# Effective Clustering for Large Datasets using Density-Based Clustering via Message Passing

Siddharth Dixit
Independent Researcher

## ABSTRACT
Density-based clustering remains a significant area of research in data science, particularly given the increasing prevalence of high-dimensional datasets with varying densities. Many existing clustering approaches struggle to effectively handle datasets that contain regions of high density surrounded by sparse areas. This study introduces a novel clustering algorithm based on the concept of mutual K-nearest neighbor relationships, designed to overcome these limitations. The proposed method requires only a single input parameter, demonstrates strong performance on high-dimensional, density-based datasets, and is computationally efficient. Furthermore, the algorithm's practical applications are illustrated through its potential to enhance search and retrieval processes within vector databases.

## Keywords
Clustering; Mutual *k*-Nearest Neighbor; Density- Based Methods; Outlier Detection; Vector Databases; Data Mining.

## 1. INTRODUCTION
Cluster analysis is one of the most important areas of data mining and is being used in a plethora of applications across the world [1, 2]. The motivation behind cluster analysis is to divide the data into groups called clusters that are not only useful but also meaningful. Every member of a group is similar to one another and dissimilar to members belonging to other groups. The entire collection of the groups of similar points is referred to as clustering [3]. In the modern era, clustering emerges as a powerful technique for identifying outliers within vector embedding datasets, owing to its foundational design principles. Its advantages are particularly notable in this context, including the ability to detect clusters of arbitrary shapes, its flexibility in operating without predefined cluster counts, and, most crucially, its inherent capability to isolate noise points—effectively pinpointing outliers.

In this paper, the primary focus is on clustering based on varying density. A clustering algorithm was developed that finds clusters in the region of high densities and low densities in datasets. The clustering algorithm uses a mutual k-nearest neighbor message- passing mechanism to find mutual relationships between points and form clusters [4, 5]. Experimental results demonstrate that the proposed clustering algorithm outperforms state-of-the-art methods across datasets with varying densities and high-dimensional features. [6, 7].

**Motivation**

### 1.1 Drawbacks of existing density based clustering
**Figure 1** shows a density-based dataset that has high-density point and low-density points. The traditional clustering algorithms like DBSCAN will be able to identify regions of high-density such as regions A and C but will neglect the regions of low-density points in region B as outliers. In these types of datasets, traditional algorithms will not be able to achieve complete clustering. In certain situations, the data points in region B might be really useful and might have meaningful relationships between them. Accordingly, a mechanism is required to effectively cluster datasets by identifying and associating both low- and high-density points with their respective groupings.

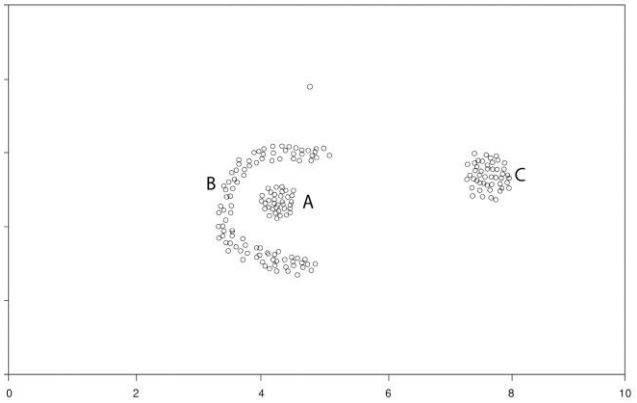



**Figure 1: Sample dataset with two dense regions and a sparse region.**

### 1.2 Limitation of parameter tuning
Existing density-based clustering algorithms like K-Means and DBSCAN are heavily dependent on input parameters. For example, DBSCAN requires two initial input parameters, namely Epsilon radius and minimum points which both have a significant influence on the clustering results. To get good clustering results from DBSCAN, there is a need to have well-trained set of Epsilon and MinPts values. To illustrate our point, DBSCAN was run on a synthetic dataset with a dense region surrounded by a sparse region. The input parameters were varied and ran DBSCAN for multiple iterations. It was observed with different input values the clustering results not only varied but also disregarded several points as outliers that were located in low-density regions **Figure 2**.

Similarly, K-means algorithm is highly dependent on the initial selection of centroids and the number of clusters k. The value of k is hard to guess, and there is no way to find out how many clusters will be appropriate. When the data is two-dimensional, it is fairly easy to identify the value of k through visual inspection but for higher dimensions it becomes nearly impossible. Thus to determine the input parameters a priori, multiple iterations with different input values are run to generate the optimal set of clusters. This process can be extremely time-consuming with larger data size. Therefore, there is critical need for a mechanism that is not dependent completely on input parameters.

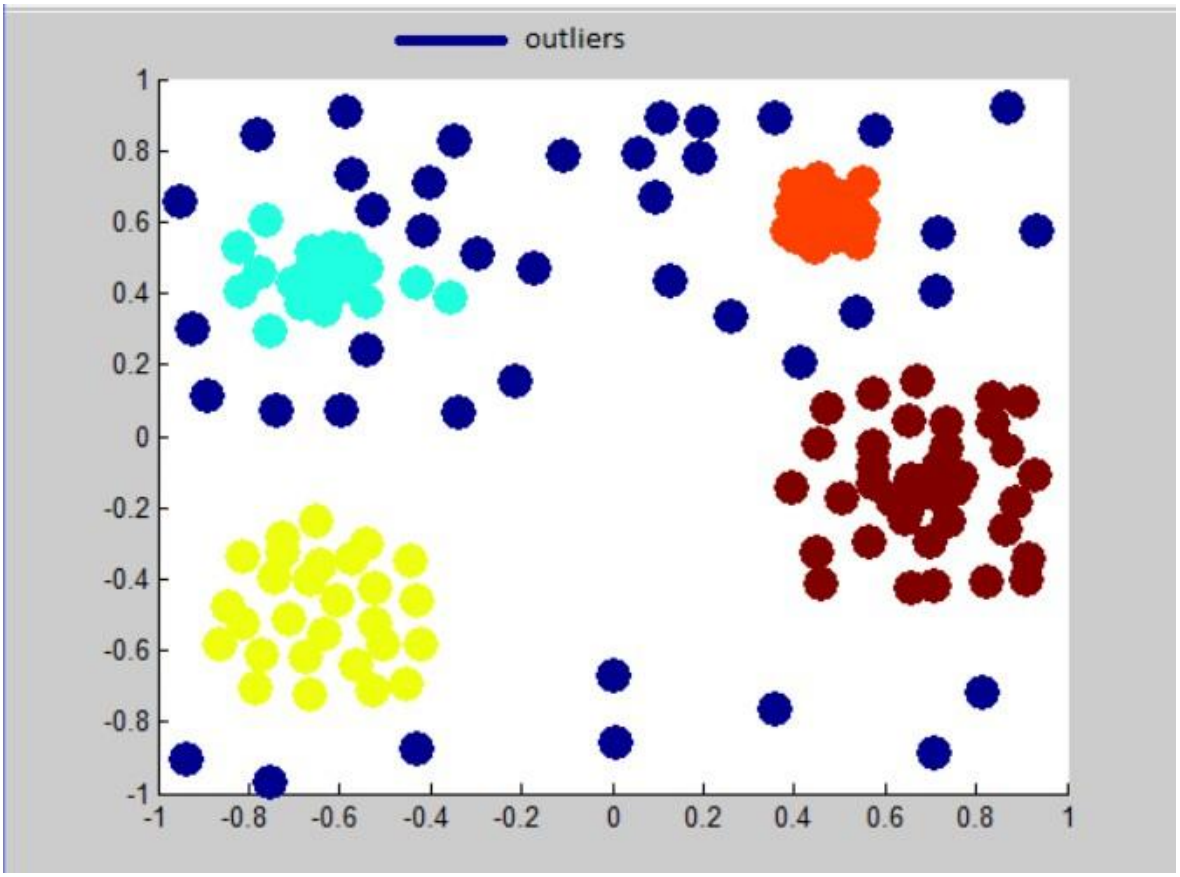



**Figure 2: DBSCAN on synthetic dataset considers several points as outliers (blue points).**

## 1.3 Drawbacks of neighbor relationship approach to clustering

Most of the existing algorithms for density based clustering such as Shared nearest neighbor or k-nearest neighbor considers closeness ‖ of a relationship from given point to all its neighbors and does not consider the other points perception of closeness with the given point. In k-nearest neighbor, every point tries to find its k-nearest neighbors and forms a cluster. However, this notion does not work well for datasets with varying density. A point in a region can see 10 points as its 10- nearest neighbor, but some of those points might be closer to other regions and might be ideally suited to be a part of that region. So there is a need to develop a method to overcome this problem by considering reciprocal relationships instead of one way nearest neighboring relationship of points.

## 1.4 Benefits of the proposed approach

In this paper, it is demonstrated that proposed approach works well with density based datasets and can clearly identify dense clusters and sparse clusters [8, 9]. The performance of our algorithm is compared with the standard clustering algorithms and show that our algorithm performance is significantly better for high-dimensional density- based datasets [1, 10]. Moreover, it requires only a single input parameter that is the k value that represents the maximum number of mutual neighbors a point can have [4]. Unlike most clustering algorithms, the user does not give the number of clusters as an input parameter, so it not only helps in generating natural clusters but also does not depend on a user to guess the ideal number of clusters beforehand [3, 11].

Furthermore, it is demonstrated with multiple runs of our algorithm on experimental data that varying k values slightly does not have a significant impact on clustering [4]. The clustering is based on a novel concept of mutual k-nearest neighbors and identifies clusters with high density and removes them from further consideration and then finds clusters in the regions where the density is sparse [4, 5].

Furthermore, the process of finding mutual neighbors is enhanced by message passing between data points. The approach does not require scanning the entire table of pairwise distances. Every point tries to find its mutual neighbor by communicating with its k-nearest points [11]. Once a point receives its mutual k-nearest neighbors, it excludes itself from further consideration thereby reducing the size of data during the run-time [4]. The merging process is implemented that uses a similar approach where the goal is to find best mutual neighbors of each cluster and merge them one at a time till the required number of clusters is obtained [4, 5]. This approach does not require comparing every cluster with another cluster to merge, and hence is a more efficient approach than related works

Point 0 has two shared neighbors with point 1 which is 2 and 3. So points 0 and 1 has an edge weight of 2 that represents 2-shared neighbors between them. Also, high strength points are decided by total sum of edges coming out of a point. For example, Point 0 has 3 edges with weights 2 each so the total strength is 6. The algorithm of SNN is described in **Figure 3**. However SNN is based on K-nearest neighbor relationship that is a one-way relationship. In Figure 6 0 and 1 are having high similarity based on shared neared neighbor concept but it does not consider if both 0 and 1 select each other as their neighbor or not.

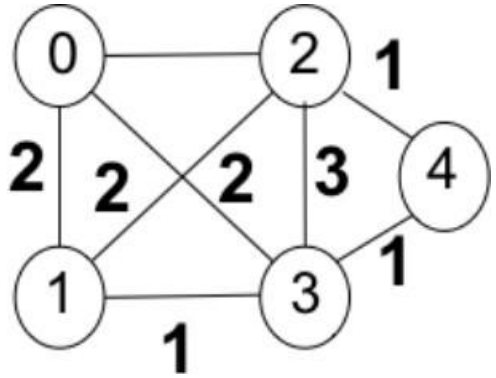



**Figure 3: A Graph Representation for a Dataset. [6, 7].**

# 2. RELATED WORK

## 2.1 DBSCAN

DBSCAN [1] overcomes problems of k-means on non-globular clusters as it can detect clusters of any arbitrary shape and is resistant to noise [4], [12]. DBSCAN uses two parameters for clustering:

- $\varepsilon$-**Neighborhood:** represents points within a radius of $\varepsilon$ from point p.
- **MinPts:** Minimum points in the $\varepsilon$-Neighborhood of p.

The algorithm is described in **Figure 3**. The basic idea of the algorithm is to form clusters that have at least MinPts within the $\varepsilon$-Neighborhood. The points that do not fall in the $\varepsilon$-Neighborhood are disregarded as noise points. For uniform datasets, DBSCAN can detect dense regions and generates clusters of various shapes and sizes based on density. However, if the density is varying DBSCAN has trouble detecting density and it can mark several points as noise. It is shown experimentally that DBSCAN is unable to detect the correct set of clusters for a synthetic dataset with dense and sparse regions. Also, it is experimentally demonstrated that DBSCAN is heavily dependent on the selection of input parameters. Varying input parameters even slightly causes a prominent change in clustering. Furthermore, for higher dimensional dataset it becomes even more complicated to find out the correct set of MinPts and epsilon radius, as it is hard to visualize unlike two-dimensional datasets. Another drawback is DBSCAN only considers a point's closeness with other points in its radius but does not consider other point's closeness with that point.

## 2.2 Shared Nearest Neighbor (SNN)

Shared Nearest Neighbor [6] overcomes the problem of clustering higher dimensional data by using the concept of shared neighbors. Shared neighbor is a pairwise relationship of points which is the number of neighbors two points have in common. For example if a point A has neighbors C and D and B has neighbors C and D, shared nearest neighbor considers A and B to be similar based on the common shared neighbors C and D. To explain the concept even further Shared Nearest Neighbors can be represented by a graph where vertex represents the points and edge represents the neighbor of a point.

## 2.3 Mutual K-nearest neighbor

Hu and Bhatnagar (2011) proposed a clustering algorithm for finding the mutual relationship between points using mutual rela- tionships. The algorithm requires pairwise distance relationship calculation for every data point and comparing every point with another point to find mutual neighbors [4, 12]. The clustering requires sorting of pairwise distances in ascending order and reading the table of sorted pairwise distances one at a time to find mutual neighbors. For example, if the size of the dataset has 10,000 points, the algorithm will require sorting of 100 million points and reading a table of 100 million rows to calculate mutual neighboring relationships. If the data size is kept increasing, it will not be feasible to store and process the pairwise distances in memory and will also require quadratic time to run the algorithm completely [10, 13].

In addition, the clustering step proposed by Hu and Bhatnagar (2011) [12] requires comparison of every cluster with each other and is CPU time intensive if initial number of clusters generated is very large. Thus, an efficient mechanism is required not only for identifying mutual nearest neighbors but also for performing cluster merging operations without the need to scan every cluster in the set [5, 6]. In this paper, the notion of mutual k- nearest neighbor relationship proposed by Zhen and Bhatnagar (2011) [4] is extended and an efficient mechanism of finding mutual neighbors by message passing and forming clusters is demonstrated. Furthermore, it is experimentally shown that the proposed algorithm is better in terms of time and space complexity and works as accurately as the algorithm proposed by previous work [1, 11].

# 3. OUR APPROACH

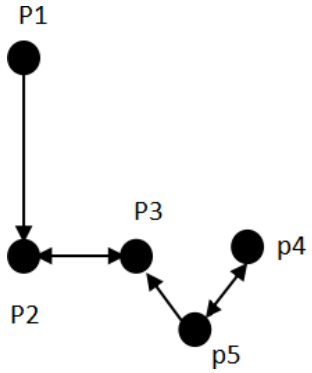
## 3.1 Mutual k-Nearest Algorithm with Message Passing

This Section deals in detail the design and implementation of Mutual k-Nearest Algorithm. The algorithm is based on the notion of mutual k-nearest neighbor relationships between data points and uses an efficient message passing system to figure out two-way nearest neighboring relationships. Most of the existing density based approaches like K-nearest neighbor and Shared nearest neighbor uses one way nearest neighboring relationship. For example, if a point $p_1$ selects $p_2$ as its nearest neighbor, it does not consider the relationship of $p_2$ with $p_1$. Mutual k-Nearest Neighbor relationship, on the other hand, considers reciprocal relationship as well i.e. $p_1$ and $p_2$ can only become a Mutual k-Nearest Neighbor pair if both $p_1$ has $p_2$ and $p_2$ has $p_1$themselves as their nearest neighbor. To formalize the definition of Mutual k-Nearest Neighbor relationship: Definition I: A point

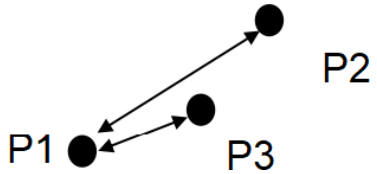
$p_1$ and $p_2$ with distance $d_{p_{12}}$ are Mutual k-Nearest Neighbors if

1. There are points fewer than k in distance space $d_{p_{12}}$
2. There are points more than k in distance space $d_{p_{12}}$, but most of them have already found their Mutual k-Nearest Neighbors Figure 4 illustrates the concept further.

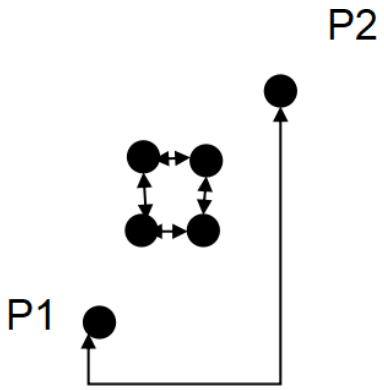
**Figure 4**. Sample points to explain mutual $k$-Nearest relationship. **Figure 4c**, **Figure 4b** Sample points to describe Definition I for $k = 2$. In **Figure 4a**, the single arrows represent nearest neighbor relationship and bi-directional arrows represent mutual relationships between the points. It can be seen that $P_1$ has $P_2$ as its neighbor in its Euclidean space but the same is not true with $P_2$. Similarly, $P_5$ has $P_3$ as its neighbor but $P_3$ has already selected $P_2$ as its mutual nearest neighbor. **Figure 4b** illustrates the first part of Definition I where $P_1$ has less than 2 points between $P$ So $P_1$ considers $P_2$ as its mutual neighbor.**Figure 4c** illustrates the second part of Definition I where $P_1$ and $P_2$ have more than 2 points between them so it considers $P_2$ as its mutual neighbor.




**(a)**
**Sample points to explain mutual k-Nearest relationship.**




**(b)**
**Sample points to describe definition I for k=2.**




**(c)**
**Sample points to describe definition I for k=2.**

**Figure 4: *k*-Nearest relationship**

## 3.2 Finding Mutual Neighbors by Message Passing Approach

It is illustrated with a simple example how message passing works to find mutual relationships for every point. Suppose there are two-dimensional data points as shown in the **Figure 5**. To explain the working of message passing algorithm considering value of K as 2.

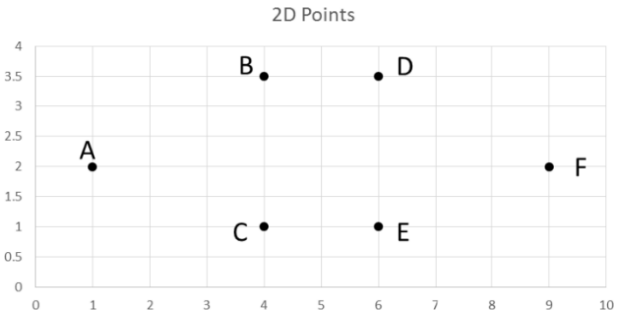



**Figure 5: A 2D Data Example to Explain M-kNN Algorithm with k=2 as Input.**

Message passing is an iterative process where every point sends request messages to nearest *k* points. The closeness of a point I determined by a pre-selected distance measure. The messages that are received by a particular point is a response message. Based on the messages sent and received, the mutual *k*-Nearest points are determined for every point. If a point $p$ receives messages from the same set of points $\{p_1, p_2, p_3, \ldots, p_n\}$ that it sent requesmessages to, for multiple iterations, then $p$ adds all the points into its mutual relationship. Since the relationship is mutual, every point in the set $\{p_1, p_2, p_3, \ldots, p_n\}$ also adds $p$ as mutual

*k*-nearest neighbor. In the message passing procedure, every point has its individual *k* value. Initially, all the points have same

*k*-value as supplied as an input. Table I shows message passing of points in **Table 1** in the first iteration. Initially, the *k* value is set to 2 for all points.

**Table 1: Message Passing First Iteration.**

| Points | Request Sent | Response Received | k | Candidate Mutual k-NN |
|---|---|---|---|---|
| A | B, C | – | 2 | – |
| B | C, D | A, C, D | 2 | C, D |
| C | B, E | A, B, E | 2 | B, E |
| D | B, E | B, E | 2 | B, E |
| E | C, D | C, D | 2 | C, D |
| F | C, E | – | 2 | – |

Every point sends messages to *k* points that are close to a particular point in terms of distance measure. For this example, let us consider the distance measure to be Euclidean. Based on points received, there are three conditions:

**Table 1** shows working of message passing in first iteration. Point B receives messages from points A, C and D but selects points C and D as mutual k-nearest neighbor candidate as they are closer to point B than point A. Similarly, point F is unable to find it mutual k-nearest neighbor pair as other points have already found their neighbor pair. So F sends request to points B, C, D and E but none of these points reciprocate and sends messages to F. If a point gets the same mutual k-nearest neighbor candidate in last three iterations, they are selected as the final neighbors and form our mutual K-nearest neighbor table. **Table 2** shows the results after the 3rd iteration. In our example, points B, C, D, and E have already found their mutual K-nearest neighbor as they have selected same candidate mutual k-Nearest candidate points in 3 iterations. It is noticed that A and F have received less than '2' messages and hence it keeps increasing its k-value in every iteration. The messages passing procedure continues till one of the conditions are met:

**Table 2: M-kNN Table in 3rd iteration.**

| Points | Request Sent | Response Received | k-value | Candidate Mutual KNN |
|---|---|---|---|---|
| A | B, C, D, E | – | 4 | – |
| B | C, D | A, C, D, F | 2 | C, D |
| C | B, E | A, B, E, F | 2 | B, E |
| D | B, E | A, B, E, F | 2 | B, E |
| E | C, D | A, D, E, F | 2 | C, D |
| F | B, C, D, E | – | 4 | – |

1. All points find their k mutual neighbors
2. No more points are available in dataset that is searching for neighbors

3. User specified number of iterations is reached (Threshold)

**Table 3: Message Passing Final Iteration.**

| Points | Request Sent | Response Received | k | Candidate Mutual k-NN |
|---|---|---|---|---|
| A | B, C, D, E, F | F | 5 | F |
| F | A, B, C, D, E | A | 5 | A |

In the final iteration, it is observed that A and F become mutual k-Nearest neighbor of each other and hence our final set of M-kNN is shown in **Table 4**.

**Table 4: Final Table After User Specified Iteration.**

| Points | M-kNN |
|---|---|
| A | F |
| B | C, D |
| C | B, E |
| D | B, E |
| E | C, D |
| F | A |

**Algorithm:** MKNN Message passing for generating MKNN table

```
Input  : No. Of Iterations N, LocalK :k_p , GlobalK: k_g, Points :P
Output: MKNN Relation Table MKNN
begin
    repeat
        for P_i ∈ P do
            SendRequest( P_i, k_p);      /* send message to k points */
        end
        for P_i ∈ P do
            NeighboringPoints P_n = GetResponse(P_i);
            if length(P_n) > k_g then
                Array MKNNCandidate_i = SelectBestK(P_n) ;
            else if length(P_n) = k_g then
                Array MKNNCandidate_i = P_n ;
            else
                k_pi = k_pi + 1 ;                    /* update k value */
            end
            if MKNNCandidate_i same in last 3 iteration then
                MKNN _i = MKNNCandidate_i;
            end
        end
    until N iterations;
    return MKNN
end
```

**Algorithm 1. Generating M-kNN table**

## 3.3 Clustering from Mutual k-Nearest Message passing table

The M-kNN relationship table generated by **Algorithm 1** will be used for clustering points in a dataset using two Algorithms. **Algorithm 2** determines the initial set of clusters by reading the M-kNN table sequentially; **Algorithm 3** performs cluster merging operation on initial set of clusters in Mutual k-nearest neighbor way where it selects the best cluster to merge based on Mutual k-Nearest neighbor relationship of the cluster. For Algorithm 1 the following parameters are defined:

**Definition 2: Radius** of a point $P$ is defined as the average distance of all its mutual $k$-nearest neighbors from $P$. For a point $P$ which has distances with its M-kNN neighbors as

$d_1, d_2, \ldots, d_K$:

$$R_P = \frac{\sum_{i=1}^{K} di}{K}$$

- **Definition 3: Cluster Initiator** A point that starts building clusters by first including all its M-kNN.

As seen from **Table 3**, in the final iteration A and F increase their $k$ value and send messages to each other and become mutual neighbors. The other points have already found their final neighbors and hence are excluded from further consideration. The algorithm is listed in **Algorithm 1**. As described in **Algorithm 1**, $k_g$ is the global $k$ value that is provided by user. Initially every point has initial $k_p$ which is equal to global $k$ value $k_g$. Every point sends request messages to nearest k points using **SendRequest** function that finds best $k$ points and sends messages to them. Responses to every point are received from **GetResponse** function. **SelectBestK** function selects the best $k$ mutual neighbor i.e. the neighbors who were requested by a point and have also responded to a point and are closest in terms of a distance measure. Finally, if a point receives same mutual neighbors in last three iterations, it adds the mutual neighbors in final M-kNN table **Table 4**. Then clustering is performed on this table that is described in next section.

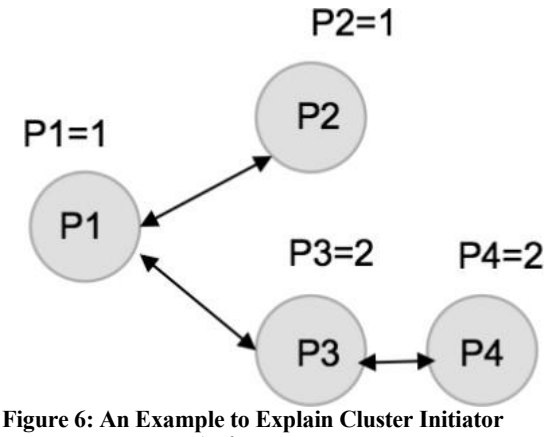



**Figure 6: An Example to Explain Cluster Initiator Assignment**

Radius of a point will help us determine the density of the clusters. **Figure 6** is used to illustrate the concept of cluster initiator assignment with 4 points. Bi-directional arrows represent mutual relationship between points. Let us assume the points $P_1$, $P_2$, $P_3$, $P_4$ are sorted in decreasing value of radius respectively.

Lets start with $P_1$ and assign $P_1$ as its own initiator 1. Point $P_2$ M-kNN points are $P_2$ and $P_3$, so they are also assigned as $P_1$'s initiator and hence labeled as 1. Point $P_4$ is not assigned, so it is assigned to 2, but $P_4$ has $P_3$ as its M-kNN neighbor that was already assigned to $P_1$. However, the distance between $P_3$ and $P_4$ is less than the distance between $P_3$ and $P_1$. Hence, $P_3$ is newly assigned to $P_4$'s initiator, which is 2.

To illustrate the process further, with the same example described in Subsection B is continued. The radius values of each point are calculated and sort them in descending order of radius values as shown in **Table 5**. Start with point A. Since A has not been assigned to an initiator, assign A as a cluster initiator.

**Table 5: Generating Preliminary Clusters.**

| Points | M-kNN | Radius | Cluster No. |
|---|---|---|---|
| A | F | 4 | A=1, F=1 |
| F | A | 4 | 1 |
| B | C, D | 2.25 | B=2, C=2, D=2 |
| C | B, E | 2.25 | C=2 |
| D | B, E | 2.25 | D=2 |
| E | C, D | 2.25 | E=3, C=2, D=3 |

Then it is checked if the mutual neighbor of A has been assigned to a cluster or not. In this case two scenarios can happen:

1. If mutual neighbor of point F has not been assigned a cluster, point F will be assigned to A's cluster
2. If mutual k-nearest neighbor point F is already assigned to

a cluster, then there are two cases:

- If distance of point F with its previous initiator > distance of F with current initiator: update F's initiator with the new initiator.
- If the distance of F with its previous initiator < distance of F with current initiator: Make no changes.

In our example, F is not assigned and hence it will be assigned to A's cluster. Similarly, cluster initiators are continuously assigned, and mutual *k*-nearest neighbors are added to the cluster of the nearest cluster initiator. Point D was initially assigned to B but in further iteration the algorithm finds that distance of point E with D is less than distance of B and D. So D's was assigned to E's cluster. From the **Table 5**, it is observed that there are three cluster initiators: A, B and E that form initial set of clusters. All the other points belong to either of these clusters. The algorithm for this procedure is mentioned in **Algorithm 2**. The algorithm takes as an input radius sorted points and outputs a preliminary table of points with cluster labels. The next procedure of the algorithm is to merge clusters obtained.

## 3.4 Clustering Merging

Unlike previous approaches to cluster merging, a new cluster merging process via message passing is defined. The process similar to Subsection 3.2 is repeated, but now clusters use message passing to find their mutual neighbors and merge with its best mutual *k*-nearest neighbor cluster. However, lets define new metrics to measure inter-cluster distance as follows:

```
Algorithm: Finding initial set of clusters
Input  : MKNN Table: MKNN, RadiusSortedPoints :P_r
Output: Preliminary Cluster labels: C
begin
    cluster=1 ;
    for P_i ∈ P_r do
        if cluster label C_i is not set then
            C_i=cluster;
        for P_j ∈ P_i do
            if cluster label C_i is not set then
                C_j=cluster;
            else
                get P_k cluster exemplar of P_j;
                if distance(P_i, P_j) < distance(P_j, P_k) then
                    C_j=cluster;
            end
            cluster=cluster+1;
        end
    end
    return C
end
```

**Algorithm2. Generating initial set of clusters**

- **Definition 4: Linkage:** A point has a linkage to a cluster $\mathcal{N}$ if there is at least one point in $\mathcal{N}$ that is a mutual neighbor (M-kNN) of point $p$.
- **Definition 5: Closeness** : Closeness of cluster Clusteri to Clusterj is no. of points in Clusteri that has a Linkage to Cluster j
- **Definition 6: Sharing:** Sharing S of cluster Clusteri into Clusterj is number of Mutual k-Nearest Neighbor pairs that have one in Clusteri and other in Clusterj
- **Definition 7: Connectivity** If $Cluster_i$ has $k_i$ points and $Cluster_j$ has $k_j$ points, the connectivity of $Cluster_i$ to $Cluster_j$ is defined as:
  Connectivity$_{ij}$ = (Sharing / ($k_i$ X $k_j$)) X (Closeness/ $k_i$)

The merging process starts with every cluster sending *k* messages to other clusters and identifying the best cluster that reciprocates as its mutual *k*-nearest neighbor. In order to merge, both clusters should have a high connectivity value with each other and should reciprocate the mutual neighbor relationship. Once two clusters become mutual *k*-nearest neighbors of each other, they merge to form a single cluster. The new cluster becomes the union of the M-kNN neighbors of the two merged clusters. This new cluster must then recalculate connectivity values when sending *k* messages to other clusters.

**Table 6: Cluster Merging First Iteration**

| Cluster | Request Sent | Response Received | Points |
|---|---|---|---|
| C1 | C2, C3 | – | A, F |
| C2 | C3, C1 | C1, C3 | B, C |
| C3 | C1, C2 | C1, C2 | D, E |

To illustrate the message passing cluster merging, the example of the previous section with our preliminary clusters is contined. From **Table 6**, it can be see that $\mathcal{C}2$ sends messages to $\mathcal{C}3$.

Here$\mathcal{C}2$ and $\mathcal{C}3$ become mutual K-nearest neighbors of each other.
So after the end of the iteration, $\mathcal{C}2$ and $\mathcal{C}3$ merge to form a single cluster. The process is repeated until the desired number of clusters is obtained.

In this specific example, the algorithm converges after merging $\mathcal{C}2$ and $\mathcal{C}3$ and generates 2 clusters as shown in **Table 7**.

**Table 7: Clusters and Their Points**

| Cluster | Points |
|---|---|
| C1 | A, F |
| C2 | B, C, D, E |

```
Algorithm: Final Clusters
Input  : Cluster Label Array: C , K value: k
Output: Final Cluster labels: C
begin
    Calculate initial ConnectMatrix CM ;
    repeat
        for C_i ∈ C do
            SendRequest(CM,k) ;   /* send message to k clusters */
        end
        for C_i ∈ C do
            NeighboringCluster C_n=Receive(CM, C_i ) ;
            Select Closest Neighbor in C_n from C_j ;
            Merge C_i,C_j into C_new ;
            Update CM and replace C with C_new ;
        end
    until no more merging can be done or required clusters have been achieved;
    return C
end
```

**Algorithm3. Merging Procedure to generate final clusters**

From **Algorithm 3**, it can be seen that the merging operation is done via message passing like **Algorithm 1**, except this time connectivity values between clusters is used rather than any distance function. **SendRequest** method is similar to**Algorithm 1** that sends request to k-neighboring cluster and Receive method finds the clusters that send response to a cluster. Then select only the closest mutual neighboring cluster to merge which have high connectivity value with the cluster and then update the merged cluster in original cluster. After merging operation, new connectivity values are calculated and then another iteration of message passing starts. The process not only prevents comparison of every cluster to other, it also reduces the total number of clusters in each iteration. Based on clustering requirement a convergence criteria can be

defined like stopping when the total number of clusters reach a specific value or the clusters have almost 0 connectivity values between each other.

# 4. EXPERIMENTAL SETUP AND RESULTS

## 4.1 Datasets

This Section deals with discussion of results that were obtained on M-kNN Message Passing algorithm (M-kNN) as described in previous chapter. In order to validate the algorithm, a synthetic supervised dataset and two datasets from UC Irvine's Machine Learning repository is used. The details of the datasets are given on **Table 8**. To demonstrate the functionality of our algorithm we design our own synthetic dataset with varying density. The real world datasets have been read directly from tab separated flat files.

**Table 8: Datasets used for experiments**

| Sno | Dataset | Inst. | Attr. | Attribute Types | Class |
|---|---|---|---|---|---|
| 1 | Synthetic | 95 | 2 | Real-valued, continuous | N/A |
| 2 | Pen Digits | 10,992 | 16 | Integers in the range 0–100 | 0–9 |
| 3 | Plants | 34,781 | 65 | States of US & Canada | N/A |

## 4.2 Experiments

All the algorithms described in the paper have been implemented on MATLAB running on a modern x86 64 bit chip. The results of the algorithm are written in a flat file that was used for further analysis.

## 4.3 Synthetic Datasets

We chose a two-dimensional synthetic dataset to test our algorithm with varying densities having dense and sparse points. As we can see in **Figure 7**, there are four regions of high densities and regions of low-density points surround 2 of them. We can also see that the low-density points and high-density points are clearly separated from each other so as to form distinct clusters. Ideally, a clustering algorithm should be able to determine four dense clusters and two sparse clusters having a total of 6 clusters.

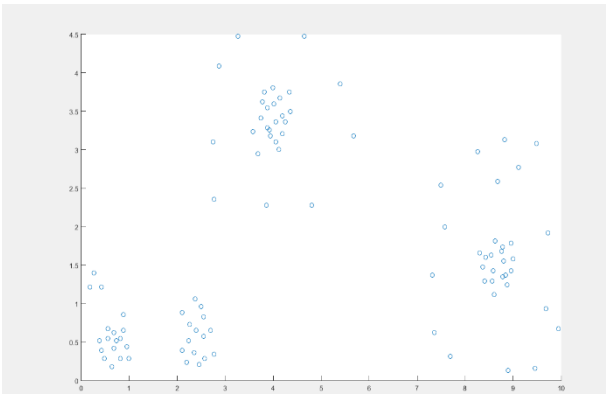
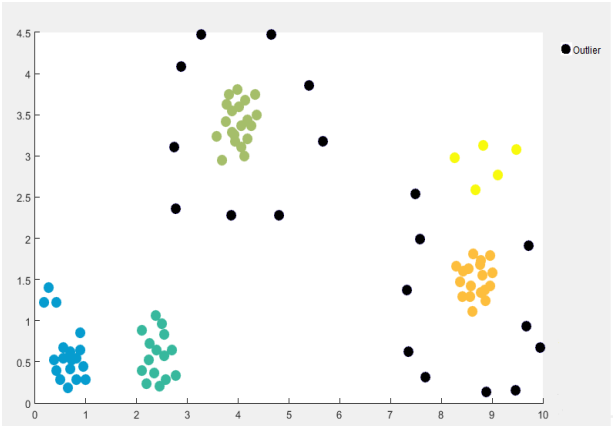


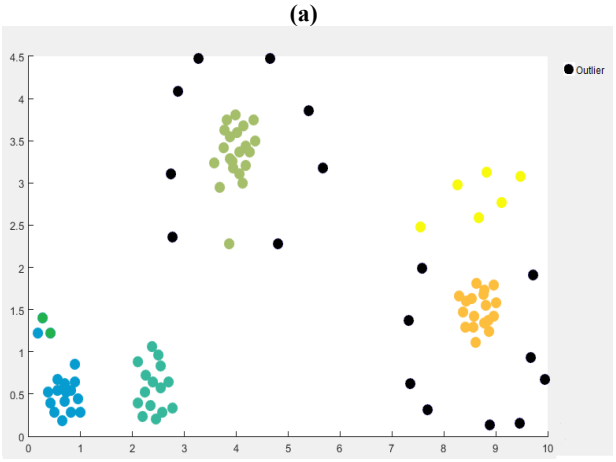
**Figure 7: Synthetic dataset with varying data density.**

Then multiple iterations of DBSCAN algorithm is run with various values of min points and epsilon to obtain clusters as shown in **Figure 8**. The colored dots represent points belonging to same cluster. Black dot signifies an outlier detected by DBSCAN. It can be clearly seen that DBSCAN is easily able to detect dense clusters, but it is unable to detect the sparse clusters accurately.

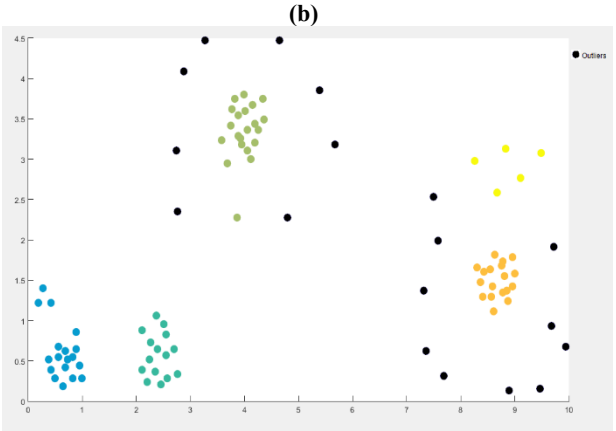
Furthermore, DBSCAN marks a lot of points as outliers as indicated by points colored black. The main problem of DBSCAN algorithm is it tries to cluster points with specific radius values and can only detect specific cluster shapes based on the input parameters. On the other hand, when multiple iterations of the k-means algorithm is run with different initial centroids and constant k value ($k = 6$) until the best clustering result is obtained. **Figure 9**.



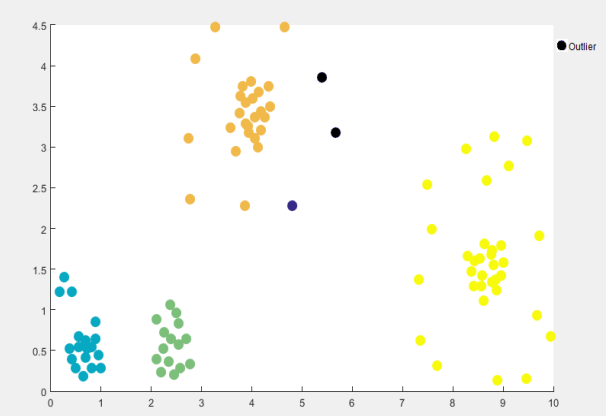

**(a)**




**(b)**




**(c)**




**(d)**

**Figure 8: a, b, c, d: Results of DBSCAN with different parameter values.**

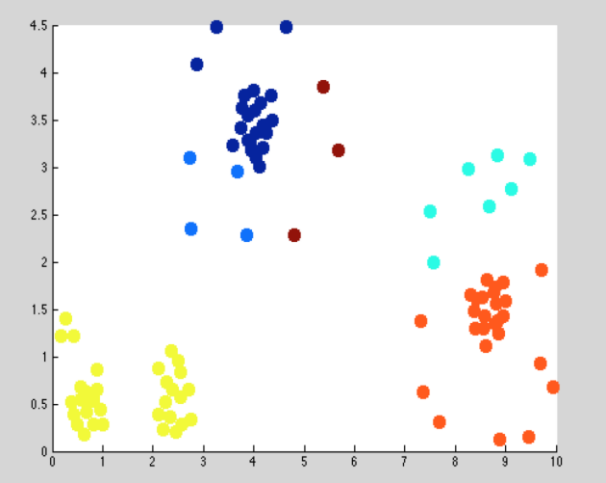


**Figure 9: Best result of K-means at K=6**

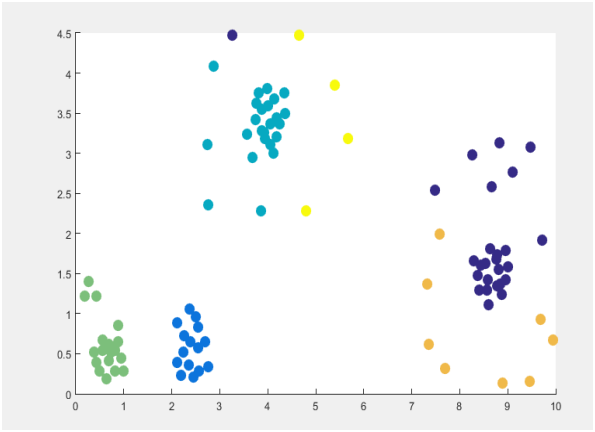
It is observed that K-Means fails to effectively detect both dense and sparse clusters, resulting in suboptimal clustering outcomes. The authors subsequently applied M-kNN to the same synthetic dataset using various *k* values, repeatedly executing the M-kNN merge operation until six clusters were formed. M-kNN successfully identified dense clusters, excluded them from further processing, and enabled surrounding sparse points to interact and form mutual *k*-nearest neighbor relationships. As illustrated in Figure 10, M-kNN identifies five clusters—four dense and two sparse—within the dataset. Furthermore, minor variations in *k* do not significantly affect clustering results, indicating the algorithm's robustness to input parameter changes. These findings suggest that M-kNN performs well in scenarios where dense clusters are embedded within sparse regions. This will be further validated using a real-world dataset in the following section.

## 4.4 Pen Digits Dataset

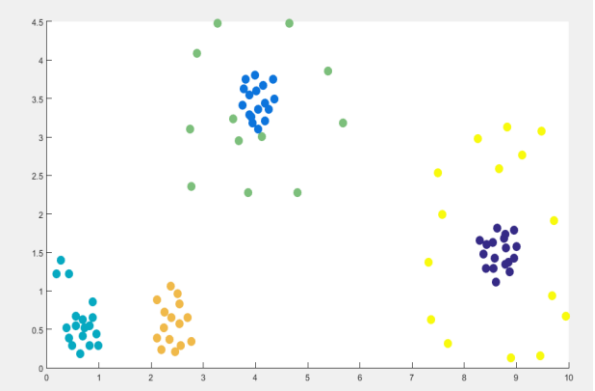
To evaluate the proposed algorithm, the Pen-Based Recognition of Handwritten Digits dataset from the UCI Machine Learning repository was utilized. Both training and test points were included for clustering, resulting in a total of 10,000 data points with 16 attributes. As the ideal number of clusters was unknown, the M-kNN algorithm was executed until a threshold of 50 clusters was reached. Clustering results were monitored progressively, starting from 500 clusters down to 50.

For K-means, multiple iterations were conducted with *k* values ranging from 50 to 500. Each iteration involved running K-means ten times with different initial centroids, and the clustering outcomes were averaged. For the DBSCAN algorithm, the minimum points and epsilon values were varied to generate clusters within the same range. Similarly, for the Shared Nearest Neighbor algorithm, the *k* value was adjusted to generate a graph, and clusters were merged until the desired range was achieved.

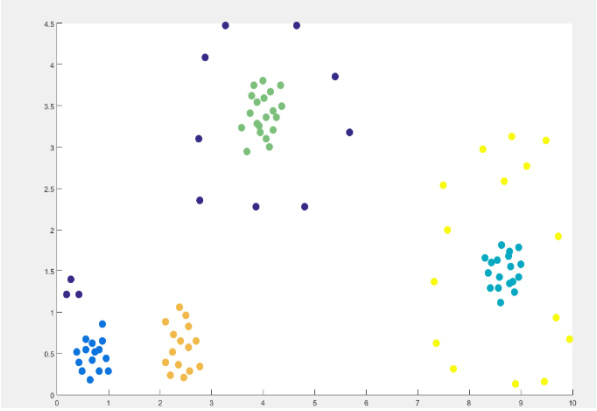
For each cluster, the class membership of its points was determined, and the majority class was assigned to the cluster. A class was considered the majority if more than 50% of the points within the cluster belonged to it. For instance, if M-kNN identified a cluster A with five points—three from class 1 and two from class 2—then class 1 was designated as the majority class for cluster A, and all points in that cluster were labeled accordingly. Once majority classes for all points is obtained which is our clustering result we compare them with original classes in our dataset and generate confusion matrix. The confusion matrix is used to calculate precision and recall values.



**(a) M-kNN results at k=3**



**(b) M-kNN results at k=4**



**(c) M-kNN results at k=5**

**Figure 10: a, b, c.**

When we plotted precision and recall values in a graph **Figure 12**, we could clearly see M-kNN performing better than existing algorithms. M-kNN gives a consistently high value of precision in every step than DBSCAN, SNN or K-Means. We can intuitively say that if we increase the number of clusters, the clustering algorithms will generate better clusters. M-kNN for large clusters gives very good results with precision value as high as 93%, which implies 93% of the points have been correctly classified by the algorithm, compared to SNN and K-means that give precision values at around 85%. DBSCAN generated fewer clusters than our algorithm for various values of epsilon radius and minimum points, and treated several points as noise.

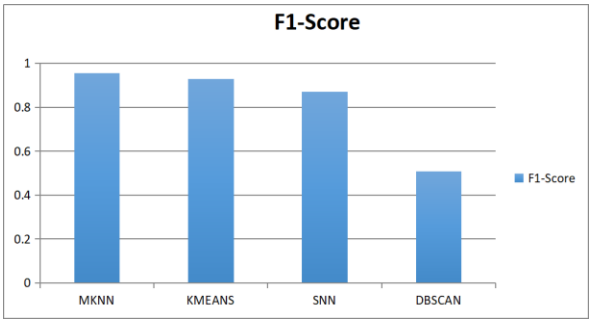



**Figure 11: Comparison of F1-score for Various Clustering Algorithms for Pen-digits Dataset.**

The results obtained in **Figure 12** are summarized in **Figure 11** by averaging precision and recall values and then calculated the F1-score for each algorithm. From **Figure 11**, we can clearly see M-kNN having a higher F1 score, thereby showing the consistency of our algorithm.

## 4.5 Performance Analysis

A large high-dimensional dataset, Plants, from the UCI repository was used to analyze CPU time. The Plants dataset contains plant names as instances and state abbreviations as attributes. Each plant name includes a list of comma-separated state abbreviations indicating the states in which it is found. The dataset was transformed into binary vectors, where a value of 1 denotes the presence of a particular plant in a state. This dataset was utilized for both clustering and CPU time analysis to evaluate algorithmic performance over time. The Jaccard distance was employed as the distance metric.

In SNN, a graph is first generated, which is then used to identify SNN relationships and form clusters. Similarly, KNN identifies *k*-nearest neighbor relationships between points to form clusters. M-kNN also follows a two-step process: first identifying M-kNN relationships, then merging clusters based on those relationships. M-kNN, KNN, and SNN were executed on datasets ranging from 2,000 to 10,000 points, and their performance was compared as shown in **Figure 13**. The total time recorded for each algorithm corresponds to the time taken to identify 200 clusters. It was observed that M-kNN exhibits superior performance, demonstrating near-linear scalability with increasing dataset size.

Theoretically, performance can be further enhanced by implementing M-kNN in a parallel computing environment. In such a setup, multiple processors execute the M-kNN message-passing algorithm concurrently. Each processor handles a subset of points that form clusters using the M-kNN approach, resulting in *k* processors generating *k* distinct clusters.

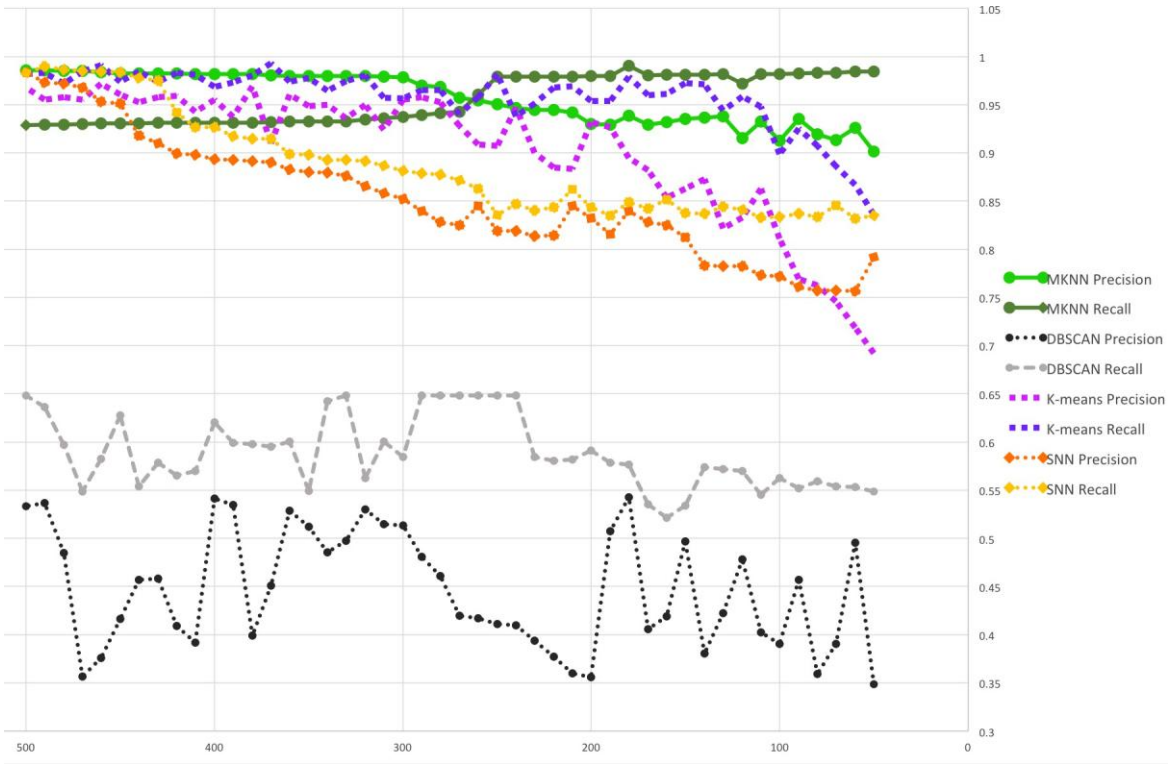



**Figure 12: Comparison of precision and recall for M-kNN vs. other algorithms.**

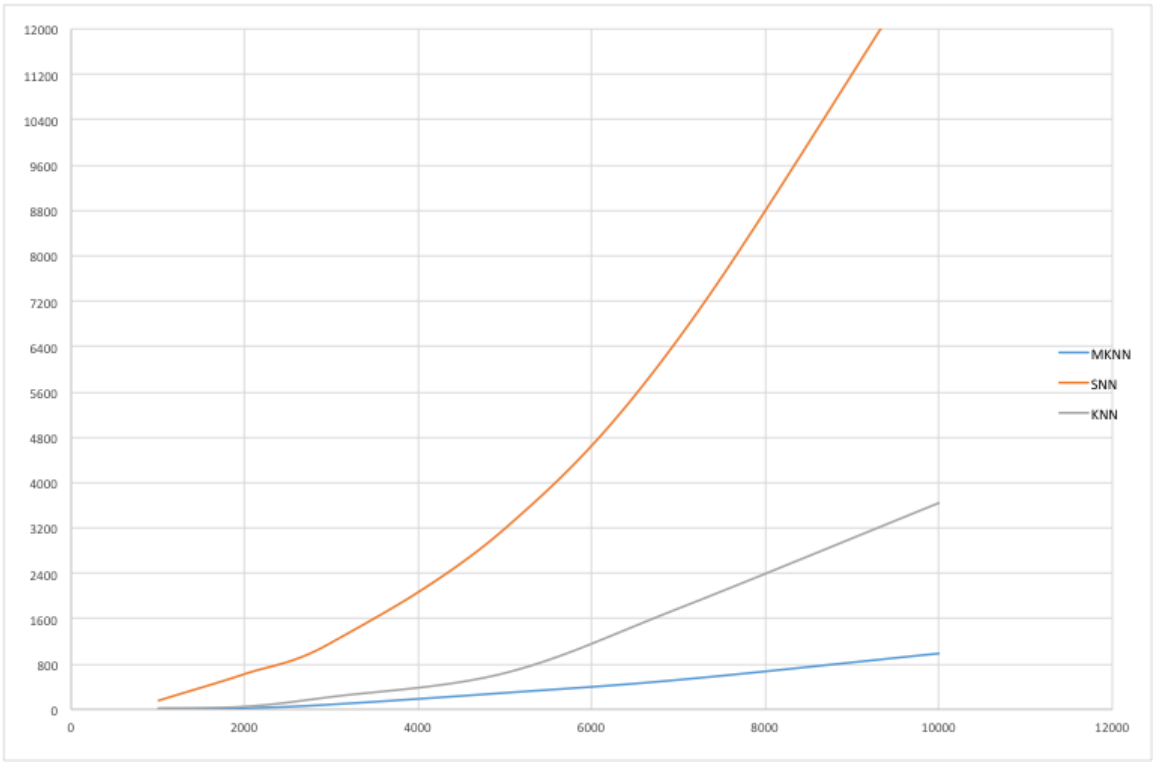



**Figure 13: Performance Analysis of M-kNN and SNN on Plants Dataset. X-axis: Size of Dataset. Y-axis: Time in Seconds.**

Each cluster will also contain information about its mutual $k$-nearest neighbors. When two clusters $C_1$ & $C_2$ merged, the processor $PR_2$ is put into an idle state and $C_2$ is copied into $C_1$. M-kNN information of $C_1$ is then updated to its new neighbors. Merge and continue dropping processors until the desired number of clusters is obtained. Thus, M-kNN execution can be further improved if implemented in a parallel processing environment. Those can be applied to detect clusters in massive vector databases to refine LLM searches. The actual implementation is beyond the scope of this paper.

## 4.6 Seeds Dataset

To further validate the functionality of the proposed algorithm, the algorithm was applied to the Seeds dataset from UC Irvine. UCI Seeds is a widely used density-based dataset for evaluating clustering algorithms, with various methods demonstrating strong clustering performance. In this dataset, kernels belong to

three distinct groups of wheat: Kama, Rosa, and Canadian, each comprising 70 elements. Ideally, a clustering algorithm should produce three high-quality clusters with 70 points each. A similar approach to the previous section was used, monitoring clustering performance until three clusters were obtained. The results show that the proposed algorithm yields high precision and recall values for this dataset as well. As illustrated in the confusion matrix in Figure 14, M-kNN successfully clusters one of the kernel groups with 100% precision. Consistent with earlier findings, the M-kNN clustering process remains stable throughout each merging step. Upon merging and achieving three clusters using the merging algorithm, the highest precision value of 89% was recorded. As with the previous dataset, the results were summarized by calculating the F1 score based on precision and recall values, as depicted in the bar graph in Figure 15.

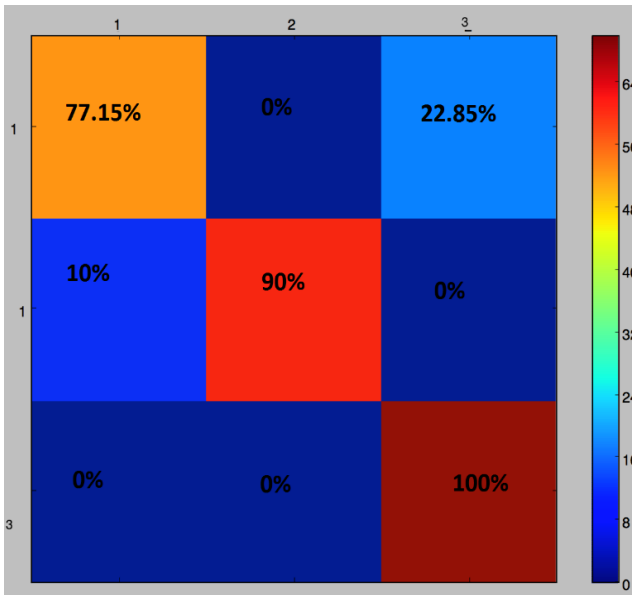



**Figure 14: Confusion matrix for M-kNN for seeds dataset at k=3.**

# 5. CONCLUSION AND FUTURE DIRECTIONS IN AI USE- CASES

## 5.1 Conclusion

M-kNN further improves clustering through mutual K-nearest, neighbor relationship by optimizing the process of message passing and merging of clusters. Previous work on M-kNN involved an extra sorting operation to discover the M-kNN relationship and also required checking every possible pairwise distance value to find the mutual neighbor. However, M-KNN message passing allows a point only to send K messages to neighboring points, thereby pruning the number of points and drastically reducing the time to calculate mutual k neighbor relationship. Also, the algorithm requires only one parameter, which is K value and the results are not highly dependent on value of K. This novel M-kNN algorithm can be applied to many real-world situations

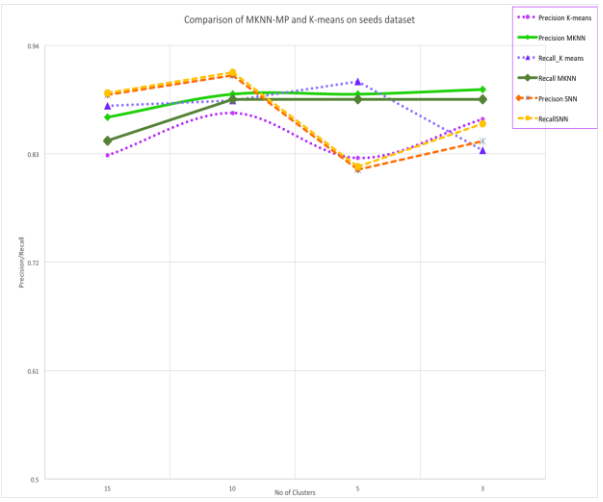



**Figure 15: Comparison of precision and recall for clustering algorithms for seeds dataset.**

specifically in situations where the data density is varying such as vector databases. In the contemporary AI-driven landscape, the expanding adoption and versatility of vector embedding technology within large language models (LLMs) underscore the critical need for robust clustering mechanisms and outlier detection. These techniques are essential to preserve search quality and ensure stability. This is particularly relevant in the industrial data, where customer data spans diverse segments and unique profiles, enabling more tailored and strategic applications.

# 6. REFERENCES

[1] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[2] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Library of Congress, 2006.

[3] P. Tan, M. Steinbach, and V. Kumar, "Data mining cluster analysis: Basic concepts and algorithms," 2013.

[4] Z. Hu and R. Bhatnagar, "Clustering algorithm based on mutual k-nearest neighbor relationships," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 2, pp. 100–145, 2012.

[5] D. Sardana and R. Bhatnagar, "Graph clustering using mutual k-nearest neighbors," in *Active Media Technology*, pp. 35–48, Springer International Publishing, 2014.

[6] L. Ertoz, M. Steinbach, and V. Kumar, "A new shared nearest neighbor clustering algorithm and its applications," in *Workshop on Clustering High Dimensional Data and its Applications at 2nd SIAM International Conference on Data Mining*, pp. 105–115, Apr. 2002.

[7] M. A. Wong and T. Lane, "A kth nearest neighbour clustering procedure," in *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface*, pp. 308–311, Springer US, Jan. 1981.

[8] H. Kriegel *et al.*, "Density-based clustering," *Wiley Interdis- ciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[9] L. Ertöz, M. Steinbach, and V. Kumar, "Finding clusters of different sizes, shapes, and densities in noisy, high

dimen- sional data," in *SDM*, 2003.

[10] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, *On the surprising behavior of distance metrics in high dimensional space*. Springer Berlin Heidelberg, 2001.

[11] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[12] Z. Hu, *Multi-Domain Clustering on Real-Valued Datasets*. PhD thesis, University of Cincinnati, 2011. https://etd. ohiolink.edu/.

[13] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *KDD Workshop on Text Mining*, vol. 400, 2000.