling salesman solutions. *Journal of Big Data* 12 (2025), 21. https://doi.org/10.1186/s40537-024-01053-x

[64] Kristina P Sinaga and Miin-Shen Yang. 2020. Unsupervised K-means clustering algorithm. *IEEE access* 8 (2020), 80716–80727.

[65] Deepti Sisodia, Lokesh Singh, Sheetal Sisodia, and Khushboo Saxena. 2012. Clustering techniques: a brief survey of different clustering algorithms. *International Journal of Latest Trends in Engineering and Technology (IJLTET)* 1, 3 (2012), 82–87.

[66] Chowdam Sreedhar, Nagulapally Kasiviswanath, and Pakanti Chenna Reddy. 2017. Clustering large datasets using K-means modified inter and intra clustering (KM-I2C) in Hadoop. *Journal of Big Data* 4, 1 (2017), 27.

[67] Nidhi Suthar, P Indr, and P Vinit. 2013. A technical survey on DBSCAN clustering algorithm. *Int. J. Sci. Eng. Res* 4, 5 (2013), 1775–1781.

[68] Rizki Suwanda, Zulfahmi Syahputra, and Elvi M Zamzami. 2020. Analysis of euclidean distance and manhattan distance in the K-means algorithm for variations number of centroid K. In *Journal of Physics: Conference Series*, Vol. 1566. IOP Publishing, 012058.

[69] Peter Trebuňa, Jana Halčinová, Milan Fil'o, and Jaromír Markovič. 2014. The importance of normalization and standardization in the process of clustering. In *2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 381–385.

[70] Sarita Tripathy and Laxman Sahoo. 2020. Improved method for noise detection by DBSCAN and angle based outlier factor in high dimensional datasets. In *ICCCE 2019: Proceedings of the 2nd International Conference on Communications and Cyber Physical Engineering*. Springer, 213–221.

[71] Santosh Kumar Uppada. 2014. Centroid based clustering algorithms—A clarion study. *International Journal of Computer Science and Information Technologies* 5, 6 (2014), 7309–7313.

[72] Kushal Virupakshappa and Erdal Oruklu. 2019. Unsupervised machine learning for ultrasonic flaw detection using gaussian mixture modeling, k-means clustering and mean shift clustering. In *2019 IEEE International Ultrasonics Symposium (IUS)*. IEEE, 647–649.

[73] Xu Wang and Yusheng Xu. 2019. An improved index for clustering validation based on Silhouette index and Calinski-Harabasz index. In *IOP Conference Series: Materials Science and Engineering*, Vol. 569. IOP Publishing, 052024.

[74] Rui Xu and Donald Wunsch. 2005. Survey of clustering algorithms. *IEEE Transactions on neural networks* 16, 3 (2005), 645–678.
[75] Abhinandan Yadav and P Singh. 2024. Refining color scheme generation: Iterative k-means clustering and ari evaluation. *Journal of Informatics Electrical and Electronics Engineering (JIEEE)* (2024), 1–12.
[76] Hui Yin, Amir Aryani, Stephen Petrie, Aishwarya Nambissan, Aland Astudillo, and Shengyuan Cao. 2024. A rapid review of clustering algorithms. *arXiv preprint arXiv:2401.07389* (2024).
[77] Hong Yu-ling et al. 2016. Research on Affinity Propagation algorithm based on common neighbors. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 003504–003509.
[78] Jing Zhang, Mingyi He, and Yuchao Dai. 2014. Modified affinity propagation clustering. In *2014 IEEE China Summit & International Conference on Signal and Information Processing (ChinaSIP)*. IEEE, 505–509.
[79] Mimi Zhang. 2022. Weighted clustering ensemble: A review. *Pattern Recognition* 124 (2022), 108428.
[80] Qinghe Zhang and Xiaoyun Chen. 2010. Agglomerative hierarchical clustering based on affinity propagation algorithm. In *2010 Third International Symposium on Knowledge Acquisition and Modeling*. IEEE, 250–253.
[81] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1997. BIRCH: A new data clustering algorithm and its applications. *Data mining and knowledge discovery* 1 (1997), 141–182.
[82] Yi Zhang, Miaomiao Li, Siwei Wang, Sisi Dai, Lei Luo, En Zhu, Huiying Xu, Xinzhong Zhu, Chaoyun Yao, and Haoran Zhou. 2021. Gaussian mixture model clustering with incomplete data. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 17, 1s (2021), 1–14.
[83] Ruofei Zhao, Yuanzhi Li, and Yuekai Sun. 2020. Statistical convergence of the EM algorithm on Gaussian mixture models. (2020).
[84] Yi Zhou, Yi Feng, Vahid Tarokh, Vadas Gintautas, Jessee McClelland, and Denis Garagic. 2019. Multi-level mean-shift clustering for single-channel radio frequency signal separation. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 1–6.

Appendix A: Tables

A Teamwork Distribution

The following table outlines the contributions of each team member to different sections of the project:

Team Member	Contributions
Gideon Peters	• Literature review on clustering algorithms. • Implemented k-means, k-medoids, and GMM algorithms. • Conducted complexity analysis for centroid-based clustering methods. • Wrote sections on k-means accuracy and improvements.
Bhoomi	• Implemented DBSCAN, OPTICS, and Affinity Propagation. • Conducted complexity analysis for density-based clustering methods. • Analyzed clustering evaluation metrics and provided mathematical definitions. • Wrote sections on metrics appropriateness.
Anjolaoluwa Lasekan	• Implemented hierarchical clustering (BIRCH, Agglomerative). • Prepared experiments and dataset preprocessing. • Evaluated results using silhouette score, Davies-Bouldin Index, and ARI. • Compiled the final report and formatted the document in LaTeX.
Masum Newaz	• Highlighted affinity propagation. • Worked on mean shift clustering algorithm. • Compiled the final report and formatted the document in LaTeX.

Table 11: Teamwork Distribution for the Project

All team members contributed equally to discussions, debugging, and reviewing the report. The project was a collaborative effort, and responsibilities were distributed at random.