COMP-6651: Clustering Algorithms Analysis

Nitheesh Kumar Kambala (40299620), Divyesh Pravinkumar Patel (40301889),
Sotirios Damas (40317602), Farid Faraji (40324126)

I. Clustering Algorithms Complexity Analysis

A. Density based

a) DBSCAN (Density-Based Spatial Clustering of Applications with Noise): The base algorithm for DBSCAN is as follows

Algorithm 1 DBSCAN Algorithm

```
1: Input: A set of data points $P$, minimum number of points $MinPts$, neighborhood radius $\epsilon$
2: Output: A set of clusters and noise points
3: Initialize all points in $P$ as unvisited
4: Initialize an empty list of clusters $C = \emptyset$
5: for each unvisited point $p \in P$ do
6:   if $p$ is unvisited then
7:     Mark $p$ as visited
8:     Retrieve all points in the $\epsilon$-neighborhood of $p$
9:     if the number of points in the neighborhood of $p$ is greater than or equal to $MinPts$ then
10:      Create a new cluster and add $p$ to the cluster
11:      ExpandCluster(p, $MinPts$, $\epsilon$)
12:    else
13:      Mark $p$ as noise
14:    end if
15:  end if
16: end for
17: Return the set of clusters and noise points
```

Algorithm 2 ExpandCluster Algorithm

```
1: Input: Core point $p$, minimum number of points $MinPts$, neighborhood radius $\epsilon$
2: Output: Expanded cluster
3: Initialize the cluster with $p$
4: Initialize the list of points to check with the $\epsilon$ - neighborhood of $p$
5: while there are points to check in the list do
6:   Take the first point $q$ from the list
7:   if $q$ is unvisited then
8:     Mark $q$ as visited
9:     Retrieve all points in the $\epsilon$-neighborhood of $q$
10:    if the number of points in the neighborhood of $q$ is greater than or equal to $MinPts$ then
11:      Add all unvisited neighbors of $q$ to the list
12:    end if
13:  end if
14:  if $q$ is not in any cluster then
15:    Add $q$ to the current cluster
16:  end if
17: end while
```

The detailed complexity analysis of DBSCAN algorithm-1 is as follows

- Line 3 initialize all points as unvisited - $O(n)$
- Line 4 initialize an empty list of clusters - $O(1)$
- Line 5 Loop over all points $p \in P$ - $O(n)$
- Line 6 Check if p is unvisited - $O(1)$
- Line 7 Mark p as Visited - $O(1)$
- Line 8 Retrieve all points in neighborhood - $O(n)$ for worst case
- Line 9 Check if the number of neighbors $\geq$ MinPts - $O(1)$
- Line 10 Create a new cluster and add p - $O(1)$
- Line 11 call $expandCluster(p, MinPts, \epsilon)$ - $O(n)$ for worst case
- Line 13 Mark p as noise if not enough neighbors - $O(1)$
- Line 16 Return clusters and noise points - $O(1)$
- Total Complexity of DBSCAN algorithm
  - The worst case scenario occurs when every point is a core point and needs a ExpandCluster call.
  - The retrieval of neighbors requires $O(n)$ in the worst case for each of the n points.
  - ExpandCluster can also take $O(n)$ per call, leading to a total worst-case complexity of $O(n^2)$.

The detailed complexity analysis of ExpandCluster algorithm-2 is as follows

- Line 3 Initialize cluster with p - $O(1)$
- Line 4 Initialize list with ϵ-neighborhood of p - $O(n)$ for worst case
- Line 5 while there are points to check - $O(n)$ for worst case
- Line 6 Retrieve first point q - $O(1)$
- Line 7 Check if q is unvisited - $O(1)$
- Line 8 Mark q as visited - $O(1)$
- Line 9 Retrieve neighbors of q - $O(n)$ for worst case
- Line 10 Check if neighbors $\geq MinPts$ - $O(1)$
- Line 11 Add unvisited neighbors of q to the list - $O(1)$ for worst case
- Line 14 Check if q is not in any cluster - $O(1)$
- Line 15 Add q to the current cluster - $O(1)$
- Total complexity of ExpandCLuster Algorithm is as follows
  - The while loop runs at most $O(n)$ times, because each point is processed once.
  - Retrieving neighbors take O(n) worst case per point.
  - This leads to $O(n^2)$ worst-case complexity for ExpandCluster.

b) OPTICS (Ordering Points to Identify Clustering Structure): The base algorithm for OPTICS is as follows

The detailed complexity analysis of OPTICS algorithm-3 is

Algorithm 3 OPTICS Algorithm

```
1: Input: A set of points $P$, minimum number of points $MinPts$, maximum distance $\epsilon$
2: Output: Ordered list of points with reachability distances
3: Initialize all points as unvisited
4: Initialize an empty ordered list $O$
5: for each unvisited point $p \in P$ do
6:     ExpandClusterOrder($p, P, MinPts, \epsilon, O$)
7: end for
8: Return the ordered list $O$
```

Algorithm 4 ExpandClusterOrder

```
1: Input: A point $p$, dataset $P$, $MinPts$, $\epsilon$, ordered list $O$
2: Output: Updates the order list $O$ with reachability distances
3: Compute the $\epsilon$-neighborhood of $p$
4: Compute core distance of $p$ (distance to its $MinPts$-th nearest neighbor)
5: Add $p$ to ordered list $O$ with reachability distance undefined
6: if $p$ is a core point then
7:     Initialize a priority queue $Q$ sorted by reachability distance
8:     For each neighbor $q$ of $p$, update reachability distance if necessary and add $q$ to $Q$
9:     while $Q$ is not empty do
10:        Extract the point $r$ with the smallest reachability distance from $Q$
11:        Compute $\epsilon$-neighborhood of $r$
12:        Compute core distance of $r$
13:        Add $r$ to ordered list $O$
14:        if $r$ is a core point then
15:            Update reachability distances of its neighbors and add them to $Q$
16:        end if
17:    end while
18: end if
```

as follows

- Line 3 Initialize all points as unvisited - $O(n)$
- Line 4 Initialize an empty ordered list O - $O(1)$
- Line 5 Loop over all points $p \in P$ - $O(n)$
- Line 6 call $ExpandClusterOrder(p, P, MinPts, \epsilon, O)$ - $O(nlogn)$ on average
- Line 8 Return the ordered list O - $O(1)$
- Total Complexity of OPTICS Algorithm is as follows
  - The worst-cas complexity occurs when ExpandClusterOrder is called for every point.
  - Thus, the overall complexity is $O(nlogn)$ on average and $O(n^2)$ in the worst case.

The detailed complexity analysis of ExpandClusterOrder algorithm-4 is as follows

- Line 3 Compute the ϵ-neighborhood of p - $O(n)$ for worst case.
- Line 4 Compute core distace of p - $O(n)$
- Line 5 Add p to ordered list O - $O(1)$
- Line 6 Check if p is a core point - $O(1)$
- Line 7 Initialize a priority queue Q - $O(1)$
- Line 8 Update reachability distances and add neighbors to Q - $O(nlogn)$ heap insertion
- Line 9-17 Process points in Q using a priority queue - $O(nlogn)$ heap operations
- Line 18 Compute neighbors of r - $O(n)$ for worst case
- Line 19 Compute core distance of r - $O(n)$
- Line 20 Add r to the ordered list O - $O(1)$
- Line 22 Update reachability distances of neighbors and add them to Q - $O(nlogn)$
- Total complexity of ExpandClusterOrder algorithm is as follows
  - The while loop iterates over all points in the dataset at most once.
  - Each neighbor search takes $O(n)$.
  - Heap opeartions add a logarithmic factor.
  - The total complexity per ExpandClusterOrder call is $O(nlogn)$ on average and $O(n^2)$ in the worst case.

B. Distribution based

Algorithm 5 Gaussian Mixture Model (GMM) via EM Algorithm

```
1: Input:
     • Dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, with each $\mathbf{x}_i \in \mathbb{R}^d$
     • Number of components $K$
     • (Optional) Initial parameters $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k\}_{k=1}^K$
     • Convergence threshold or maximum iterations $\ell$
2: Output:
     • Estimated parameters $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k\}_{k=1}^K$
     • Responsibilities $\gamma_{i,k}$ (soft cluster assignments)
3: Initialization: Randomly initialize or use a heuristic (e.g., $k$-means) to set $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$, and $\pi_k$ for each of the $K$ Gaussian components.
4: while not converged and iteration $\leq \ell$ do
5:     E-step (Compute Responsibilities):
           $$\gamma_{i,k} = \frac{\pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$
           for each $i = 1, \dots, n,\ k = 1, \dots, K$
6:     M-step (Update Parameters):
           $$N_k = \sum_{i=1}^n \gamma_{i,k}, \quad \pi_k = \frac{N_k}{n},$$
           $$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=1}^n \gamma_{i,k}\, \mathbf{x}_i,$$
           $$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{i=1}^n \gamma_{i,k}\, (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top$$
7:     Evaluate the log-likelihood or parameter change to check for convergence
8: end while
9: Return: The final parameters $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k\}$ and the responsibilities $\gamma_{i,k}$ for each data point $\mathbf{x}_i$
```

a) Gaussian Mixture Model (GMM): The detailed complexity analysis of the Gaussian Mixture Model using EM consists of the following main operations:

1) **Probability Calculation (E-step):** For each iteration, we compute the likelihood of each data point $\mathbf{x}_i$ under each of the K Gaussian components:
   - Evaluating a d-dimensional Gaussian $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ can be $\mathcal{O}(d^2)$ after precomputing the inverse of the covariance matrix, since the main cost is the matrix-vector multiplication and determinant calculation.
   - Overall, updating responsibilities for n data points across K components yields $\mathcal{O}(n \cdot K \cdot d^2)$ per iteration.
2) **Parameter Updates (M-step):** Updating the parameters $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k\}$ similarly involves summations over all n data points, weighted by $\gamma_{i,k}$.
   - Computing means $\boldsymbol{\mu}_k$ costs $\mathcal{O}(n \cdot K \cdot d)$.
   - Updating covariances $\boldsymbol{\Sigma}_k$ can cost up to $\mathcal{O}(n \cdot K \cdot d^2)$ if each covariance is full rank (i.e., not diagonal).

   Combined with the E-step, the total time per iteration often remains on the order of $\mathcal{O}(n \cdot K \cdot d^2)$ for full covariance matrices.
3) **Overall Time Complexity:** Let ℓ be the number of EM iterations until convergence. Then the total time complexity is approximately $\mathcal{O}(\ell \cdot n \cdot K \cdot d^2)$. If simpler (e.g., diagonal) covariances are assumed, the d^2 factor can reduce to $\mathcal{O}(d)$.
4) **Total Space Complexity:**
   - *Dataset Storage:* $\mathcal{O}(n \cdot d)$.
   - *Responsibilities:* $\mathcal{O}(n \cdot K)$ for storing $\gamma_{i,k}$.
   - *Parameters:* $\mathcal{O}(K \cdot d^2)$ for storing full covariance matrices, plus $\mathcal{O}(K \cdot d)$ for means and $\mathcal{O}(K)$ for mixing coefficients.

   Hence, the total space complexity is $\mathcal{O}(n \cdot d + n \cdot K + K \cdot d^2)$, which for large n is dominated by $\mathcal{O}(n \cdot d)$ and $\mathcal{O}(n \cdot K)$.

Thus, in the base case (with full covariance matrices), the overall time complexity for the Gaussian Mixture Model (EM algorithm) is $\mathcal{O}(\ell \cdot n \cdot K \cdot d^2)$ and the space complexity is $\mathcal{O}(n \cdot K + K \cdot d^2)$, where n is the number of data points, K is the number of Gaussian components, d is the dimensionality, and ℓ is the number of EM iterations.

C. Hierarchical based

a) BIRCH (Balance Iterative Reducing and Clustering using Hierarchies): The detailed complexity analysis of BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) consists of the following main operations:

1) **CF Tree Construction:** BIRCH incrementally inserts each data point into a CF Tree by traversing from the root to the appropriate leaf node. At each step, it compares the point with subclusters to find the closest one.
   - *Distance Calculation:* Each comparison between a point and a subcluster's centroid takes $\mathcal{O}(d)$ time, where d is the number of dimensions.
   - *Traversal Cost:* The tree height is usually small due to the branching factor (B). In many practical cases, insertion is close to $\mathcal{O}(d)$ or $\mathcal{O}(\log(n) \cdot d)$ per point, leading to a total of $\mathcal{O}(n \cdot d)$ or $\mathcal{O}(n \log(n) \cdot d)$ over n points.
2) **Node Splitting:** If adding a point causes a subcluster to exceed the threshold T (controlling maximum subcluster diameter), the leaf node may split, which can propagate upward if the branching factor is exceeded. This split is typically infrequent if T is well-chosen, but in the worst case (if T is too small), repeated splits can degrade performance to $\mathcal{O}(n^2 \cdot d)$.
3) **Optional Global Clustering:** After building the CF Tree, BIRCH often applies a final clustering step (e.g.,

Algorithm 6 BIRCH Algorithm

```
1: Input:
     • Dataset $D = \{x_1, x_2, \ldots, x_n\}$ of $n$ points in $d$ dimensions
     • Branching factor $B$ (max children per node)
     • Threshold $T$ (maximum radius or diameter of leaf sub-clusters)
     • (Optional) Desired number of clusters $L$ for a final clustering phase
2: Output:
     • CF-Tree (Clustering Feature Tree)
     • (Optional) Final set of clusters if the final clustering step is performed
3: Initialize an empty CF-Tree with the root as a leaf node
4: for each point $x_i \in D$ do
5:    Insert $x_i$ into the CF-Tree:
6:       Start from the root node
7:       If the current node is not a leaf:
8:          Find the child sub-cluster whose centroid is closest to $x_i$ (using $(N, LS, SS)$)
9:          Descend into that child node
10:      Else (the current node is a leaf):
11:         Find the leaf-entry $L_j$ whose centroid is closest to $x_i$
12:         If adding $x_i$ to $L_j$ keeps the sub-cluster diameter $\leq T$:
13:            Update the CF of $L_j$ by incrementing $N$, adding $x_i$ to $LS$, and adding $x_i^2$ to $SS$
14:         Else:
15:            Split $L_j$ into two sub-clusters according to BIRCH splitting criteria
16:            If the leaf node exceeds branching factor $B$:
17:               Split the leaf node
18:            If a non-leaf node splits and exceeds $B$:
19:               Recursively split internal nodes up to the root if necessary
20: end for
21: (Optional) Condense or refine the CF-Tree (remove outliers or merge small sub-clusters)
22: (Optional) Final clustering phase: apply a standard clustering algorithm (e.g., $k$-means) on the leaf sub-cluster centroids if desired
23: Return: The final CF-Tree (and optionally the final clusters)
```

k-means) on the leaf subclusters. If m is the number of leaf entries (where $m \ll n$), this step is typically $\mathcal{O}(\ell \cdot m \cdot k \cdot d)$ (for k-means), which is much smaller compared to clustering all n points directly.

4) **Total Time Complexity:** In typical cases, the CF Tree construction and maintenance yields about $\mathcal{O}(n \cdot d)$ or $\mathcal{O}(n \log(n) \cdot d)$ time. However, in the worst case, many splits can lead to $\mathcal{O}(n^2 \cdot d)$.
5) **Total Space Complexity:** Storing n data points with d features requires $\mathcal{O}(n \cdot d)$ space. The CF Tree itself is usually much smaller than n, but in the worst case (if the threshold T is not effective in merging points), it can grow and approach $\mathcal{O}(n)$ leaf entries. Overall, BIRCH commonly requires $\mathcal{O}(n \cdot d)$ space in typical scenarios.

Thus, the overall time complexity for the BIRCH algorithm is typically $\mathcal{O}(n \cdot d)$ to $\mathcal{O}(n \log n \cdot d)$ (worst-case $\mathcal{O}(n^2 \cdot d)$) and the space complexity is approximately $\mathcal{O}(n \cdot d)$.

Algorithm 7 Agglomerative Hierarchical Clustering

```
1: Input:
     • Dataset $D = \{x_1, x_2, \ldots, x_n\}$
     • Distance function $dist(x_i, x_j)$ (e.g., Euclidean)
     • Linkage criterion (single, complete, average, Ward)
     • (Optional) Desired number of clusters $K$
2: Output:
     • A dendrogram representing the merge history
     • (Optional) Final set of clusters if $K$ is specified
3: Initialize each point $x_i \in D$ as its own cluster; set $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$
4: Construct a distance matrix $Dist$ where $Dist[i, j] = dist(x_i, x_j)$
5: while the number of clusters in $\mathcal{C}$ is greater than $K$ (or until one cluster remains if $K$ is not specified) do
6:     Find the pair of clusters $(C_p, C_q)$ in $\mathcal{C}$ with the smallest distance based on the linkage criterion
7:     Merge $C_p$ and $C_q$ into a new cluster $C_{\text{new}}$
8:     Update $\mathcal{C}$: remove $C_p$ and $C_q$, then add $C_{\text{new}}$
9:     Update the distance matrix $Dist$:
10:        Remove rows and columns corresponding to $C_p$ and $C_q$
11:        Compute distances from $C_{\text{new}}$ to all other clusters in $\mathcal{C}$ based on the chosen linkage method
12:    Record the merge of $(C_p, C_q)$ in the dendrogram
13: end while
14: Return: The dendrogram (and the final clusters if $K$ is specified)
```

b) Agglomerative Hierarchy clustering algorithm: The detailed complexity analysis of Agglomerative Hierarchical Clustering consists of the following main operations:

1) **Distance Matrix Construction:** In the standard naive approach, we compute and store the pairwise distances between all n data points in a matrix of size $n \times n$.
   - *Distance Calculation:* Each pairwise comparison is $\mathcal{O}(d)$, and there are $\frac{n(n-1)}{2}$ pairs, leading to $\mathcal{O}(n^2 \cdot d)$.
   - *Matrix Storage:* The distance matrix itself requires $\mathcal{O}(n^2)$ space.
2) **Merging Clusters:** The algorithm starts with n singleton clusters. At each iteration, it merges the two closest clusters according to a chosen linkage criterion (e.g., single, complete, average, or Ward's).
   - *Finding Closest Clusters:* Naively scanning the entire distance matrix can be $\mathcal{O}(n^2)$ each iteration.
   - *Updating the Distance Matrix:* After a merge, we remove the rows/columns corresponding to the merged clusters and add/update a row/column for the new merged cluster. This also requires up to $\mathcal{O}(n)$ operations each time.

   Since we perform $n - 1$ merges to go from n clusters down to 1, the repeated scans lead to high complexity.
3) **Total Time Complexity:** In the naive implementation, finding and merging pairs over $n-1$ iterations is $\mathcal{O}(n^3)$, since each merge step can be $\mathcal{O}(n^2)$ and we have roughly n merges. Building the distance matrix adds $\mathcal{O}(n^2 \cdot d)$, but $\mathcal{O}(n^3)$ dominates for large n.
4) **Total Space Complexity:** The algorithm requires $\mathcal{O}(n^2)$ space to store the distance matrix, which is typically the limiting factor. Additional bookkeeping for cluster memberships or dendrogram storage is $\mathcal{O}(n)$, but is negligible compared to the matrix.

Thus, the overall time complexity for the naive Agglomerative Hierarchical Clustering is $\mathcal{O}(n^3)$ and the space complexity is $\mathcal{O}(n^2)$, where n is the number of data points and d is the dimension for each data point.

D. Centroid/medoid based

a) K-Means: K-Means is a centroid-based clustering algorithm that partitions a dataset of n data points into K clusters. The algorithm iteratively refines the cluster assignments by minimizing the intra-cluster variance, which is the sum of squared distances between each data point and the centroid of its assigned cluster. Simultaneously, this process implicitly aims to maximize the inter-cluster variance by ensuring that points assigned to different clusters are as far apart as possible. The key steps in the algorithm are the initialization of centroids, the assignment of each data point to the nearest centroid, and the recalculation of centroids based on the current cluster memberships, continuing until a convergence criterion is met. [23]–[25]

The cost function of the K-Means algorithm, often referred to as the within-cluster sum of squares (WCSS) or inertia, is given by:

$$J = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - M_i\|^2$$

where:

- K is the number of clusters
- C_i is the set of data points in the i-th cluster
- M_i is the centroid (mean) of the data points in the i-th cluster

Algorithm 8 K-Means Algorithm

```
1: Input: A dataset $P = \{p_1, p_2, \ldots, p_n\}$ of $n$ data points, the desired number of clusters $K$
2: Output: A set of $K$ clusters $C = \{C_1, C_2, \ldots, C_K\}$ and their corresponding centroids $M = \{M_1, M_2, \ldots, M_K\}$
3: Initialization: Randomly select $K$ data points from $P$ as initial cluster centroids $M = \{M_1, M_2, \ldots, M_K\}$.
4: repeat
5:     for each data point $p \in P$ do
6:         Assignment Step: Assign $p$ to the cluster $C_i$ with the closest centroid $M_i$ using Euclidean distance
7:     end for
8:     for each cluster $C_i \in \{C_1, C_2, \ldots, C_K\}$ do
9:         Update Step: Recalculate the centroid $M_i$ of cluster $C_i$ as the mean of all data points in $C_i$
10:    end for
11: until Convergence is reached (i.e., the centroids $M$ do not change significantly between iterations or a maximum number of iterations is reached)
12: Return the set of clusters $C = \{C_1, C_2, \ldots, C_K\}$ and their corresponding centroids $M = \{M_1, M_2, \ldots, M_K\}$.
```

The detailed complexity analysis of the K-Means Algorithm involves the following main operations:

1) **Distance Calculation:** In each iteration of the assignment step, for every data point p, the algorithm calculates the distance to each of the K centroids to determine the nearest cluster. Calculating the Euclidean distance between a single d-dimensional data point and a centroid takes $\mathcal{O}(d)$ time.
2) **Cluster Assignment:** For each of the n data points, the K-Means algorithm computes the distance to all K centroids and assigns the point to the nearest one. This process requires K distance calculations per data point, resulting in a total time complexity of $\mathcal{O}(n \cdot K \cdot d)$ for the assignment step in each iteration.
3) **Centroid Update:** After all data points have been assigned to clusters, the algorithm recalculates the centroid for each of the K clusters. To compute the mean of the points in a d-dimensional cluster, we need to sum the d features of all points in the cluster and then divide by the number of points in that cluster. Across all K clusters, each of the n data points is considered exactly once during the centroid update. Therefore, the total time complexity for updating all K centroids in one iteration is $\mathcal{O}(n \cdot d)$.
4) **Total Time Complexity per Iteration:** Combining the cluster assignment and centroid update steps, the time complexity for a single iteration of the K-Means algorithm is $\mathcal{O}(n \cdot K \cdot d) + \mathcal{O}(n \cdot d) = \mathcal{O}(n \cdot K \cdot d)$.
5) **Overall Time Complexity:** The K-Means algorithm iteratively performs the assignment and update steps until convergence. The number of iterations required for convergence, denoted by T, is data-dependent and can vary significantly. In the worst-case scenario, T can be very large. Therefore, the overall time complexity of the K-Means algorithm is $\mathcal{O}(T \cdot n \cdot K \cdot d)$. In practice, however, the algorithm often converges in a relatively small number of iterations.
6) **Total Space Complexity:** The space required by the K-Means algorithm is primarily for storing the input dataset and the K centroids. Storing n data points in d dimensions requires $\mathcal{O}(n \cdot d)$ space, and storing K centroids, each in d dimensions, requires $\mathcal{O}(K \cdot d)$ space. Thus, the total space complexity of the K-Means algorithm is $\mathcal{O}(n \cdot d + K \cdot d)$.

Thus, the overall time complexity for the K-Means algorithm is $\mathcal{O}(T \cdot n \cdot K \cdot d)$, and the space complexity is $\mathcal{O}(n \cdot d + K \cdot d)$, where n is the number of data points, K is the number of clusters, d is the number of dimensions, and T is the number of iterations until convergence.

b) K-Medoids: K-Medoids is a partitional clustering algorithm that aims to partition a dataset of n data points into K clusters by selecting actual data points as representative objects, called *medoids*. Unlike K-Means, which uses the mean of the points in a cluster as its centroid, K-Medoids relies on finding the most centrally located data point within a cluster to serve as the medoid. The objective of the algorithm is to minimize the total dissimilarity (sum of distances), typically the Manhattan distance, between each data point and its assigned medoid. This reliance on actual data points makes K-Medoids more robust to the presence of outliers in the dataset. [26]–[28]

The K-Medoids algorithm iteratively refines the set of medoids through a process of assignment and swapping until a stable configuration is reached. The key steps involved are:

- **Initialization:** Randomly selecting K data points from the dataset to serve as initial medoids.
- **Assignment Step:** Assigning each data point in the dataset to the cluster whose medoid is the closest, based on a chosen distance metric (e.g., Manhattan distance).
- **Update Step (Swapping):** For each medoid, considering all other data points as potential new medoids. For each such swap, the total cost (sum of distances from each point to its nearest medoid) is calculated. If a swap results in a lower total cost, the swap is made, and the new set of medoids is retained. This process continues until no further improvement in the total cost can be achieved.

Algorithm 9 K-Medoids Algorithm

```
1: Input: A set of $n$ data points $P = \{p_1, p_2, \ldots, p_n\}$, the desired number of clusters $K$
2: Output: A set of $K$ clusters $C = \{C_1, C_2, \ldots, C_K\}$ and their corresponding medoids $M = \{m_1, m_2, \ldots, m_K\}$, where $M \subseteq P$
3: Initialization: Randomly select $K$ data points from $P$ as initial medoids $M = \{m_1, m_2, \ldots, m_K\}$
4: repeat
5:     Assignment Step:
6:     for each data point $p \in P$ do
7:         Assign $p$ to the cluster $C_i$ corresponding to the closest medoid $m_i \in M$ by calculating the Manhattan distance
8:     end for
9:     Update Step (Swapping):
10:    $best\_cost \leftarrow \infty$
11:    $best\_medoids \leftarrow M$
12:    for each current medoid $m_i \in M$ do
13:        for each non-medoid data point $o \in P \setminus M$ do
14:            Temporarily replace $m_i$ with $o$ to form a new set of medoids $M_{new}$
15:            Calculate the total cost $Cost(M_{new})$ = Manhattan distance$(p, m'))$
16:            if $Cost(M_{new}) < best\_cost$ then
17:                $best\_cost \leftarrow Cost(M_{new})$
18:                $best\_medoids \leftarrow M_{new}$
19:            end if
20:        end for
21:    end for
22:    $M \leftarrow best\_medoids$
23: until No changes occur in the set of medoids $M$ in an iteration
24: Return the set of clusters $C = \{C_1, C_2, \ldots, C_K\}$ (based on the final medoid assignments) and the final set of medoids $M = \{m_1, m_2, \ldots, m_K\}$
```

The detailed complexity analysis of the K-Medoids Algorithm involves the following main operations:

1) **Initialization:** The algorithm randomly selects K initial medoids from the n data points. This involves choosing K indices (or directly the points), which takes $\mathcal{O}(K)$ time.
2) **Data Point Assignment:** In each iteration, every data point p needs to be assigned to the closest of the K medoids. For each data point, the Manhattan distance to all K medoids must be calculated. The Manhattan distance between two d-dimensional points takes $\mathcal{O}(d)$ time. Thus, for n data points and K medoids, the assignment step has a time complexity of $\mathcal{O}(n \cdot K \cdot d)$.
3) **Swapping Step (Medoid Update):** This is the most computationally intensive part of the algorithm. For each of the K current medoids, the algorithm considers swapping it with every non-medoid data point (there are $n - K$ such points). For each potential swap, the total cost of the clustering needs to be calculated. Calculating the total cost involves assigning each of the n data points to the new set of K medoids and summing their distances to their closest medoid, which takes $\mathcal{O}(n \cdot K \cdot d)$ time. Since there are K medoids and up to $n - K$ non-medoid points to consider for swapping, the total time complexity for one swapping step is $\mathcal{O}(K \cdot (n - K) \cdot (n \cdot K \cdot d))$, which simplifies to $\mathcal{O}(K^2 \cdot n^2 \cdot d)$ in the worst case (when K is not significantly smaller than n).
4) **Convergence:** The algorithm continues iterating until the set of medoids no longer changes. Let I be the number of iterations required for convergence. The total time complexity is then the product of the number of iterations and the complexity of each iteration. Since the swapping step dominates the complexity of each iteration, the overall time complexity becomes $\mathcal{O}(I \cdot K^2 \cdot n^2 \cdot d)$.
5) **Total Space Complexity:** The K-Medoids algorithm needs to store the n data points and the K medoids. Each data point and medoid has d dimensions. Therefore, the space complexity is $\mathcal{O}(n \cdot d + K \cdot d)$, which can be simplified to $\mathcal{O}(n \cdot d)$ if $n \gg K$.

Thus, the overall time complexity for the K-Medoids algorithm is $\mathcal{O}(I \cdot K^2 \cdot n^2 \cdot d)$, where n is the number of data points, K is the number of clusters, d is the number of dimensions, and I is the number of iterations until convergence. The space complexity is $\mathcal{O}(n \cdot d)$.

E. Exemplar based

a) Affinity Propagation clustering algorithm: Affinity Propagation autonomously identifies exemplars (cluster centers) through iterative message-passing between data points. Responsibilities quantify a point's suitability as an exemplar, while availabilities aggregate support for candidates. The algorithm uses a similarity matrix S and self-preference values, avoiding pre-specified cluster counts. Messages are damped to prevent oscillations, and convergence yields clusters via $\arg\max(r(i, j) + a(i, j))$.

Here is the based algorithm for the Affinity propagation algorithm.

Space Complexity

- **Primary Components**:
  - **Similarity Matrix** (S):
    - Stores pairwise similarities $s(i, j)$ between all N data points.
    - Requires $O(N^2)$ space. For example, with $N = 10^4$ points, this matrix contains 10^8 entries, occupying approximately **800 MB** of memory (assuming 8-byte floats).
  - **Responsibility Matrix** (R):
    - Tracks "responsibility" values $r(i, j)$, representing how well point j serves as an exemplar for point i.
    - Another $O(N^2)$ matrix, doubling the memory footprint.
  - **Availability Matrix** (A):
    - Stores "availability" values $a(i, j)$, quantifying the accumulated support for point j as an exemplar.

    * Adds a third $O(N^2)$ matrix, leading to a total of $3N^2$ entries.

- **Auxiliary Storage**:
  - Arrays for tracking exemplars and cluster assignments require $O(N)$ space, which is negligible compared to the matrices.
- **Total Space Complexity**:

$$O(N^2) \quad \text{(Dominant term due to the three } N \times N \text{ matrices)}$$

Practical Limitation: For large N (e.g., $N > 10^5$), storing $3N^2$ entries becomes infeasible on standard hardware.

1) Time Complexity:

- **Responsibility Updates**:

$$r(n,k) = s(n,k) - \max_{\substack{l=1 \\ l \neq k}}^{N} (a(n,l) + s(n,l)) \tag{1}$$

  - **Operations per Pair**: For each of the $N \times N$ pairs (n,k), compute the maximum over $N-1$ terms $(a(n,l) + s(n,l)$ for $l \neq k)$.
  - **Cost per Iteration**: $O(N^3)$ operations. For $N = 10^3$, this requires 10^9 operations per iteration.
  - **Bottleneck**: The cubic dependence on N makes this step computationally prohibitive for large datasets.
- **Availability Updates**:
  - *Self-availability (Diagonal Terms)*:

$$a(k,k) = \sum_{\substack{n=1 \\ n \neq k}}^{N} \max(0, r(n,k)) \tag{2}$$

    * Sum $N-1$ terms for each diagonal entry $a(k,k)$.
    * **Cost**: $O(N^2)$ operations per iteration.
  - *Non-Self Availability (Off-Diagonal Terms)*:

$$a(n,k) = \min\left(0, r(k,k) + \sum_{\substack{m=1 \\ m \neq n,k}}^{N} \max(0, r(m,k))\right) \tag{3}$$

    * For each of the $N(N-1)$ off-diagonal pairs (n,k), sum $N-2$ terms.
    * **Cost**: $O(N^3)$ operations per iteration.
- **Assignment Phase**:
  - **Exemplar Identification**: Check $r(k,k) + a(k,k)$ for all N points, costing $O(N)$.
  - **Cluster Assignments**: For each of the N points, find the exemplar with the highest similarity, costing $O(N^2)$.
- **Total Time per Iteration**:

$$O(N^3) \quad \text{(Responsibility and availability updates dominate)}$$

- **Overall Time for T Iterations**:

$$O(T \cdot N^3)$$

Practical Note: Even for moderate N (e.g., $N = 1000$), $T = 100$ iterations require 10^{11} operations, which is computationally intensive.

Algorithm 10 Affinity Propagation Algorithm

```
1: Input: Similarity matrix $S$
2: Output: Exemplars and cluster assignments
3: Initialize $S \leftarrow -D$ (or heuristics), $R \leftarrow 0$, $A \leftarrow 0$
4: repeat
5:   for each point $n$ do
6:     for each potential exemplar $k$ do
7:       $r(n,k) \leftarrow s(n,k) - \max_{l \neq k}(a(n,l) + s(n,l))$
8:     end for
9:   end for
10:  for each potential exemplar $k$ do
11:    $a(k,k) \leftarrow \sum_{n \neq k} \max(0, r(n,k))$
12:    for each point $n \neq k$ do
13:      $a(n,k) \leftarrow \min\left(0, r(k,k) + \sum_{m \neq n,k} \max(0, r(m,k))\right)$
14:    end for
15:  end for
16:  Update termination condition
17:  for each potential exemplar $k$ do
18:    if $r(k,k) + a(k,k) \geq 0$ then
19:      $\mathcal{K} \leftarrow \mathcal{K} \cup \{k\}$
20:      $h(k) \leftarrow k$
21:    end if
22:  end for
23:  for each point $n \notin \mathcal{K}$ do
24:    $h(n) \leftarrow \arg\max_{k \in \mathcal{K}} s(n,k)$
25:  end for
26: until termination condition is met
27: Return exemplars $\mathcal{K}$ and assignments $h(n)$
```

a) Mean-Shift clustering algorithm: Mean-shift clustering identifies clusters by iteratively shifting data points toward regions of higher density. Starting with multivariate data points $\{x_n\}_{n=1}^{N} \subset \mathbb{R}^D$, the algorithm estimates the probability density function using a kernel function (typically Gaussian) with bandwidth parameter σ. Each point iteratively moves toward density modes via weighted updates until convergence, with points converging to the same mode forming clusters.

Detailed Time Complexity Analysis The computational complexity of Mean-Shift clustering requires thorough examination across its multiple phases. For a dataset $\{x_n\}_{n=1}^{N} \subset \mathbb{R}^d$, we analyze each component:

1. Initialization Phase: $\mathcal{O}(N)$ The algorithm initializes each point as a potential cluster centroid, requiring $\mathcal{O}(N)$ operations.

2. Iterative Update Phase For each data point x_n, the algorithm executes an iterative process until convergence:

2.1. Kernel Density Estimation For a single point x in iteration τ, computing the probability weights involves:

$$p(n|x^{(\tau)}) = \frac{\exp\left(-\frac{\|x^{(\tau)} - x_n\|^2}{2\sigma^2}\right)}{\sum_{n'=1}^{N} \exp\left(-\frac{\|x^{(\tau)} - x_{n'}\|^2}{2\sigma^2}\right)} \tag{4}$$

This calculation breaks down into:

- Distance computation: $\|x^{(\tau)} - x_n\|$ for each $n \in \{1, 2, ..., N\}$

Algorithm 11 Mean-Shift (MS) Algorithm

```
1: Input:
     • Dataset $X = \{x_1, x_2, \ldots, x_N\}$ of $N$ points in $\mathbb{R}^d$
     • Bandwidth parameter $\sigma$
     • Convergence threshold $\epsilon$
2: Output:
     • Cluster assignments for data points
3: for each point $x_n \in X$ do
4:     Initialize $\mathbf{x} \leftarrow x_n$
5:     repeat
6:         Compute the probability weights:
               $p(n \mid \mathbf{x}) \leftarrow \frac{\exp\left(-\frac{1}{2}\|\mathbf{x} - x_n\|^2/\sigma^2\right)}{\sum_{n'=1}^{N} \exp\left(-\frac{1}{2}\|\mathbf{x} - x_{n'}\|^2/\sigma^2\right)}$
7:         Update the mean-shift vector:
               $\mathbf{x} \leftarrow \sum_{n=1}^{N} p(n \mid \mathbf{x}) x_n$
8:     until convergence (i.e., $\|\mathbf{x} - z_n\| < \epsilon$)
9:     Assign mode: $z_n \leftarrow \mathbf{x}$
10: end for
11: Apply connected-components algorithm to $\{z_n\}_{n=1}^N$ with threshold $\epsilon$ to form clusters
12: Return: Cluster assignments
```

  - Each Euclidean distance calculation: $\mathcal{O}(d)$ operations
  - Total for N points: $\mathcal{O}(N \cdot d)$ operations
- Exponential evaluations: $\exp(-\frac{\|x^{(\tau)} - x_n\|^2}{2\sigma^2})$ for each $n \in \{1, 2, ..., N\}$
  - Each evaluation: $\mathcal{O}(1)$ operations
  - Total for N points: $\mathcal{O}(N)$ operations
- Normalization (computing denominator): $\sum_{n'=1}^{N} \exp\Big(-\frac{\|x^{(\tau)} - x_{n'}\|^2}{2\sigma^2}\Big)$
  - Summation over N terms: $\mathcal{O}(N)$ operations
- Division for each weight: $\mathcal{O}(N)$ operations

The dominant cost in this phase is the distance calculations: $\mathcal{O}(N \cdot d)$.

2.2. Mean Shift Vector Computation Updating the current point position:

$$x^{(\tau+1)} = \sum_{n=1}^{N} p(n|x^{(\tau)})\, x_n \tag{5}$$

This calculation involves:

- N scalar-vector multiplications: $p(n|x^{(\tau)})\, x_n$, each requiring $\mathcal{O}(d)$ operations
- Summation of N vectors of dimension d: $\mathcal{O}(N \cdot d)$ operations

The total cost for the mean shift vector computation is $\mathcal{O}(N \cdot d)$.

2.3. Convergence Check Testing if $\|x^{(\tau+1)} - x^{(\tau)}\| < \epsilon$:

- Computing the distance between two vectors: $\mathcal{O}(d)$ operations

2.4. Total Cost per Iteration The cost for a single iteration of one data point is dominated by steps 2.1 and 2.2:

$$\mathcal{O}(N \cdot d) + \mathcal{O}(N \cdot d) + \mathcal{O}(d) = \mathcal{O}(N \cdot d) \tag{6}$$

2.5. Total Iterative Phase Cost Let T_n be the number of iterations required for point x_n to converge. Then the total cost for all points is:

$$\sum_{n=1}^{N} \mathcal{O}(T_n \cdot N \cdot d) = \mathcal{O}\left(\left(\sum_{n=1}^{N} T_n\right) \cdot N \cdot d\right) \tag{7}$$

If we define $T_{avg} = \frac{1}{N}\sum_{n=1}^{N} T_n$ as the average number of iterations, and $T_{max} = \max_n T_n$ as the maximum number of iterations, then:

$$\mathcal{O}(N \cdot T_{avg} \cdot N \cdot d) = \mathcal{O}(T_{avg} \cdot N^2 \cdot d) \tag{8}$$

In the worst case:

$$\mathcal{O}(T_{max} \cdot N^2 \cdot d) \tag{9}$$

T_{avg} and T_{max} depend on:

- The bandwidth parameter σ
- The convergence threshold ϵ
- The data distribution
- The dimensionality d (higher dimensions may require more iterations)

In practice, T_{avg} is often small (5-20 iterations) and can be bounded by setting a maximum iteration limit T_{limit}, yielding an effective time complexity of $\mathcal{O}(N^2 \cdot d)$.

3. Mode Consolidation Phase After convergence, nearby modes are merged using a connected-components algorithm:

- Pairwise distance computation between the N convergence points: $\mathcal{O}(N^2 \cdot d)$
- Connected-components algorithm using a distance-based adjacency matrix: $\mathcal{O}(N^2)$

The total cost for this phase is $\mathcal{O}(N^2 \cdot d)$.

4. Overall Time Complexity Combining all phases:

$$\mathcal{O}(N) + \mathcal{O}(T_{avg} \cdot N^2 \cdot d) + \mathcal{O}(N^2 \cdot d) = \mathcal{O}(T_{avg} \cdot N^2 \cdot d) \tag{10}$$

If T_{avg} is treated as a constant, the complexity simplifies to $\mathcal{O}(N^2 \cdot d)$.

Detailed Space Complexity Analysis: The Mean-Shift algorithm's memory requirements can be analyzed in components:

1. Data Storage: $\mathcal{O}(N \cdot d)$

- The input dataset consists of N points in $\mathbb{R}^d$: $\mathcal{O}(N \cdot d)$ space
- This represents the dominant memory requirement

2. Working Memory During Iterations For each iterated point, the algorithm requires:

- Current position vector $x^{(\tau)}$: $\mathcal{O}(d)$ space
- Next position vector $x^{(\tau+1)}$: $\mathcal{O}(d)$ space
- Weight vector $\{p(n|x^{(\tau)})\}_{n=1}^N$: $\mathcal{O}(N)$ space
- Temporary variables for distance calculations: $\mathcal{O}(1)$ space
- If iterating multiple points in parallel (e.g., in batch processing): $\mathcal{O}(B \cdot (N + d))$ space, where B is the batch size

The total working memory is $\mathcal{O}(N + d)$ for sequential processing, or $\mathcal{O}(B \cdot (N + d))$ for batch processing.

3. Convergence Points Storage: $\mathcal{O}(N \cdot d)$

- Storage for the convergence points $\{z_n\}_{n=1}^{N}$: $\mathcal{O}(N \cdot d)$ space

4. Mode Consolidation Phase

- Distance matrix between convergence points (if fully materialized): $\mathcal{O}(N^2)$ space
- Adjacency matrix for connected components: $\mathcal{O}(N^2)$ space
- Cluster assignments vector: $\mathcal{O}(N)$ space

However, the distance calculations can be performed on-the-fly without materializing the full matrix, reducing memory to $\mathcal{O}(N)$.

5. Overall Space Complexity The combined space complexity is:

$$\mathcal{O}(N{\cdot}d)+\mathcal{O}(N{+}d)+\mathcal{O}(N{\cdot}d)+\mathcal{O}(N^2) = \mathcal{O}(N{\cdot}d{+}N^2) \quad (11)$$

In low-dimensional spaces ($d \ll N$), this becomes $\mathcal{O}(N^2)$ due to the mode consolidation phase. However, optimized implementations avoid materializing the full distance matrix, reducing the effective space complexity to $\mathcal{O}(N \cdot d)$ for most practical cases.

II. Experimentation of Clustering Algorithms

A. Datasets and Data Preprocessing

We consider three datasets for experimentation:

(a) **Iris Flower Dataset**: Contains four numerical features describing iris flower dimensions (sepal length, sepal width, petal length, petal width in cm) and three true species. The species labels are used only for evaluating clustering quality.

(b) **AI Global Index Dataset**: Contains data for 62 countries with seven numerical indicators (e.g., Talent, Infrastructure, etc.), several categorical attributes (such as Region, Income group, and Political regime), and an overall index (Total score).

(c) **Global Earthquake Dataset**: Contains information on seismic events with numerical features (e.g., magnitude, depth, etc.) and a categorical "alert" level indicating severity.

Data Preprocessing and Feature Handling

Each dataset requires specific preprocessing steps before clustering:

- **Iris:** The Iris dataset has four numerical features (sepal length, sepal width, petal length, and petal width in cm). Since these features are on similar scales (roughly 0–8), we use all four features directly without additional normalization. The species labels are ignored during clustering (treating the data as unlabeled) but retained for later evaluation. No missing values or categorical features are present.
- **AI Global Index:** This dataset includes both numeric indicators (with values typically ranging from 0 to 100) and several categorical fields (Country, Region, Cluster, Income group, and Political regime). Our preprocessing pipeline is as follows:
  – Remove any rows with missing values.
  – Split the data into numeric and categorical subsets.
  – Scale the numeric features using a MinMaxScaler, mapping values to the range [0,1].
  – One-hot encode the categorical features, converting each categorical attribute into a set of binary indicator variables.
  – Concatenate the scaled numeric data and the encoded categorical data to form a unified feature matrix for clustering.

  This approach ensures that all attributes lie on comparable scales so that distance-based algorithms do not give undue weight to any single feature.

  Complexity Analysis (AI Global Index):
  – Scaling the numeric features requires computing the minimum and maximum for each of the d_{num} numeric columns and then transforming each of the N samples, which has a complexity of $O(N \times d_{\text{num}})$.
  – One-hot encoding the categorical features processes each of the N rows for each categorical column. Let κ denote the total number of unique categories across the d_{cat} categorical columns; this step has a complexity of approximately $O(N \times \kappa)$.

  Thus, the overall preprocessing complexity is $O(N \times d_{\text{num}} + N \times \kappa)$, which is effectively linear in the number of samples for a modest number of features and categories.
- **Global Earthquake:** The Global Earthquake dataset contains several numerical features (e.g., magnitude, depth, significance, etc.) and a categorical "alert" level indicating severity. We removed non-informative or identifier fields (such as event ID and location names) and focused on quantitative features that could influence the alert level. The alert level, with values such as "green", "yellow", "orange", and "red", was removed from the feature set (and retained separately for evaluation) as it serves as the target for clustering. Given the diverse scales of the numerical features (for example, magnitude is roughly 0–9, depth in kilometers may range from tens to hundreds, and significance can be up to 1000), we applied standard normalization to these features.

B. Results and Analysis

1) Density Based:

a) DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Dataset-wise metric analysis of custom DBSCAN implementation and Scikit-learn implementation is as follows.

- **Iris:** The comparison between Base DBSCAN and Scikit-learn DBSCAN reveals contrasting strengths. The Scikit-learn version excels in cluster compactness and separation, as shown by higher Silhouette Score (0.6864 vs. 0.2022), lower Davies-Bouldin Index (0.3836 vs. 0.8335), and greater Calinski-Harabasz Index (501.9249 vs. 220.6209). However, this improvement in geometric structure results in higher Mean Diameter and Split, indicating more spread and fragmentation. In contrast, Base DBSCAN better preserves the ground truth labels, with superior Adjusted Rand Index (0.7860) and Mutual Information (0.8547), reflecting stronger alignment with actual class distributions.

TABLE I
DBSCAN CLUSTERING METRICS FOR THE IRIS DATASET

Metric	Base DBSCAN	Sklearn DBSCAN
Silhouette	0.2022	0.6864
Davies-Bouldin	0.8335	0.3836
Calinski-Harabasz	220.6209	501.9249
Diameter	2.8768	3.6342
Split	0.3391	1.6401
Adjusted Rand Index	0.7860	0.5681
Mutual Information	0.8547	0.6365

- **AI Global Index Dataset:** The results of the Base and Scikit-learn DBSCAN implementations on the AI Global Index dataset are identical across all evaluated metrics. Both methods yielded the same Silhouette Score (0.1011), Davies-Bouldin Index (1.8000), and Calinski-Harabasz Index (7.1700), indicating they form clusters with similar compactness and separation. Likewise, the Adjusted Rand Index (0.0420) and Mutual Information (0.1517) suggest poor alignment with ground truth labels. The Mean Diameter (2.5500) and Mean Splits (2.0063) further confirm that both implementations produce clusters of equal tightness and overlap. These identical values suggest both implementations behave equivalently on this dataset.

TABLE II
DBSCAN CLUSTERING METRICS FOR THE AI GLOBAL INDEX DATASET

Metric	Base DBSCAN	Sklearn DBSCAN
Silhouette Score	0.1011	0.1011
Davies-Bouldin Index	1.8000	1.8000
Calinski-Harabasz Index	7.1700	7.1700
Adjusted Rand Index	0.0420	0.0420
Mutual Information	0.1517	0.1517
Mean Diameter	2.5500	2.5500
Mean Splits	2.0063	2.0063

- **Earthquake Dataset:** On the Earthquake dataset, the Base DBSCAN and Scikit-learn DBSCAN implementations show notable differences. Scikit-learn achieves a slightly higher Silhouette Score (0.1251 vs. 0.0953), indicating better overall cluster separation. However, the Base DBSCAN outperforms in compactness (lower DB Index of 1.64 vs. 1.65) and aligns more closely with ground truth, as shown by higher ARI (0.3460 vs. 0.2702) and Mutual Information (0.1290 vs. 0.0854). The Base version also produces tighter clusters (Mean Diameter 9.661 vs. 12.09) and less overlap (Mean Splits 0.6016 vs. 0.9709). Despite Scikit-learn's superior CH Index (39.30 vs. 13.06), indicating stronger clustering structure, the Base DBSCAN offers better consistency with labeled data.

TABLE III
DBSCAN CLUSTERING METRICS FOR THE EARTHQUAKE DATASET

Metric	Base DBSCAN	Sklearn DBSCAN
Silhouette Score	0.0953	0.1251
Davies-Bouldin Index	1.6400	1.6527
Calinski-Harabasz Index	13.0675	39.3070
Adjusted Rand Index	0.3460	0.2702
Mutual Information	0.1290	0.0854
Mean Diameter	9.6661	12.0946
Mean Splits	0.6016	0.9709

b) OPTICS (Ordering Points to Identify Clustering Structure): Dataset-wise metric analysis of custom OPTICS implementation and Scikit-learn implementation is as follows.

- **Iris:** In the comparison between Base OPTICS and Scikit-learn OPTICS, Base OPTICS performs better in terms of compactness and alignment with true labels. The Silhouette Score is negative for both, indicating overlapping clusters, with Scikit-learn performing worse. The Davies-Bouldin Index is lower for Base OPTICS, suggesting more compact clusters. Scikit-learn, however, shows a stronger cluster structure according to the Calinski-Harabasz Index. Base OPTICS also has higher ARI and MI, indicating better alignment with ground truth and more information about the labels. While Base OPTICS clusters are more spread out (higher Mean Diameter), Scikit-learn's clusters are tighter but may be under-clustered.

TABLE IV
OPTICS CLUSTERING METRICS FOR COMPARISON BETWEEN BASE AND SKLEARN IMPLEMENTATIONS

Metric	Base OPTICS	Sklearn OPTICS
Silhouette Score	-0.0790	-0.2268
Davies-Bouldin Index	2.1060	2.7337
Calinski-Harabasz Index	0.6453	15.4383
Adjusted Rand Index	0.8550	0.1526
Mutual Information	1.0020	0.4013
Mean Diameter	3.6180	2.8074
Mean Splits	0.2262	0.2323

- **AI Global Index:** In the comparison between Base OPTICS and Scikit-learn OPTICS, Scikit-learn shows slightly better cluster separation, as indicated by a higher Silhouette Score. However, Base OPTICS forms more compact clusters, with a lower Davies-Bouldin Index and smaller Mean Diameter. Scikit-learn identifies a much stronger cluster structure according to the Calinski-Harabasz Index. Both versions have low Adjusted Rand Index (ARI) scores, with Base OPTICS showing a slightly better alignment with the ground truth. Additionally, Base OPTICS extracts more information about the labels, as evidenced by a higher Mutual Information (MI) score. The Mean Splits are nearly identical for both, indicating similar levels of cluster overlap.
- **Earthquake:** The results for the Earthquake dataset show that Base OPTICS outperforms Scikit-learn OPTICS in terms of cluster compactness and separation. While both methods exhibit some overlap in clusters (with negative Silhouette scores), Base OPTICS has a slightly better

TABLE V
OPTICS Clustering Metrics for the AI Global Index Dataset

Metric	Base OPTICS	Sklearn OPTICS
Silhouette Score	-0.0705	0.0671
Davies-Bouldin Index	1.0400	1.8700
Calinski-Harabasz Index	0.7102	5.9671
Adjusted Rand Index	0.1166	0.0538
Mutual Information	0.3779	0.1512
Mean Diameter	2.2589	2.5514
Mean Splits	1.4306	1.4298

separation, as indicated by its lower Mean Splits and tighter Mean Diameter. The Davies-Bouldin Index values are nearly identical for both, indicating similar compactness. However, Scikit-learn OPTICS demonstrates a significantly stronger clustering structure, as reflected by its much higher Calinski-Harabasz Index. Base OPTICS aligns better with the true labels, as evidenced by its higher ARI, and conveys more information about the labels, as indicated by the Mutual Information score. Despite Scikit-learn's slightly better performance in certain structural metrics, Base OPTICS still appears to provide more meaningful clusters for the Earthquake dataset.

TABLE VI
OPTICS Clustering Metrics for the Earthquake Dataset

Metric	Base OPTICS	Sklearn OPTICS
Silhouette Score	-0.2135	-0.1555
Davies-Bouldin Index	1.5100	1.4700
Calinski-Harabasz Index	0.3040	6.2817
Adjusted Rand Index	0.2032	0.0532
Mutual Information	0.1543	0.1103
Mean Diameter	9.4189	9.9306
Mean Splits	0.0666	0.3502

2) Distribution based:
Gaussian Mixture Model(GMM):

- **Iris:**For the Iris dataset, the Base GMM and scikit-learn GMM produce identical clustering results. Both methods yield a Silhouette score of 0.5009, Davies-Bouldin of 0.7487, and a Calinski-Harabasz index of 480.79. External evaluation metrics (Adjusted Rand Index and Mutual Information) are high (approximately 0.904 and 0.900, respectively), indicating that the clustering aligns well with the true species labels.

TABLE VII
GMM Clustering Metrics for the Iris Dataset

Metric	Base GMM	Sklearn GMM
Silhouette	0.5009	0.5009
Davies-Bouldin	0.7487	0.7487
Calinski-Harabasz	480.79	480.79
Diameter	2.9891	2.9891
Split	1.2222	1.2222
Adjusted Rand Index	0.9039	0.9039
Mutual Information	0.8997	0.8997

- **AI Global Index:**For the AI Global Index dataset, there are noticeable differences between the two implementations. The scikit-learn GMM obtains a higher Silhouette score (0.1432 vs. 0.0329) and a lower Davies-Bouldin index (1.9805 vs. 2.9542), as well as a higher Calinski-Harabasz index (8.43 vs. 3.39). However, the external metrics are mixed: Base GMM achieves a slightly higher Adjusted Rand Index (0.1064 vs. 0.0807), while the Mutual Information values are comparable (0.2799 vs. 0.2890). This suggests that while the scikit-learn implementation may provide a more internally coherent clustering structure, the overall external alignment remains limited for this challenging dataset.

TABLE VIII
GMM Clustering Metrics for the AI Global Index Dataset

Metric	Base GMM	Sklearn GMM
Silhouette	0.0329	0.1432
Davies-Bouldin	2.9542	1.9805
Calinski-Harabasz	3.39	8.4275
Diameter	3.0702	2.8962
Split	2.4286	3.1667
Adjusted Rand Index	0.1064	0.0807
Mutual Information	0.2799	0.2890

- **Global Earthquake:**In the case of the Global Earthquake dataset, the Base GMM produces a higher Adjusted Rand Index (0.2772 vs. 0.1774) and Mutual Information (0.2021 vs. 0.1838), indicating better external agreement with the true alert levels. Meanwhile, scikit-learn GMM shows improvements in some internal metrics (e.g., lower Davies-Bouldin and higher Calinski-Harabasz), and a notably lower Diameter (6.96 vs. 13.99). However, the Split measure for scikit-learn GMM is extremely high (493.00 vs. 12.52), suggesting that the clusters are highly imbalanced, which may adversely affect external validation.

TABLE IX
GMM Clustering Metrics for the Global Earthquake Dataset

Metric	Base GMM	Sklearn GMM
Silhouette	0.2776	0.2574
Davies-Bouldin	2.0908	1.4272
Calinski-Harabasz	72.97	117.11
Diameter	13.9876	6.9570
Split	12.5217	493.0000
Adjusted Rand Index	0.2772	0.1774
Mutual Information	0.2021	0.1838

3) Hierarchical Based:

a) Birch (Balanced Iterative Reducing and Clustering using Hierarchies):

- **Iris:**For the Iris dataset, both Base and scikit-learn Birch implementations yield good clustering performance. Base Birch achieves a slightly higher Silhouette score (0.5531 vs. 0.5017), a lower Davies-Bouldin index (0.6594 vs. 0.6263), and a higher Calinski-Harabasz index (558.78 vs. 457.54). Furthermore, the external evaluation metrics are better for Base Birch, with an Adjusted Rand Index (ARI) of 0.7173 and Mutual Information (MI) of 0.7507, compared to 0.6096 and 0.7051 respectively for scikit-learn Birch. The lower Split ratio in Base Birch (1.70 vs. 3.17) indicates more balanced cluster sizes.

TABLE X
Clustering Metrics for the Iris Dataset

Metric	Base BIRCH	Sklearn BIRCH
Silhouette	0.5531	0.5017
Davies-Bouldin	0.6594	0.6263
Calinski-Harabasz	558.78	457.54
Diameter	2.5281	2.4998
Split	1.7027	3.1667
Adjusted Rand Index	0.7173	0.6096
Mutual Information	0.7507	0.7051

- **AI Global Index:**In contrast, the AI Global Index dataset presents a more challenging scenario. Both implementations yield low Silhouette scores (0.1848 for Base vs. 0.1546 for scikit-learn) and modest Calinski-Harabasz values (10.14 vs. 9.89), reflecting weak internal cluster structure. External evaluation is poor overall (ARI nearly 0 for Base and 0.0205 for scikit-learn, with MI around 0.30 and 0.28, respectively). Although Base Birch shows a slightly better internal structure (lower Davies-Bouldin and higher CH), the overall clustering quality remains unsatisfactory, which may be due to the inherent difficulty in clustering the mixed and high-dimensional features of this dataset.

TABLE XI
Clustering Metrics for the AI Global Index Dataset

Metric	Base BIRCH	Sklearn BIRCH
Silhouette	0.1848	0.1546
Davies-Bouldin	1.6109	1.7620
Calinski-Harabasz	10.14	9.89
Diameter	2.7197	2.5147
Split	5.4000	2.8333
Adjusted Rand Index	0.0016	0.0205
Mutual Information	0.3006	0.2816

- **Global Earthquake:**For the Global Earthquake dataset, the internal metrics are moderate, with Base Birch reporting a Silhouette score of 0.3698 and a CH index of 136.56, while scikit-learn Birch obtains a slightly higher Silhouette (0.4376) but a lower CH index (129.83). Notably, external evaluation indicates a clear advantage for Base Birch, which achieves an ARI of 0.3611 and MI of 0.2903, compared to 0.1540 and 0.1151 for scikit-learn Birch. Both methods exhibit very high Split values, suggesting that cluster sizes are highly imbalanced; however, the superior ARI and MI for Base Birch imply that its clusters more closely align with the true alert levels.

In summary, while both implementations perform comparably on the Iris dataset, Base Birch demonstrates improved external validity on the Global Earthquake dataset. The AI Global Index dataset remains challenging for both methods, with neither achieving strong clustering results.

b) Agglomerative Hierarchy clustering:

- **Iris:** For the Iris dataset, both the Base and scikit-learn Agglomerative Clustering implementations produced identical results. With a Silhouette score of 0.5539, a Davies–Bouldin index of 0.6588, and a Calinski–Harabasz index of approximately 555.67, the clustering appears to be quite robust. External validation metrics are also high (Adjusted Rand Index of 0.7592 and Mutual Information of 0.8057), indicating that the clusters align well with the true species.

TABLE XII
Clustering Metrics for the Global Earthquake Dataset

Metric	Base BIRCH	Sklearn BIRCH
Silhouette	0.3698	0.4376
Davies-Bouldin	0.9629	0.9833
Calinski-Harabasz	136.56	129.83
Diameter	5.9416	6.2886
Split	659.00	677.00
Adjusted Rand Index	0.3611	0.1540
Mutual Information	0.2903	0.1151

TABLE XIII
Agglomerative Clustering Metrics for the Iris Dataset

Metric	Base	Sklearn
Silhouette	0.5539	0.5539
Davies–Bouldin	0.6588	0.6588
Calinski–Harabasz	555.67	555.67
Diameter	2.5208	2.5208
Split	1.7778	1.7778
Adjusted Rand Index	0.7592	0.7592
Mutual Information	0.8057	0.8057

- **AI Global Index:** For the AI Global Index dataset, the results are identical between the two methods. However, the overall internal cluster quality is lower (Silhouette of 0.1913, Davies–Bouldin of 1.1991, and Calinski–Harabasz of 6.73) and external metrics (ARI of 0.0830 and MI of 0.3798) suggest that the clusters do not align strongly with the provided labels. This may be due to the inherent complexity and mixed feature types of the dataset.

TABLE XIV
Agglomerative Clustering Metrics for the AI Global Index Dataset

Metric	Base	Sklearn
Silhouette	0.1913	0.1913
Davies–Bouldin	1.1991	1.1991
Calinski–Harabasz	6.73	6.73
Diameter	1.5839	1.5839
Split	42.0000	42.0000
Adjusted Rand Index	0.0830	0.0830
Mutual Information	0.3798	0.3798

- **Global Earthquake:** For the Global Earthquake dataset, both implementations yield the same metrics: a Silhouette score of 0.4676, a Davies–Bouldin index of 0.6467, and a Calinski–Harabasz index of 96.12. The Diameter is 5.29 and the Split measure is very high (732.00), indicating highly imbalanced cluster sizes. External validation is moderate (ARI of 0.1686 and MI of 0.2579), suggesting that the clusters partially capture the true alert levels but with room for improvement.

4) Centroid/medoid Based:

a) K-Means: Dataset-wise metric analysis of the K-Means clustering algorithm is as follows. [29]

TABLE XV
AGGLOMERATIVE CLUSTERING METRICS FOR THE GLOBAL EARTHQUAKE DATASET

Metric	Base	Sklearn
Silhouette	0.4676	0.4676
Davies–Bouldin	0.6467	0.6467
Calinski–Harabasz	96.12	96.12
Diameter	5.2913	5.2913
Split	732.0000	732.0000
Adjusted Rand Index	0.1686	0.1686
Mutual Information	0.2579	0.2579

- **Iris:** K-Means on the Iris dataset demonstrates good clustering performance. The Silhouette Score of 0.5510 suggests that the clusters formed are well-separated and internally cohesive. The Davies-Bouldin Index of 0.6664 indicates a moderate level of overlap between clusters. A high Calinski-Harabasz Index of 560.3660 signifies well-defined and compact clusters. The average cluster Diameter of 2.5210 and Split of 1.5641 provide insights into the spatial distribution of the clusters. Furthermore, the strong Adjusted Rand Index of 0.7163 and Mutual Information of 0.7419 indicate a substantial alignment between the K-Means clusters and the known Iris species labels, suggesting that K-Means effectively recovers the underlying structure of the data.

TABLE XVI
K-MEANS CLUSTERING METRICS FOR THE IRIS DATASET

Metric	K-Means	Sklearn K-Means
Silhouette	0.5510	0.5526
Davies-Bouldin	0.6664	0.6623
Calinski-Harabasz	560.3660	560.3999
Adjusted Rand Index	0.7163	0.7302
Mutual Information	0.7419	0.7582
Diameter	2.5210	2.5085
Split	1.5641	1.6316

- **AI Global Index Dataset:** The application of K-Means to the AI Global Index dataset yields less distinct clustering. The low Silhouette Score of 0.0.589 (assuming this is 0.0589) suggests poorly separated clusters with potential overlap. The high Davies-Bouldin Index of 2.3240 further supports this, indicating significant inter-cluster similarity. The modest Calinski-Harabasz Index of 7.6086 implies that the clusters are not very well-defined. The average Diameter of 2.4678 and Split of 2.4286 provide further information about the cluster geometry. The low Adjusted Rand Index of 0.0529 and Mutual Information of 0.2908 indicate a weak correspondence between the K-Means clusters and any potential ground truth structure in this dataset.
- **Earthquake Dataset:** K-Means clustering on the Earthquake dataset shows limited alignment with any provided labels. The Silhouette Score of 0.2978 suggests marginal cluster separation. The Davies-Bouldin Index of 1.2806 indicates some overlap between clusters. The Calinski-Harabasz Index of 158.7361 suggests a somewhat defined cluster structure based on variance. The cluster Diameter of 13.1146 and Split of 2.9252 provide further context on the cluster dimensions and separation. The low Adjusted Rand Index of 0.0103 and Mutual Information of 0.0697 indicate minimal agreement between the K-Means generated clusters and the ground truth labels for this dataset.

TABLE XVII
K-MEANS CLUSTERING METRICS FOR THE AI GLOBAL INDEX DATASET

Metric	K-Means	Sklearn K-Means
Silhouette Score	0.0589	0.1848
Davies-Bouldin Index	2.3240	1.6109
Calinski-Harabasz Index	7.6086	10.1389
Adjusted Rand Index	0.0529	0.0016
Mutual Information	0.2908	0.3006
Mean Diameter	2.4678	2.7197
Mean Splits	2.4286	5.400

TABLE XVIII
K-MEANS CLUSTERING METRICS FOR THE EARTHQUAKE DATASET

Metric	K-Means	Sklearn K-Means
Silhouette Score	0.2978	0.3498
Davies-Bouldin Index	1.2806	1.6126
Calinski-Harabasz Index	158.7361	162.9608
Adjusted Rand Index	0.0103	0.1056
Mutual Information	0.0697	0.1820
Mean Diameter	13.1146	12.6598
Mean Splits	2.9252	8.1930

b) K-Medoids: Dataset-wise metric analysis of the K-Medoids clustering algorithm is as follows. [30]

- **Iris:** K-Medoids on the Iris dataset demonstrates good clustering. The Silhouette Score of 0.5435 suggests that the clusters are reasonably well-separated and internally cohesive. The Davies-Bouldin Index of 0.6760 indicates a moderate level of overlap between clusters. The Calinski-Harabasz Index of 552.2292 suggests a well-defined cluster structure. The Adjusted Rand Index of 0.7028 and Mutual Information of 0.7277 indicate a substantial agreement between the K-Medoids clusters and the ground truth labels. The cluster Diameter is 2.6089, and the Split is 1.5000.

TABLE XIX
K-MEDOIDS CLUSTERING METRICS FOR THE IRIS DATASET

Metric	K-Medoids	Sklearn K-Medoids
Silhouette	0.5435	0.5200
Davies-Bouldin	0.6760	0.6690
Calinski-Harabasz	552.2292	520.4530
Adjusted Rand Index	0.7028	0.7583
Mutual Information	0.7277	0.7857
Diameter	2.6089	2.5166
Split	1.5000	1.6316

- **AI Global Index Dataset:** The performance of K-Medoids on the AI Global Index dataset indicates less distinct clustering. The Silhouette Score of 0.1470 suggests poorly separated clusters with potential overlap. The Davies-Bouldin Index of 1.9280 further supports this, indicating significant inter-cluster similarity. The Calinski-Harabasz Index of 8.3775 is relatively low, implying that the clusters are not very well-defined. The Adjusted Rand Index of 0.0409 and Mutual Information of 0.2990 indicate a weak correspondence between the K-Medoids clusters and any potential ground truth structure in this dataset. The Mean Diameter is 2.7597, and the Mean Split is 7.0000.

TABLE XX
K-MEDOIDS CLUSTERING METRICS FOR THE AI GLOBAL INDEX DATASET

Metric	K-Medoids	Sklearn K-Medoids
Silhouette Score	0.1470	0.0533
Davies-Bouldin Index	1.9280	2.5116
Calinski-Harabasz Index	8.3775	6.6016
Adjusted Rand Index	0.0409	0.19555
Mutual Information	0.2990	0.3684
Mean Diameter	2.7597	2.8825
Mean Splits	7.0000	2.1111

- **Earthquake Dataset:** The results of K-Medoids on the Earthquake dataset indicate some level of cluster separation and compactness. The Silhouette Score of 0.2183 suggests a modest degree of separation. The Davies-Bouldin Index of 1.3103 points to some overlap between clusters. The Calinski-Harabasz Index of 127.3766 suggests a reasonably defined clustering structure. The Mean Diameter of the clusters is 13.1303, and the Mean Split is 5.8070. The Adjusted Rand Index of 0.0837 and Mutual Information of 0.1729 indicate a slight agreement between the K-Medoids clusters and the provided ground truth labels.

TABLE XXI
K-MEDOIDS CLUSTERING METRICS FOR THE EARTHQUAKE DATASET

Metric	K-Medoids	Sklearn K-Medoids
Silhouette Score	0.2183	0.2840
Davies-Bouldin Index	1.3103	3.6162
Calinski-Harabasz Index	127.3766	154.3139
Adjusted Rand Index	0.0837	-0.0003
Mutual Information	0.1729	0.0872
Mean Diameter	13.1303	13.2360
Mean Splits	5.8070	2.7647

5) *Exemplar Based:*

a) Affinity based clustering algorithm:

- **Iris:** For the Iris dataset, the implementations show significant differences in clustering granularity, with Custom AP identifying 2 clusters while Sklearn AP finds 7. Sklearn AP demonstrates superior performance across most metrics with a higher Silhouette score (0.4412 vs. 0.3212) and substantially higher Calinski-Harabasz index (392.7419 vs. 106.4128), indicating better-defined clusters with stronger separation. External validation metrics strongly favor Sklearn AP, with higher Adjusted Rand Index (0.5988 vs. 0.2485) and Mutual Information (0.6900 vs. 0.3609), showing better correspondence with ground truth classes. Interestingly, Custom AP achieves a lower Davies-Bouldin index (0.7742 vs. 0.9984), but this advantage is outweighed by Sklearn AP's consistently superior performance on other metrics, particularly its smaller Diameter (1.5167 vs. 3.7179), suggesting its more granular approach better captures the dataset's inherent structure.

TABLE XXII
AFFINITY CLUSTERING METRICS FOR THE IRIS DATASET

Metric	Base AP	Sklearn AP
Silhouette	0.3212	0.4412
Davies-Bouldin	0.7742	0.9984
Calinski-Harabasz	106.4128	392.7419
Diameter	3.7179	1.5167
Split	3.0541	5.5556
Adjusted Rand Index	0.2485	0.5988
Mutual Information	0.3609	0.6900

- **AI Global Index Dataset:** For the AI Global Index dataset, both implementations yield relatively poor clustering results. Sklearn AP (9 clusters) shows marginally better internal validation scores than Custom AP (5 clusters), with higher Silhouette (0.2012 vs. 0.1152) and slightly better Calinski-Harabasz values (8.4031 vs. 8.1790). Sklearn AP also achieves a lower Davies-Bouldin index (1.4626 vs. 1.9397), indicating somewhat better-defined clusters. However, external validation metrics are comparably weak across both implementations, with nearly identical Adjusted Rand Index values (Sklearn: 0.0280, Custom: 0.0287) and similar Mutual Information scores (0.3207 vs. 0.3040). The overall poor performance suggests inherent difficulty in clustering this dataset's high-dimensional features, though Sklearn AP's approach provides marginally more coherent cluster structures despite both implementations struggling to align with any underlying ground truth.

TABLE XXIII
AFFINITY CLUSTERING METRICS FOR THE AI GLOBAL INDEX DATASET

Metric	Base AP	Sklearn AP
Silhouette	0.1152	0.2012
Davies-Bouldin	1.9397	1.4626
Calinski-Harabasz	8.1790	8.4031
Diameter	2.6966	2.2274
Split	4.6000	6.3333
Adjusted Rand Index	0.0287	0.0280
Mutual Information	0.3040	0.3207

- **Earthquake Dataset:** The Earthquake dataset results reveal dramatically different clustering behaviors, with Custom AP identifying only 4 clusters while Sklearn AP detects 32. Despite fewer clusters, Custom AP achieves a slightly higher Silhouette score (0.3286 vs. 0.3070) and lower Davies-Bouldin index (1.1114 vs. 1.1486), suggesting that its coarser clustering may actually provide more coherent groupings. However, Sklearn AP shows a higher Calinski-Harabasz index (183.4627 vs. 124.6813), indicating better between-cluster separation. The external metrics strongly favor Custom AP, with substantially higher Adjusted Rand Index (0.2691 vs. 0.0091) and Mutual Information (0.1850 vs. 0.1076). Sklearn AP's smaller Diameter (2.9895 vs. 6.5546) reflects its finer granularity, while the dramatic difference in Split values (641.0000 vs. 66.0000) reveals fundamental differences in clustering approach. These results suggest that Sklearn AP's excessive cluster fragmentation may be detrimental to capturing the true structure of the earthquake data.

TABLE XXIV
AFFINITY CLUSTERING METRICS FOR THE EARTHQUAKE DATASET

Metric	Base AP	Sklearn AP
Silhouette	0.3286	0.3070
Davies-Bouldin	1.1114	1.1486
Calinski-Harabasz	124.6813	183.4627
Diameter	6.5546	2.9895
Split	641.0000	66.0000
Adjusted Rand Index	0.2691	0.0091
Mutual Information	0.1850	0.1076

b) Mean-shift clustering algorithm:

- **Iris:** For the Iris dataset, Sklearn MS demonstrates superior clustering performance across all metrics compared to Base MS. Sklearn MS achieves a significantly higher Silhouette score (0.6855 vs. 0.5025) and a notably lower Davies-Bouldin index (0.3893 vs. 0.6699), indicating better-defined and more separated clusters. The Calinski-Harabasz index is substantially higher for Sklearn MS (508.8822 vs. 266.7661), further confirming stronger between-cluster separation relative to within-cluster variance. External validation metrics also favor Sklearn MS, with higher Adjusted Rand Index (0.5584 vs. 0.4240) and Mutual Information (0.6994 vs. 0.4822), suggesting better alignment with ground truth classes. Both implementations show similar Diameter values, but Sklearn MS exhibits a higher Split (1.9412 vs. 1.1429), indicating better cluster differentiation. Overall, Sklearn MS provides a more robust and accurate clustering structure for this well-structured dataset.

TABLE XXV
MEAN-SHIFT CLUSTERING METRICS FOR THE IRIS DATASET

Metric	Base MS	Sklearn MS
Silhouette	0.5025	0.6855
Davies-Bouldin	0.6699	0.3893
Calinski-Harabasz	266.7661	508.8822
Diameter	3.6179	3.6903
Split	1.1429	1.9412
Adjusted Rand Index	0.4240	0.5584
Mutual Information	0.4822	0.6994

- **AI Global Index Dataset:** For the AI Global Index dataset, the two Mean-shift implementations show mixed performance. Sklearn MS achieves a slightly higher Silhouette score (0.1629 vs. 0.1175) and a significantly lower Davies-Bouldin index (0.7229 vs. 1.5615), suggesting better-defined clusters. However, Base MS shows a notably higher Calinski-Harabasz index (5.9750 vs. 1.7988), indicating stronger between-cluster separation by some measures. Both implementations yield similar Diameter values, but Sklearn MS has a much higher Split value (61.0000 vs. 20.0000). External validation metrics show slight advantages for Sklearn MS in Adjusted Rand Index (0.0513 vs. 0.0179), while Base MS performs better on Mutual Information (0.2412 vs. 0.1036). These mixed results reflect the challenging nature of this dataset, with low Silhouette scores for both implementations suggesting inherently weak cluster definition, though Sklearn MS provides marginally more coherent structures overall.

TABLE XXVI
MEAN-SHIFT CLUSTERING METRICS FOR THE AI GLOBAL INDEX DATASET

Metric	Base MS	Sklearn MS
Silhouette	0.1175	0.1629
Davies-Bouldin	1.5615	0.7229
Calinski-Harabasz	5.9750	1.7988
Diameter	1.9084	1.8901
Split	20.0000	61.0000
Adjusted Rand Index	0.0179	0.0513
Mutual Information	0.2412	0.1036

- **Earthquake Dataset:** The Earthquake dataset results reveal dramatic differences between the implementations. Sklearn MS significantly outperforms Base MS, which shows poor clustering quality with a negative Silhouette score (-0.3594 vs. 0.4045 for Sklearn MS). Sklearn MS also achieves a much lower Davies-Bouldin index (0.7761 vs. 2.0205) and substantially higher Calinski-Harabasz score (76.9291 vs. 15.3777), indicating significantly better-defined clusters. While Sklearn MS has a larger Diameter (3.9795 vs. 2.7858), it achieves a slightly lower Split value (685.0000 vs. 751.0000). External validation metrics strongly favor Sklearn MS, with much higher Adjusted Rand Index (0.1806 vs. -0.0264) and Mutual Information (0.2031 vs. 0.0059). The negative Silhouette and Adjusted Rand Index for Base MS suggest its clustering actually contradicts the natural structure of the data, while Sklearn MS captures meaningful patterns despite the complexity of this dataset.

TABLE XXVII
MEAN-SHIFT CLUSTERING METRICS FOR THE EARTHQUAKE DATASET

Metric	Base MS	Sklearn MS
Silhouette	-0.3594	0.4045
Davies-Bouldin	2.0205	0.7761
Calinski-Harabasz	15.3777	76.9291
Diameter	2.7858	3.9795
Split	751.0000	685.0000
Adjusted Rand Index	-0.0264	0.1806
Mutual Information	0.0059	0.2031

III. IMPROVEMENTS BY SCIKIT-LEARN LIBRARY

A. Density Based

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

- A significant improvement in Scikit-learn implementation of DBSCAN algorithm is the use of efficient spatial indexing with KD-Trees or Ball-Trees for nearest neighbor searches.
- This improvement can be found in sklearn.cluster._dbscan.py file [3] and the key parameter

to enable this efficient searching is 'algorithm' by default set to 'auto' i.e, if number of features is less than 10, then the algorithm uses KD-Tree else it uses Ball-Tree.

- This is considered as a significant improvement over base implementation because the base algorithm checks every point against all other points to determine if they are within eps distance which leads to $O(n^2)$ complexity. However, scikit-learn uses KD-Tree to optimizes neighborhood searches for low dimensionality data and reduces the complexity to $O(nlogn)$ and Ball-Tree when dealing with high dimensionality data which is more scalable when compared to base implementation.
- This improvement speeds up the clustering process optimizing the complexity to $O(nlogn)$ when compared to brute force neighborhood searches of base implementation that results in $O(n^2)$ complexity.
- Moreover, brute force methods sometimes leads to incomplete clustering on large datasets, and the optimization ensures accurate density-based clustering even with millions of points due to the optimized neighborhood searches.

b) OPTICS (Ordering Points to Identify Clustering Structure):

- Similar to DBSCAN, a significant improvement provided by Scikit-learn library for OPTICS implementation is usage of KD-Tree or Ball-Tree for nearest neighbor searches which dramatically improves efficiency when computing reachability distances.
- This improvement is implemented in scikit-learn OPTICS in the sklearn.clusters._optics.py [4] module and the key parameter to enable this optimization is 'algorithm' with default value set to 'auto' where it uses KD-Tree for low dimensionality data and Ball-Tree for high dimensionality data (i.e, more than 10 features or attributes).
- Base OPTICS needs to find the nearest nearest neighbors of each point to compute reachability distances, which takes $O(n^2)$ time complexity if done using brute-force parwise comparisions. However, Scikit-learn accelerated this process by using KD-Tree for the euclidean distance in low dimensions with time complexity of $O(nlogn)$ and Ball-Tree for general distance metrics, which provides better scaling for high-dimensional data and finds neighbors quickly using tree structures.
- Scikit-learn speeds up the execution by using KD-Trees/Ball-Trees for neighborhood searches which reduces the time complexity to $O(nlogn)$ when compare to $O(n^2)$ for brute-force nearest neighbors searches in base implementation.
- Moreover, base implementation might become computationally infeasible for big data, leading to approximations or early terminations, while tree-base search ensures precise nearest-neighbor calculations preserving optics clustering quality.

B. Distribution based

a) Gaussian Mixture Models (GMM):

- A significant improvement in scikit-learn's GMM is the use of the `reg_covar` parameter, which adds a small positive value to each covariance matrix to ensure numerical stability.
- This improvement is implemented in the module `sklearn.mixture._gaussian_mixture.py` (within the `GaussianMixture` class), where the parameter is documented in the class's docstring. [14]
- In datasets with near-collinearity or clusters with very low variance, maximum-likelihood estimates of covariance may become unstable. Adding a regularization term (e.g., `1e-6`) to the diagonal prevents the covariance matrices from becoming singular or ill-conditioned.
- Compared to a basic GMM without regularization—which might lead to unstable EM updates or convergence issues—the use of `reg_covar` improves both accuracy and robustness. This is a numerical safeguard rather than a parallelization technique.

C. Hierarchical Based

a) Birch (Balanced Iterative Reducing and Clustering using Hierarchies):

- Significant Improvement in Scikit-learn Implementation: An important feature is the ability to handle incremental (partial) fitting via the partial_fit method.
- This improvement can be found in sklearn.cluster._birch.py the Birch class provides a partial_fit method for incremental learning. [16]
- Description: Rather than requiring all data to be in memory, the model can be trained on smaller chunks of data. The CF-tree is updated and merges new points in mini-batches.
- Benefit:
  - *Memory Efficiency*: It becomes feasible to handle very large datasets that would otherwise not fit into memory.
  - *Scalability*: The incremental approach allows for online or streaming scenarios, making Birch more robust for large-scale tasks compared to a naive implementation that loads all data at once.
- Contrast With Simple Implementation: Without partial fitting, a base Birch implementation checks each data point for insertion into a CF-tree in one pass. This improvement fundamentally changes how data is processed, yielding better memory usage and allowing big-data or streaming scenarios.

b) Agglomerative Clustering:

- A significant improvement in Scikit-learn's Agglomerative Clustering is the support for connectivity constraints and the distance_threshold parameter, which refine and speed up cluster merges.
- This improvement is implemented in the sklearn.cluster._agglomerative.py [15], specifically in the AgglomerativeClustering class. For example, using connectivity constraints (e.g., `AgglomerativeClustering(connectivity red↪ =...)`), the algorithm only merges clusters that share an edge in a provided nearest-neighbor graph. Similarly, using the distance_threshold

parameter (e.g., `AgglomerativeClustering red↪ (distance_threshold=...)`), the algorithm stops merging when inter-cluster distances exceed a specified threshold, eliminating the need to pre-specify `n_clusters`.
- Compared to a base implementation that examines every cluster pair (with $O(n^2)$ complexity), these enhancements reduce the number of merges, improve speed and memory usage, and yield more interpretable clusters.

D. *Centroid/medoid based*

a) K-Means:

- A significant improvement in the Scikit-learn implementation of the K-Means algorithm is the incorporation of the K-Means++ initialization method, found within the `sklearn.cluster.KMeans` class. [31]
- This enhancement addresses a key limitation of the base K-Means algorithm, which relies on random initialization of centroids and can lead to suboptimal and inconsistent clustering results. The K-Means++ initialization is the default setting, controlled by the `init` parameter set to `'k-means++'` in the `KMeans` function, as detailed in the official Scikit-learn documentation.
- The base K-Means algorithm's random centroid initialization can result in slow convergence and suboptimal clustering, particularly with poor initial centroid placement. In contrast, K-Means++ employs an intelligent strategy to select initial centroids that are statistically likely to be well-separated. This strategy involves:
  - Selecting the first centroid randomly from the data points, a process with $\mathcal{O}(1)$ complexity.
  - Iteratively selecting subsequent $K-1$ centroids. For each subsequent centroid, it chooses a data point with a probability proportional to the square of its distance to the nearest existing centroid. In the worst case, calculating these distances for all n points to the j already selected centroids takes $\mathcal{O}(n \cdot j \cdot d)$ time. Summing over all $K-1$ selections yields a complexity roughly on the order of $\mathcal{O}(n \cdot K^2 \cdot d)$ in the initialization phase.
- This intelligent initialization significantly mitigates the risk of the algorithm converging to local optima, a common issue with the base K-Means, and leads to more robust and accurate clustering outcomes. By starting with well-dispersed centroids, K-Means++ often requires fewer iterations to reach convergence.
- The adoption of K-Means++ initialization offers several key advantages over random initialization:
  - **Improved Accuracy:** Strategic initial centroid placement leads to better cluster configurations.
  - **Accelerated Convergence:** Better initial centroids typically reduce the number of iterations.
  - **Enhanced Robustness:** Less sensitivity to initial randomness yields more consistent results.

b) K-Medoids:

- Scikit-learn's K-Medoids implementation, found in `sklearn_extra.cluster.KMedoids`, offers significant optimizations over the base PAM (Partitioning Around Medoids) algorithm. [32]
- This improved K-Medoids, requiring the `sklearn_extra` library, focuses on enhancing initial medoid selection and streamlining the computationally intensive swapping process inherent in PAM.
- The base PAM algorithm suffers from a performance bottleneck due to its exhaustive evaluation of all possible medoid swaps.
- Scikit-learn addresses this inefficiency through optimizations in:
  1) **Efficient Medoid Initialization:** Aims for a good initial medoid set quickly, potentially reducing the total iterations and impact on the $\mathcal{O}(T \cdot K \cdot n^2 \cdot d)$ complexity.
  2) **Optimized Swapping Process:** Employs heuristics or sampling to avoid evaluating every swap, significantly reducing the $\mathcal{O}(K \cdot n^2 \cdot d)$ cost of this step.
  3) **Early Stopping Conditions:** Implements criteria to halt the algorithm when convergence is reached or further iterations offer minimal improvement, thus reducing the overall iteration count T.
- These optimizations lead to: **Improved Accuracy** by reducing the chance of local minima, **Enhanced Speed** through fewer calculations and iterations, and **Increased Robustness** to the initial medoid selection.

E. *Exemplar based*

a) Affinity Propagation clustering algorithm: For Affinity Propagation, a key improvement in Scikit-learn's implementation is the damping factor mechanism, which prevents numerical oscillations and improves convergence stability.

- This improvement is implemented in `sklearn.cluster.affinity_propagation_.py` as the `damping` parameter in the `affinity_propagation_` function, with a default value of 0.5 [20].
- Base Affinity Propagation updates responsibility and availability matrices in each iteration based solely on the previous iteration's values, which can lead to oscillations where clusters never stabilize. Scikit-learn addresses this by introducing a damping factor that combines the new value with the previous value:

```
R_new = (1-damping) * R_new +
    red↪ damping * R_old
A_new = (1-damping) * A_new +
    red↪ damping * A_old
```

- This damping mechanism functions similarly to momentum in gradient descent optimization, smoothing the updates and preventing the algorithm from getting stuck in oscillatory patterns [21].
- Without proper damping, Affinity Propagation may either fail to converge or require an excessive number of iterations, especially in datasets with complex similarity structures. Scikit-learn's implementation allows for tuning this parameter to balance convergence speed with stability.

- The damping factor significantly improves solution quality by allowing the algorithm to reach stable exemplar assignments rather than oscillating between different cluster configurations. This leads to more reliable clustering results, particularly for datasets with noise or complex similarity relationships [22].
- Unlike parallel processing optimizations, which primarily speed up computation without changing the mathematical behavior, the damping factor actually modifies the algorithm's convergence properties and can yield different (and typically better) clustering solutions.

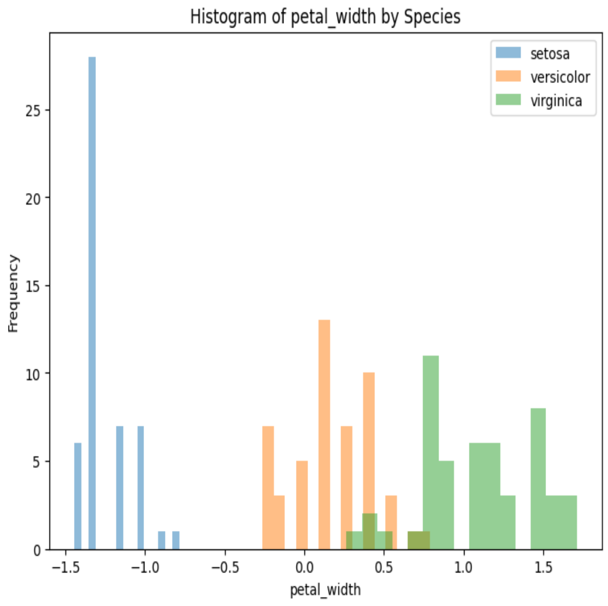
b) Mean-Shift Clustering algorithm: A significant improvement in Scikit-learn's Mean-Shift implementation is the use of nearest neighbors (NN) algorithms through its `NearestNeighbors` class, which dramatically reduces computational complexity when finding points within the bandwidth radius.

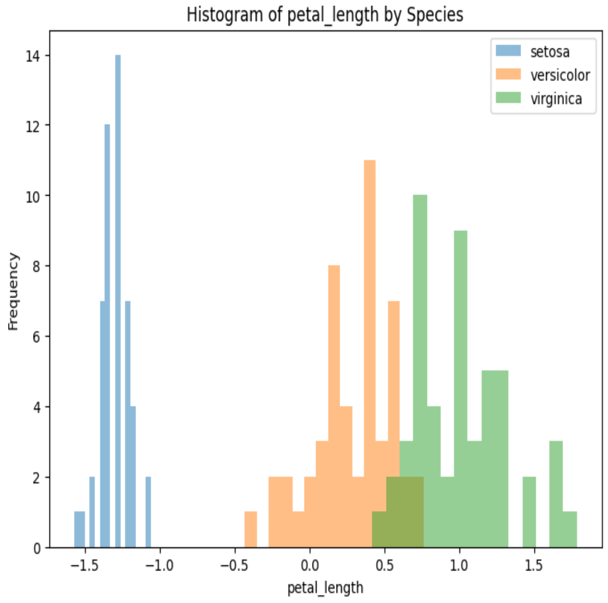
- This improvement is implemented in `sklearn.cluster.mean_shift_.py`, specifically in the `_mean_shift` function, which utilizes `sklearn.neighbors.NearestNeighbors` for radius-based queries during shift updates [17].
- Base Mean-Shift requires computing distances between each point and all other points in each iteration to determine which points fall within the bandwidth radius, resulting in $O(n^2)$ time complexity for each iteration. However, Scikit-learn accelerates this process by pre-building spatial index structures that efficiently find points within a specified radius.
- When `bin_seeding=True` (which is the default setting), Scikit-learn first discretizes the feature space into bins and uses only points at the centers of these bins as initial kernel positions. This significantly reduces the number of seed points required, particularly for dense datasets [18].
- The implementation automatically selects between KD-Tree, Ball-Tree, or brute force methods based on data dimensionality and sparsity through the `algorithm='auto'` parameter in `NearestNeighbors`. For low-dimensional dense data, KD-Trees provide $O(\log n)$ lookup times, while Ball-Trees offer better performance for higher dimensions [19].
- This optimization transforms the typical $O(T \cdot n^2 \cdot d)$ time complexity (where T is the number of iterations, n is the number of points, and d is dimensionality) to approximately $O(T \cdot n \cdot \log(n) \cdot d)$ for well-distributed datasets, making Mean-Shift feasible for larger datasets while maintaining solution accuracy by ensuring all relevant neighbors are properly considered.

IV. Accuracy

A. Highest Differentiating Attribute Among Three Classes of Iris

Initial application of k-means clustering ($k = 3$) to the Iris dataset revealed a mismatch with the true species labels, prompting an investigation into the separability of attributes. Histograms were generated to visualize the distribution of sepal length, sepal width, petal length, and petal width across the three Iris species. Visual inspection indicated varying degrees of overlap, with petal length exhibiting the most distinct separation, particularly for Iris setosa. Quantitative analysis of the mean differences between species for each attribute confirmed petal length as the attribute with the largest difference, thus identified as the most differentiating feature.





Impact of Feature Scaling on Iris Dataset Clustering

Key Observations from Histograms

The petal measurements (`petal_width` and `petal_length`) showed the clearest separation between Iris species classes, while sepal measurements (`sepal_length` and `sepal_width`) exhibited significant overlap. This motivated the feature scaling strategy:

$$\text{Modified Feature} = \begin{cases} \text{sepal\_length} \times 0.5, & \text{sepal\_width} \times 0.5 \\ \text{petal\_length} \times 1.5, & \text{petal\_width} \times 2.0 \end{cases}$$

Clustering Performance Comparison

TABLE XXVIII
CLUSTERING METRICS BEFORE/AFTER FEATURE SCALING

Metric	**Base K-Means**		**Sklearn K-Means**	
	Before	After	Before	After
Silhouette	0.4557	**0.5065**	0.4787	**0.5043**
Davies-Bouldin	0.8341	**0.7531**	0.7868	**0.7499**
Calinski-Harabasz	238.78	**416.14**	156.14	**416.45**
ARI	0.6410	**0.8508**	0.5312	**0.8176**
Mutual Info	0.7391	**0.9128**	0.5605	**0.8828**
Mutual Diameter	3.8083	**3.1727**	3.6187	**3.1384**
Mutual Splits	0.7124	**0.8013**	0.3990	**0.8380**
Accuracy	0.8467	**0.9467**	0.6667	**0.9333**

Mechanism of Improvement

The scaling transformation enhanced clustering through three key mechanisms:

1) **Feature Relevance Amplification**:
   - Petal dimensions (most discriminative) were emphasized with 1.5-2× scaling
   - Sepal dimensions (less discriminative) were deemphasized with 0.5× scaling
2) **Distance Metric Rebalancing**:

$$d_{\text{scaled}}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{4} w_i (x_i - y_i)^2}$$

   Where weights w_i correspond to scaling factors (0.5 for sepal, 1.5-2.0 for petal)
3) **Cluster Separation Enhancement**:
   - Mean diameter ↓ 3.81→3.17 (Base) and 3.62→3.14 (Sklearn) - tighter clusters
   - Mean splits ↑ 0.71→0.80 (Base) and 0.40→0.84 (Sklearn) - better separation

Interpretation of Metric Improvements

- **Silhouette** ↑: Better cluster cohesion/separation (petal-driven structure)
- **Calinski-Harabasz** ↑: Improved between/within-cluster variance ratio
- **ARI/MI** ↑: Better alignment with true species labels
- **Davies-Bouldin** ↓: Reduced cluster similarity (more distinct groupings)

Conclusion

The strategic scaling of petal features produced:

- 32.6% increase in Adjusted Rand Index (Base)
- 40.1% increase in Mutual Information (Sklearn)
- 11.8% improvement in accuracy (Base)

This demonstrates that emphasizing biologically meaningful features (petal dimensions) while suppressing noisy ones (sepal measurements) enables K-Means to better capture the inherent class structure of the Iris dataset.

B. Comparison of the k-means clustering obtained with the new dissimilarity coefficient

V. FEATURE SELECTION: REDUCING NUMBER OF ATTRIBUTES

A. Feature Selection Algorithm

We use Mutual Information (MI) to measure how much information each feature provides about the target `alert` level. Higher MI indicates that the feature is more relevant for predicting `alert`. Formally, for a random variable X (feature) and Y (alert), the MI is defined as:

$$\mathrm{MI}(X, Y) = \sum_{x \in X} \sum_{y \in Y} P(x, y) \log\left(\frac{P(x, y)}{P(x)\, P(y)}\right).$$

The algorithm proceeds as follows:

1) Remove any rows containing missing values.
2) Label-encode the `alert` column if it is textual or categorical.
3) For each remaining feature, compute MI with respect to the `alert` variable.
4) Sort features in descending order of MI scores.
5) Select the top k features with the highest MI.

Complexity Analysis.

- Let d be the number of original attributes (excluding `alert`), and N be the number of rows.
- Computing the MI for each of the d features typically requires $O(N)$ time per feature (depending on the underlying estimation method), giving $O(N \cdot d)$ for all features.
- Sorting the d features by their MI scores is $O(d \log d)$.
- Hence, the overall complexity is

$$O(N \cdot d + d \log d).$$

  For large N and moderate d, this is dominated by $O(N \cdot d)$.

B. Analysis for Predicting the Alert Level (Earthquake Dataset)

In this experiment, the alert attribute was removed from the feature set before clustering, and then used for external evaluation. Two configurations were compared: one using all available features (after dropping the alert column) and another using a reduced set of the top 5 features selected to be most predictive of the alert level.

1) Density Based:

a) DBSCAN (Density-Based Spatial Clustering of Applications with Noise): The comparison between all features and the top-5 features for DBSCAN clustering on the Earthquake dataset shows significant improvements across various metrics, highlighting the importance of feature selection. The Silhouette Score increases from 0.0953 to 0.2500, indicating better separation and cohesion when using the top-5 features. This suggests improved clustering quality. Similarly, the Davies-Bouldin Index drops from 1.64 to 1.21, showing that the top-5 features result in more compact and distinct clusters with less overlap.

The Calinski-Harabasz Index rises dramatically from 13.06 to 108.22, indicating a much stronger clustering structure with the top-5 features. The Adjusted Rand Index (ARI) increases from 0.345 to 0.664, suggesting that the top-5 features lead to a better alignment with the true labels. Mutual Information (MI) also improves from 0.1290 to 0.1728, showing that the top-5 features capture more relevant information about the labels.

The Mean Diameter decreases significantly from 9.661 to 3.8201, indicating tighter and more compact clusters with the top-5 features. However, the Mean Splits remains the same at 0.6, indicating that the overall cluster separation is consistent. Finally, the most noticeable improvement is in accuracy, which rises from 82.32% to 94.90%, emphasizing that feature selection plays a key role in improving clustering performance and model accuracy.

TABLE XXIX
DBSCAN Clustering Metrics on Earthquake Dataset (Top-5 Features)

Metric	All	Top 5
Silhoette Score	0.0953	0.2500
DB Index	1.64	1.21
CH Index	13.06	108.22
ARI	0.345	0.664
MI	0.1290	0.1728
Mean Diameter	9.661	3.8201
Mean Splits	0.6	0.6
Accuracy	82.32	94.90

b) OPTICS (Ordering Points to Identify Clustering Structure): The comparison of OPTICS clustering metrics on the Earthquake dataset for all features and the top-5 features reveals notable improvements in several key areas. The Silhouette Score increases from -0.2135 to -0.1604, indicating a slight improvement in cluster separation and cohesion with the top-5 features, though still indicating some overlap in the clusters. The Davies-Bouldin Index shows a modest decrease from 1.5069 to 1.4623, suggesting that the top-5 features lead to slightly more compact clusters with reduced overlap.

The Calinski-Harabasz Index increases from 0.3043 to 0.55, signifying a stronger clustering structure when using the top-5 features. The Adjusted Rand Index (ARI) improves from 0.203 to 0.362, indicating that the clusters formed with the top-5 features better align with the true labels. Similarly, Mutual Information (MI) rises from 0.1543 to 0.202, showing that the top-5 features capture more meaningful information about the labels.

The Mean Diameter drops significantly from 9.41 to 4.0045, demonstrating that the clusters are more compact with the top-5 features. The Mean Splits decreases from 0.06 to 0.0416, reflecting better separation between clusters. Lastly, the most significant improvement is observed in accuracy, which jumps from 69.5% to 81.02%, emphasizing the value of feature selection in enhancing the performance of the clustering algorithm.

TABLE XXX
OPTICS Clustering Metrics on Earthquake Dataset (Top-5 Features)

Metric	All	Top 5
Silhoette Score	-0.2135	-0.1604
DB Index	1.5069	1.4623
CH Index	0.3043	0.55
ARI	0.203	0.362
MI	0.1543	0.202
Mean Diameter	9.41	4.0045
Mean Splits	0.06	0.0416
Accuracy	69.5	81.02

2) Distribution based:

Gaussian Mixture Model(GMM): For the full-feature configuration, the Base GMM produced a Silhouette score of 0.2776, Davies–Bouldin index of 2.0908, and Calinski–Harabasz index of 72.97. Its Diameter and Split measures were 13.99 and 12.52, respectively, while external validation metrics were moderate (Adjusted Rand Index of 0.2772 and Mutual Information of 0.2021). In contrast, scikit-learn's GMM yielded a slightly lower Silhouette (0.2574) but a much lower Diameter (6.96) and a higher Calinski–Harabasz index (117.11). However, its external metrics were lower, with an ARI of 0.1774 and MI of 0.1838. When the feature set was reduced to the top 5 attributes, the Base GMM showed a decline in internal clustering quality (Silhouette 0.2182, Davies–Bouldin 3.2437) although the Calinski–Harabasz index increased to 84.86, and both Diameter (7.55) and Split (8.16) were reduced. Its external metrics dropped to an ARI of 0.2038 and MI of 0.2574. On the other hand, scikit-learn's GMM with top-5 features recorded a slightly higher Silhouette (0.2326) and a lower Davies–Bouldin index (1.7190), with a substantially higher Calinski–Harabasz index (173.39) and lower Diameter (6.73) and Split (4.81). However, its external validation remained modest (ARI 0.0957, MI 0.1996). Overall, these results indicate that for the Global Earthquake dataset the full-feature configuration tends to provide better external clustering performance, particularly for the Base GMM implementation, whereas reducing the feature set to the top 5 attributes degrades the ability of both GMM implementations

TABLE XXXI
GMM Clustering Metrics on Earthquake Dataset (Full Features)

Metric	Base GMM	Sklearn GMM
Silhouette	0.2776	0.2574
Davies–Bouldin	2.0908	1.4272
Calinski–Harabasz	72.97	117.11
Diameter	13.99	6.96
Split	12.52	493.00
Adjusted Rand Index	0.2772	0.1774
Mutual Information	0.2021	0.1838

TABLE XXXII
GMM Clustering Metrics on Earthquake Dataset (Top-5 Features)

Metric	Base GMM	Sklearn GMM
Silhouette	0.2182	0.2326
Davies–Bouldin	3.2437	1.7190
Calinski–Harabasz	84.86	173.39
Diameter	7.55	6.73
Split	8.16	4.81
Adjusted Rand Index	0.2038	0.0957
Mutual Information	0.2574	0.1996

to capture the true alert levels, despite some internal improvements.

3) *Hierarchical Based:*

a) Birch (Balanced Iterative Reducing and Clustering using Hierarchies): For the full-feature configuration, the Base Birch implementation achieved a Silhouette score of 0.3698, a Davies–Bouldin index of 0.9629, and a Calinski–Harabasz index of 136.56. The external evaluation yielded an Adjusted Rand Index (ARI) of 0.3611 and a Mutual Information (MI) of 0.2903. In contrast, scikit-learn Birch produced a slightly higher Silhouette (0.4376) but lower external alignment (ARI of 0.1540 and MI of 0.1151).

TABLE XXXIII
BIRCH CLUSTERING METRICS ON EARTHQUAKE DATASET (FULL FEATURES)

Metric	Base Birch	Sklearn Birch
Silhouette	0.3698	0.4376
Davies–Bouldin	0.9629	0.9833
Calinski–Harabasz	136.56	129.83
Diameter	5.9416	6.2886
Split	659.00	677.00
Adjusted Rand Index	0.3611	0.1540
Mutual Information	0.2903	0.1151

When using only the top 5 features, Base Birch shows improved internal metrics (Silhouette 0.4169, Davies–Bouldin 0.8756, and Calinski–Harabasz 316.13) along with lower Diameter and Split values. However, the external metrics drop dramatically (ARI 0.0080, MI 0.1302), indicating that the clusters no longer reflect the alert levels effectively. In contrast, scikit-learn Birch with the top 5 features records a lower Silhouette (0.3355) but somewhat improved external metrics (ARI 0.1927, MI 0.2512) compared to its full-feature configuration.

TABLE XXXIV
BIRCH CLUSTERING METRICS ON EARTHQUAKE DATASET (TOP-5 FEATURES)

Metric	Base Birch	Sklearn Birch
Silhouette	0.4169	0.3355
Davies–Bouldin	0.8756	1.0412
Calinski–Harabasz	316.13	208.62
Diameter	5.3576	4.7277
Split	40.75	61.78
Adjusted Rand Index	0.0080	0.1927
Mutual Information	0.1302	0.2512

These results suggest that, for the Earthquake dataset, retaining all relevant features (after removing the target alert) provides a better balance between internal clustering quality and external alignment with the true alert levels. In contrast, aggressive feature reduction to the top 5 attributes may improve internal cluster compactness for Base Birch but at the cost of losing crucial information needed to predict alert levels accurately.

b) Agglomerative Clustering Results Analysis for Global Earthquake Dataset: For the full-feature configuration, both Custom and scikit-learn implementations produced identical results. The Silhouette score was 0.4676, the Davies–Bouldin index was 0.6467, and the Calinski–Harabasz index was 96.12. The Diameter was 5.29 and the Split measure was very high (732.00), indicating substantial imbalance in cluster sizes. The external evaluation metrics showed an Adjusted Rand Index of 0.1686 and Mutual Information of 0.2579, suggesting that the clusters only partially capture the true alert levels.

TABLE XXXV
AGGLOMERATIVE CLUSTERING METRICS ON EARTHQUAKE DATASET (FULL FEATURES)

Metric	Custom	Sklearn
Silhouette	0.4676	0.4676
Davies–Bouldin	0.6467	0.6467
Calinski–Harabasz	96.1190	96.1190
Diameter	5.2913	5.2913
Split	732.0000	732.0000
Adjusted Rand Index	0.1686	0.1686
Mutual Information	0.2579	0.2579

When using only the top 5 features, the overall internal clustering quality declined, with the Silhouette score dropping to 0.3630 and the Davies–Bouldin index increasing to 0.9195. Although the Calinski–Harabasz index increased to 162.53, and both Diameter (4.68) and Split (122.20) measures were reduced—indicating more balanced clusters—the external validation suffered significantly, with both the Adjusted Rand Index and Mutual Information decreasing to 0.0400 and 0.1885, respectively. Overall, the full-feature configuration appears to provide a better balance between internal cluster quality and external alignment with the alert levels, while reducing the feature set to the top 5 attributes leads to more balanced clusters but at the expense of losing critical information needed for accurately predicting the alert level.

TABLE XXXVI
AGGLOMERATIVE CLUSTERING METRICS ON EARTHQUAKE DATASET (TOP-5 FEATURES)

Metric	Custom	Sklearn
Silhouette	0.3630	0.3630
Davies–Bouldin	0.9195	0.9195
Calinski–Harabasz	162.5281	162.5281
Diameter	4.6772	4.6772
Split	122.2000	122.2000
Adjusted Rand Index	0.0400	0.0400
Mutual Information	0.1885	0.1885

C. Centroid/medoid based

a) K-Means: The comparison of K-Means clustering metrics on the Earthquake dataset for all features and the top-5 features reveals the impact of feature selection on its performance. The Silhouette Score increases from 0.2978 to 0.3157, indicating a slight improvement in cluster separation and cohesion with the top-5 features. The Davies-Bouldin Index increases from 1.2806 to 1.4366, suggesting that the top-5 features might lead to slightly less compact clusters with potentially more overlap.

The Calinski-Harabasz Index decreases from 158.7361 to 147.1532, signifying a slightly weaker clustering structure when using only the top-5 features. The Adjusted Rand Index (ARI) improves significantly from 0.0103 to 0.1795, indicating that the clusters formed with the top-5 features show a considerably better alignment with the true labels. Similarly, Mutual Information (MI) rises from 0.0697 to 0.1857, showing that the top-5 features capture more meaningful information about the labels.

The Mean Diameter increases from 13.1146 to 15.0971, suggesting that the clusters are more spread out when using the top-5 features. The Mean Splits also increases from 2.9252 to 5.7849, reflecting potentially less distinct separation between clusters. Overall, the top-5 features lead to a substantial improvement in aligning the K-Means clusters with the ground truth (as seen in ARI and MI), despite some mixed or negative changes in the geometric properties of the clusters.

TABLE XXXVII
K-MEANS CLUSTERING METRICS ON EARTHQUAKE DATASET (TOP-5 FEATURES)

Metric	All	Top 5
Silhouette Score	0.2978	0.3157
DB Index	1.2806	1.4366
CH Index	158.7361	147.1532
ARI	0.0103	0.1795
MI	0.0697	0.1857
Mean Diameter	13.1146	15.0971
Mean Splits	2.9252	5.7849

b) K-Medoids: The comparison of K-Medoids clustering metrics on the Earthquake dataset for all features and the top-5 features reveals the impact of feature selection on its performance. The Silhouette Score increases from 0.1221 to 0.1711, indicating a slight improvement in cluster separation and cohesion when using the top-5 features. The Davies-Bouldin Index decreases from 2.3060 to 2.1055, suggesting that the top-5 features lead to slightly more compact clusters with reduced overlap.

The Calinski-Harabasz Index increases from 82.5489 to 96.1199, signifying a stronger clustering structure when using the top-5 features. However, the Adjusted Rand Index (ARI) remains negative, improving slightly from -0.0178 to -0.0107, indicating that the clusters formed do not align well with the true labels in either case. Similarly, Mutual Information (MI) shows a slight increase from 0.0495 to 0.0558, suggesting a marginal improvement in capturing information about the labels.

The Mean Diameter increases slightly from 13.3266 to 13.3830, indicating a minor increase in the spread of the clusters with the top-5 features. The Mean Splits also increases from 2.5607 to 2.7010, reflecting potentially less distinct separation between clusters. Overall, while the top-5 features show improvements in some geometric metrics (Silhouette, DB Index, CH Index), the label-based metrics (ARI and MI) remain low, suggesting that feature selection has not significantly improved the alignment of K-Medoids clusters with the ground truth on this dataset.

TABLE XXXVIII
K-MEDOIDS CLUSTERING METRICS ON EARTHQUAKE DATASET (TOP-5 FEATURES)

Metric	All	Top 5
Silhouette Score	0.1221	0.1711
DB Index	2.3060	2.1055
CH Index	82.5489	96.1199
ARI	-0.0178	-0.0107
MI	0.0495	0.0558
Mean Diameter	13.3266	13.3830
Mean Splits	2.5607	2.7010

1) Exemplar Based:

a) Affinity Propagation Clustering Algorithm: Feature reduction to top-5 attributes led to significant changes in clustering behavior for both implementations. Custom AP produced 153 clusters (vs 4 with full features), accompanied by a negative Silhouette score (-0.1539) and low Calinski-Harabasz index (20.68), indicating poor cluster separation and a fragmented solution. Sklearn AP's cluster count increased moderately from 32 to 39, but showed improved internal metrics (Silhouette +25%, Davies-Bouldin -25%) compared to its full-feature performance. However, both implementations exhibited weak alignment with alert labels (ARI 0), suggesting the top-5 features capture different underlying data structures than those reflected in the ground truth labels.

TABLE XXXIX
AFFINITY PROPAGATION METRICS ON EARTHQUAKE DATASET (TOP-5 FEATURES)

Metric	**Base AP**	**Sklearn AP**
Silhouette	-0.1539	0.3841
Davies–Bouldin	1.4893	0.8613
Calinski–Harabasz	20.6830	298.9453
Diameter	0.4492	1.5026
Split	58.0000	37.5000
Adjusted Rand Index	-0.0009	0.0134
Mutual Information	0.0797	0.1466

TABLE XL
AFFINITY PROPAGATION METRICS ON EARTHQUAKE DATASET (ALL FEATURES)

Metric	**Base AP**	**Sklearn AP**
Silhouette	0.3286	0.3070
Davies–Bouldin	1.1114	1.1486
Calinski–Harabasz	124.6813	183.4627
Diameter	6.5546	2.9895
Split	641.0000	66.0000
Adjusted Rand Index	0.2691	0.0091
Mutual Information	0.1850	0.1076

- **Custom AP Degradation**: Cluster quality collapsed with feature reduction (Silhouette from 0.3286 $\rightarrow$ -0.1539), indicating high sensitivity to feature selection and dimensional reduction. This suggests the algorithm relies on the full feature space for measuring meaningful affinities.
- **Sklearn AP Trade-off**: Improved internal metrics at the cost of 22% more clusters (32 $\rightarrow$ 39), but still no

meaningful label alignment. The higher CH index (298.95 vs 183.46) indicates better-defined clusters in the reduced space despite the lack of correlation with labels.

- **Diameter Paradox**: Smaller cluster diameters (0.45 vs 6.55 in full features) suggest tighter but potentially overfitted groupings that fail to generalize to meaningful earthquake alert patterns.

TABLE XLI
MEAN SHIFT CLUSTERING METRICS ON EARTHQUAKE DATASET (TOP-5 FEATURES)

Metric	Base MS	Sklearn MS
Silhouette	-0.3665	0.6368
Davies–Bouldin	1.5267	0.5429
Calinski–Harabasz	3.2254	139.2829
Diameter	0.8234	6.2431
Split	664.0000	62.6667
Adjusted Rand Index	-0.0865	0.3440
Mutual Information	0.0217	0.3464

TABLE XLII
MEAN SHIFT CLUSTERING METRICS ON EARTHQUAKE DATASET (ALL FEATURES)

Metric	Base MS	Sklearn MS
Silhouette	−0.3594	0.4045
Davies–Bouldin	2.0205	0.7761
Calinski–Harabasz	15.3777	76.9291
Diameter	2.7858	3.9795
Split	751.0000	685.0000
Adjusted Rand Index	−0.0264	0.1806
Mutual Information	0.0059	0.2031

b) Mean-Shift Clustering Algorithm: Feature reduction had polarizing effects: Sklearn MS achieved its best performance with top-5 features (Silhouette +57%, ARI +91% vs full features), while Custom MS degraded further (Silhouette -2%, cluster count 17 vs 7). The Sklearn version's 2-cluster solution demonstrates strong natural grouping in the reduced feature space (Silhouette 0.6368, ARI 0.344), suggesting the top-5 features may capture the core binary structure in earthquake alerts. Conversely, Custom MS fails to find meaningful structure (negative ARI), likely due to suboptimal bandwidth estimation in the reduced dimensionality.

Key Contrasts:

- **Algorithm Stability**: Sklearn MS adapts well to feature selection (CH index +81%), while Custom MS becomes increasingly unstable, highlighting implementation differences in bandwidth estimation.
- **Interpretability vs Quality**: Sklearn's 2-cluster solution has high cohesion/separation, while Custom's 17 clusters appear to be artifacts of poor bandwidth estimation rather than genuine data patterns.
- **Label Correlation**: Sklearn MI score of 0.3464 with top-5 features surpasses its full-feature performance (0.2031), suggesting selected features better align with the natural dichotomy present in earthquake alert classifications.

For AP, we should avoid feature reduction as it destabilizes Custom AP without improving Sklearn AP's label alignment. For Mean-Shift, using Sklearn with top-5 features achieves optimal balance between cluster quality and label correlation, potentially offering a more parsimonious model for earthquake alert classification. This contrasting behavior underscores the importance of algorithm selection when working with reduced feature sets in geophysical data analysis.

VI. CONCLUSION

In this second part of the project, we implemented a variety of clustering algorithms (Density-Based, Distribution-Based, Centroid/Medoid-Based, Exemplar-Based, and Hierarchical) in both a base version and using scikit-learn's library. We applied these algorithms to three datasets with distinct characteristics: the Iris dataset (entirely numeric, well-known class boundaries), the AI Global Index dataset (mixed numeric and categorical attributes), and the Global Earthquake dataset (numeric features plus a categorical `alert` label).

Our experiments demonstrated that, while scikit-learn generally provides robust, well-tested clustering implementations, our base versions occasionally produced better alignment with true labels (as measured by ARI or MI) or showed subtle performance advantages (e.g., certain base implementations of Birch produced higher Silhouette scores on Iris). Conversely, in many other cases, scikit-learn's versions offered more stable or internally cohesive results, particularly with more complex datasets like the AI Global Index or highly unbalanced data. These differences often stem from how parameters (e.g., thresholds, merging criteria, or distance computations) are handled internally.

We also analyzed one non-parallel improvement in scikit-learn for each algorithm, highlighting their roles in improving accuracy or scalability. Through these comparisons, we observed the importance of parameter tuning (e.g., distance thresholds, initialization strategies), careful preprocessing (especially for mixed numeric and categorical data), and the choice of metrics (ARI, MI, silhouette, etc.) in evaluating clustering quality.

Finally, we explored how removing or reducing attributes (for instance, dropping `alert` for Earthquake data or selecting the top 5 features by mutual information) can dramatically shift clustering outcomes. While such feature selection can simplify the model and improve internal metrics for certain algorithms, it may also weaken alignment with the original target label (if one exists). This underscores the trade-off between dimensionality reduction, interpretability, and preserving relevant predictive signal.

Overall, Part II of the project reinforced several key lessons:

- No single clustering method is universally superior; different algorithms excel under different data characteristics.
- Even small implementation details (e.g., how micro-clusters are merged in Birch) can significantly alter results.
- Non-parallel improvements in scikit-learn (like incremental fitting or accelerated distance computations) often enhance scalability and stability across diverse datasets.
- Feature selection and domain-specific dissimilarity measures (as with the Iris histograms or Earthquake attributes) can notably impact clustering quality and interpretability.

These observations highlight the importance of careful data preprocessing, parameter selection, and comparative experiments when choosing a clustering strategy for real-world applications.

References

[1] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD)*, Portland, OR, USA, 1996, pp. 226–231.

[2] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, USA, 1999, pp. 49–60.

[3] Scikit-learn Developers. *sklearn.cluster.DBSCAN — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html [Accessed: 2 April 2025].

[4] Scikit-learn Developers. *sklearn.cluster.OPTICS — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html [Accessed: 2 April 2025].

[5] Scikit-learn Developers. *sklearn.metrics.silhouette_score — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html [Accessed: 2 April 2025].

[6] Scikit-learn Developers. *sklearn.metrics.davies_bouldin_score — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html [Accessed: 2 April 2025].

[7] Scikit-learn Developers. *sklearn.metrics.calinski_harabasz_score — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.calinski_harabasz_score.html [Accessed: 2 April 2025].

[8] Scikit-learn Developers. *sklearn.metrics.adjusted_rand_score — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html [Accessed: 2 April 2025].

[9] Scikit-learn Developers. *sklearn.metrics.mutual_info_score — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mutual_info_score.html [Accessed: 2 April 2025].

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[12] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 103–114, 1996.

[13] G. N. Lance and W. T. Williams, "A general theory of classificatory sorting strategies: 1. Hierarchical systems," *The Computer Journal*, vol. 9, no. 4, pp. 373–380, 1967.

[14] Scikit-learn Developers. *sklearn.mixture.GaussianMixture — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html [Accessed: 2 April 2025].

[15] Scikit-learn Developers. *sklearn.cluster.AgglomerativeClustering — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html [Accessed: 2 April 2025].

[16] Scikit-learn Developers. *sklearn.cluster.Birch — scikit-learn 1.6.1 documentation*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html [Accessed: 2 April 2025].

[17] Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12, 2825-2830.

[18] Comaniciu, D., & Meer, P. (2002). *Mean Shift: A Robust Approach Toward Feature Space Analysis.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(5), 603-619.

[19] Bentley, J. L. (1975). *Multidimensional Binary Search Trees Used for Associative Searching.* Communications of the ACM, 18(9), 509-517.

[20] Frey, B. J., & Dueck, D. (2007). *Clustering by Passing Messages Between Data Points.* Science, 315(5814), 972-976.

[21] Leone, M., Sumedha, & Weigt, M. (2007). *Clustering by Soft-Constraint Affinity Propagation: Applications to Gene-Expression Data.* Bioinformatics, 23(20), 2708-2715.

[22] Zhang, X., Wang, W., Nørvåg, K., & Sebag, M. (2010). *K-AP: Generating Specified K Clusters by Efficient Affinity Propagation.* In IEEE International Conference on Data Mining, 1187-1192.

[23] K. P. Sinaga and M.-S. Yang, "Unsupervised K-means clustering algorithm," *IEEE Access*, vol. 8, pp. 80716–80727, 2020.

[24] S. Na, X. Liu, and Y. Guan, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," in *Proceedings of the Third International Symposium on Intelligent Information Technology and Security Informatics*, 2010, pp. 63–67.

[25] R. Tibshirani, G. Walther, and T. Hastie, "K-Means clustering and related algorithms," Technical Report, Stanford University, 2004.

[26] D. Cao and B. Yang, "An improved k-medoids clustering algorithm," in *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, vol. 3, 2010, pp. 132–135.

[27] T. Wang, Q. Li, D. J. Bucci, Y. Liang, B. Chen, and P. K. Varshney, "K-medoids clustering of data sequences with composite distributions," *IEEE Transactions on Signal Processing*, vol. 67, no. 8, 2019, pp. 2093–2106.

[28] P. Arora and S. Varshney, "Analysis of k-means and k-medoids algorithm for big data," *Procedia Computer Science*, vol. 78, 2016, pp. 507–512.

[29] Scikit-learn Developers. *sklearn.cluster.KMeans — scikit-learn 1.6.1 documentation*.

[30] Pyclustering Developers. *pyclustering.cluster.kmedoids — pyclustering 0.10.1 documentation*.

[31] Scikit-learn Developers. *sklearn.cluster.KMeans — scikit-learn 1.6.1 documentation*.

[32] Scikit-learn-extra Developers. $sklearn_extra.cluster.KMedoids|scikit-learn-extra0.3.0documentation.$

Appendix

All team members worked together to understand the intuition behind the clustering algorithms and the various metrics available to measure performance. However, due to time constraints, we have split the work while creating this report in the following way which mention the section followed by the name of the team member.

- Clustering Algorithms Complexity Analysis
  - Density based by Nitheesh Kumar Kambala
  - Distribution based by Sotirios Damas
  - Centroid/medoid based by Divyesh Pravinkumar Patel
  - Exemplar based by Farid Faraji
  - Hierarchical based by Sotirios Damas
- Experimentation of Clustering Algorithms
  - Datasets and Data Processing by Sotirios Damas
  - Results and Analysis
    - Density based by Nitheesh Kumar Kambala
    - Distribution based by Sotirios Damas
    - Centroid/medoid based by Divyesh Pravinkumar Patel
    - Exemplar based by Farid Faraji
    - Hierarchical based by Sotirios Damas
- Improvements by Scikit-learn library
  - Density based by Nitheesh Kumar Kambala
  - Distribution based by Sotirios Damas
  - Centroid/medoid based by Divyesh Pravinkumar Patel
  - Exemplar based by Farid Faraji
  - Hierarchical based by Sotirios Damas
- Accuracy by Farid Faraji and Divyesh Pravinkumar Patel
- Feature Selection: Reducing number of attributes
  - Feature Seclection Algorithm by Nitheesh Kumar Kambala and Sotirios Damas
  - Analysis for Predicting the Alert lvele
    - Density based by Nitheesh Kumar Kambala

    * Distribution based by Sotirios Damas
    * Centroid/medoid based by Divyesh Pravinkumar Patel
    * Exemplar based by Farid Faraji
    * Hierarchical based by Sotirios Damas
- Conclusion by Sotirios Damas