Computing and Informatics, Vol. 40, 2021, 1322–1344, doi: 10.31577/cai_2021_6_1322

EFFICIENT DENSITY-BASED PARTITIONAL CLUSTERING ALGORITHM

Zareen Alamgir, Hina Naveed

Computer Science Department
National University of Computer and Emerging Sciences
Lahore, Pakistan
e-mail: zareen.alamgir@nu.edu.pk, naveedhina32@gmail.com

Abstract. Clustering is an important data mining technique that helps to detect hidden structures and patterns in the data. K-means algorithm is one of the most popular and widely used partitional clustering algorithms. It is a simple and efficient method but has several shortcomings. One major drawback of traditional K-means is that it selects initial centroids randomly, resulting in low-quality clusters. Various K-means extensions are designed to solve the issue of the random initial centroid. A novel density-based K-means (DK-means) algorithm is recently proposed that uses density-parameters for selecting initial centroids. It outperforms K-means in terms of accuracy at the cost of time. In this research, we present an efficient density-based K-means algorithm (EDK-means) that uses advance data structures and significantly reduces the DK-means algorithm's execution time. Furthermore, we rigorously evaluated the performance of density-based K-means on different challenging real-world datasets and compared it with traditional K-means. The experimental results are promising and show that density-based K-means outperforms K-means. It converges more rapidly than basic K-means, and it works well for the datasets with different cluster sizes.

Keywords: Clustering, K-means, density-based K-means, EDK-means, partitional clustering

Mathematics Subject Classification 2010: 91C20, 11Y16

1 INTRODUCTION

The era of the dot com bubble is far gone, but it started a chain reaction leading to the information age, and right now, the era of information is at its peak with significant advancements in areas such as big data, IoT, and data mining. Now we have devices collecting data 24/7, let alone the terra bytes of data uploaded on the web every day in the form of research, social interactions, news, and entertainment. The continuously growing sea of information is useless if not utilized properly. The need of the hour is to process the massive data and extract useful patterns from it. Almost 80 % of this information is unstructured and unsupervised. This is where clustering comes into place; it is an unsupervised technique that can extract meaningful information from a massive pile of data.

Clustering is a subfield of data mining specialized in finding hidden structures in data without external information. The clustering algorithms use heuristics to solve the problem in a reasonable amount of time but do not ensure an optimal clustering solution. Clustering has vast and diverse applications. It is extensively used in data preprocessing, customer segmentation, data partitioning, outlier detection, and data analyses. It is an unsupervised technique that groups data points with similar traits and characteristics. To discover meaningful patterns and knowledge from data is extremely challenging if the external information is not available. Many clustering algorithms are designed to handle various types of datasets, and among them, partitional clustering is the most widely deployed.

K-means is the most popular partitional clustering algorithm. The idea behind the widely used clustering technique is to partition the given dataset $D = \{x_1, x_2, \dots x_n\}$ with n data points into K clusters $C = \{c_1, c_2 \dots c_k\}$, where C is a set of cluster centers. However, finding K cluster centers in the data space is an NP-hard problem [19]. The clustering algorithms use heuristics to solve the problem in a reasonable amount of time but do not ensure an optimal clustering solution. K-means is an optimization problem with the objective to minimize the distances between the data points and their closest cluster centers, thus producing compact clusters. The objective function can be formulated as follows:

$$F(C; D) = \sum_{j=1}^{k} \sum_{i \in cluster\, j} d^2(x_i, c_j) \tag{1}$$

where $d(x_i, c_j)$ is the distance between the data point x_i and the cluster center c_j. Algorithm 1 gives the basic pseudo-code of the K-means algorithm.

K-means is simple and efficient, but it has some flaws that question the algorithm's accuracy and make it unstable. There exist many variations of K-means that try to overcome these shortcomings [4, 15]. The random selection of initial cluster centers is one of the major drawbacks of the traditional K-means algorithm. Different ideas and methods are adopted to randomly selecting the k initial centers. Few most common approaches are:

Algorithm 1 Basic K-means Algorithm

Input: Dataset D, Number of clusters K
Output: K cluster centers $c_1, c_2, \ldots, c_k$

Initialization Step:
Randomly initialize the C centers $c_1, c_2, \ldots, c_k$
Learning Loop:
repeat
  for each data point x in D **do**
    for $j = 1, 2, \ldots, K$ **do**
      Calculate the distance between x_i and c_j
    end for
    Assign point x_i to the closest centroid
  end for
  for $j = 1, 2, \ldots, K$ **do**
    Update c_j to the mean of all points assigned to it
  end for
until none of the centroid changes its value

1. Select the first k elements in D as initial centers,
2. Pick k elements uniformly distributed over D, and
3. Randomly assign the n elements to the k clusters, then compute the mean of each cluster and use it as its initial center.

Poor initialization of centroids can adversely affect the results of K-means and can lead to slow convergence, empty clusters, low-quality clusters, and increase the likelihood of getting stuck in the local optima. Selecting K initial centroids randomly at each run gives different clustering results [20]. Ideally, if an algorithm is repeated multiple times, it should give similar results. This is clearly not the case with K-means when the objective function $F(C; D)$ possesses multiple minima [21]. To select the best solution, we generally use one or more validity criteria in order to assess and compare the quality of all these candidate solutions.

It is interesting to note that improved initialization techniques only reduce the chances of K-means getting trapped in local optima. However, the main objective of K-means is to minimize the intra-cluster variance, and it terminates when centroids do not change. Hence, it can converge to local optima, regardless of the initialization technique used, and may never reach global optima. K-means serves as a heuristic and attempts to find an optimal solution that might be global or local. In real-world applications, local minima of the objective function represent approximate yet satisfactory solutions. Many efforts have been made to find better local optimum solutions that include local search strategies (Hartigan-Wong method), metaheuristics, and genetic algorithms.

The different variants of the K-means that strive to deal with the problem of selecting initial centroids have stability issues and need to preset the values of input

parameters. Recently proposed the density-based K-means (DK-means) algorithm seems quite promising [1]. It uses density parameters to find the initial centroids effectively. The algorithm improves the accuracy of basic K-means and makes it stable but at the cost of execution time. Hence, it is not feasible to apply the DK-means algorithm on a huge dataset.

In this research work, we propose an EDK-means algorithm, an efficient version of density-based K-means that employs advanced data structures to enhance the speed of density-based K-means and reduce its time complexity while maintaining its clustering quality. We rigorously evaluate the performance of the proposed algorithm on various challenging datasets with many features. This paper makes the following contributions:

- Introduce an efficient density-based K-means (EDK-means) algorithm that uses advanced data structures to improves the time requirement of the density-based K-means algorithm.
- Evaluate the performance of the EDK-means algorithm on different datasets in-term of accuracy and time.
- Compare the EDK-means algorithm's performance with traditional K-means with different initializations.
- Examine the behavior of the EDK-means algorithm on challenging datasets having different cluster sizes, shapes and densities.

The rest of the paper is organized as follows: Section 2 presents the discussion on related work. Section 3 presents the efficient density-based K-means algorithm. Section 4 includes the details of datasets, results, and analysis of the algorithm. Section 5 concludes the paper and outlines the future work.

2 LITERATURE REVIEW

Clustering forms groups of similar and related data points to discover the underlying pattern and hidden information in the data. There are various types of clustering algorithms: partitional clustering, hierarchical clustering, density-based clustering, model-based clustering, and grid-based clustering. The different clustering algorithms are designed to identify clusters of varied shapes, sizes, and densities in the presence of noise and outliers [6].

K-means, a primary partitional clustering algorithm, is the most popular choice among researchers of different domains as it is a simple, easy, and efficient method. However, the traditional K-means algorithm has various shortcomings and drawbacks. It does not work well if the data has outliers or clusters with non-convex shapes, varying sizes, and densities. One of the major shortcomings of traditional K-means is that it randomly selects initial centroids; this can significantly degrade the clustering quality. In the last decade or so, several variations and extensions are proposed to K-means to handle numerous issues faced by K-means. Arthur and Vassilvitskii [3] introduce a variation to the K-means algorithm called K-means++

that caters to the initialization problem of basic K-means. K-means++ works on the intuition that k initial centroids spread out and span the dataset. It selects the first centroid randomly, and the next centroids are selected with probability proportional to its squared distance from the closest previously selected centroid. Later, Sohler et al. [11] combine the K-means++ algorithm with the local search strategy. This variation starts with the K-means++ algorithm and generates k clusters. Then it continues for another $O(k \log \log k)$ rounds and generates few more centers. The newly sampled centers are exchanged with old centers if the exchange improves the cost; otherwise, they are discarded. Aggarwal and Singh [2] blend K-means++ with techniques inspired by nature: Cuckoo, Bat, and Krill Herd algorithms. These techniques enhance the quality of clusters and improve performance. Li [13] introduces the more generalized form of the K-means++ algorithm, which works by selecting the farthest data point from the nearest cluster center and achieve results similar to K-means++. The K-means++ algorithm and its variations suffer from instability and need to preset the values of the parameters.

Tzortzis et al. develop a clustering algorithm called MinMax K-means [17] that targets the problem of selecting the initial centroids. It chooses the initial cluster centers randomly like basic K-means but assigns weights to clusters to minimize the maximum intra-cluster distance. Fränti and Sieranoja [7] study the factors that affect the performance of the K-means algorithm. They show that the MaxMin strategy reduces the clustering error due to the poor initialization of K-means from 15 % to 6 % on average. However, it also suffers from the instability issue. Hou [8] presents a K-means silhouette algorithm that uses silhouette index to measure the cluster quality. The algorithm uses maximum and minimum distances to improve the way of setting the initial centroids. This approach also helps in determining the optimal number of clusters. Lakshmi et al. [10] present another approach to solve the initialization problem of traditional K-means. The method starts by choosing the first initial centroid using feature mean and then removes its k nearest neighbors to find the next cluster center. Yu et al. [18] develop two improved versions of the K-means algorithm, namely tri-level K-means, and bi-layer K-means. The tri-level K-means algorithm uses three layers of clustering and data normalization to deal with the noise and the issue of initial cluster centers. It also caters to the situation in which data changes frequently and trained cluster centers can no longer describe data in clusters. Bi-layer K-means is designed to handle the scenario when data points in the cluster are substantially different, and the centroid fails to define each point. It uses sub-clustering and data-matching techniques for clustering.

Researchers often combine different clustering techniques to handle challenging and complicated datasets and situations: some blend partitional clustering techniques with density-based algorithms to attain high-quality clustering. Density-based clustering works by calculating the density of each sample point. DBSCAN [5] is one of the most important density-based clustering algorithms. Singh and Meshram [16] examine the performance of DBSCAN and its variations such as DEN-

DIS [14], DBCURE [9] and others. Recently few authors have proposed the use of the density concept for effectively finding the initial cluster centers. Li et al. [12] propose an algorithm that uses M nearest neighbor, distance, and density to find initial cluster centers. Zhu and Ma [1] propose DK-means, a density-based K-means algorithm that uses density parameters to select the initial cluster centroids effectively. The algorithm attains stability, however, at the cost of time. Zhu and Ma also present a novel variance-based clustering validity index (VCVI) to detect the number of clusters in the datasets.

The K-means algorithm is susceptible to initial cluster centers. The different variations of K-means adopt different methods for choosing the initial cluster centers. However, most of the proposed variations are not stable and get stuck at a local optimum. Moreover, the few that handle the problem of stability like DK-means do so at the cost of time and thus are not practical for massive datasets.

3 EFFICIENT DENSITY-BASED K-MEANS (EDK-MEANS)

In this section, we present EDK-means, an efficient version of density-based K-means algorithm that employs advanced data structures like KD-trees and heaps to improve the performance of the DK-means algorithm. Density-based K-means [1] is the new variant of K-means that uses density parameters to choose initial cluster centroids instead of choosing them randomly. It combines the idea of partitional and density-based clustering algorithms. DK-means uses the average distance between the data points to identify points with high density. The average distance is computed as follows:

$$AvgDist = \frac{2}{n(n-1)} \sum_{i}^{n-1} \sum_{j=i+1}^{n} d(x_i, x_j) \tag{2}$$

where n is the number of data points and $(d(x_i, x_j))$ is the Euclidean distance between two data points x_i and x_j.

The neighborhood of a data point is a region with ϵ radius around the point. It is defined in terms of *AvgDist* and α, where α is the influence factor.

$$\epsilon = \alpha \times AvgDist, \quad 0 < \alpha \leq 1. \tag{3}$$

The influence factor α is set according to the characteristics of the underlying data distribution. In order to cater data with dense clusters that are relatively far apart, the α value should be in the interval $[0.05, 0.2]$ [1]. On the other hand, if the clusters in the data are not very dense, then the value of α is restricted to the interval of $[0.2, 0.5]$. The density of a data point x_i is defined as the number of points in its ϵ neighborhood. However, in the absence of any prior information about the cluster density and separation, one cannot set α according to these two unknown characteristics of the data. Moreover, each element's density is computed in terms of ϵ, and ϵ is defined as a function of α. To get out of this dilemma, the solution is quite simple: for each dataset repeat the algorithm for different α values, use

some validity criteria to compare the resulting solutions, and select only the best one. This methodology is adopted in the experiments. Note that clustering, like any other NP-Hard problem, can only be tackled using non-deterministic algorithms that provide approximate solutions, which can be acceptable and quite satisfactory. Usually, these solutions depend not only on the data but also on the algorithmic parameters, such as α and ϵ in our case. Therefore, it is the robustness of the algorithm over different structures in the data that is important, not its stability for limited structures.

The basic idea of DK-means is to select the point with maximum density, set it as the first initial centroid, and remove the data points that lie in its ϵ neighborhood. Repeat the process until k centroids are found and then apply the clustering process of traditional K-means to cluster data. DK-means improves the stability and accuracy of classic K-means but at the cost of execution time. The algorithm 2 gives the pseudo-code of the DK-means algorithm. It is evident that DK-means takes $O(n^2)$ time in the initialization step as it does not use any efficient data structures or techniques to compute the neighborhood and density of each data point [1]. Selection of k centroids and removal of high-density neighbor points for each selected centroid have $O(k * n)$ time complexity. Overall, the algorithm has $O(n^2)$ time complexity making it impractical for large real-world datasets.

Algorithm 2 DK-means Algorithm

Input: Dataset D, number of clusters K, neighborhood ϵ
Output: K clustering partitions

Initialization Step:

1: Calculate the density of each data point using the following formula and add it to set S.

$$\rho(x_i, \epsilon) = \sum_{j=1, j\neq i}^{n} Sgn(\epsilon, d(x_i, x_j)),$$

$$Sgn(x) = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

2: **for** $j = 1, 2, \ldots, K$ **do**
3: $c_j = \max(S)$ ▷ Select a point with maximum density from S and make it j^{th} initial centroid
4: Remove all high density data points in the neighborhood of c_j from S.
5: **end for**

Learning Loop:

6: same as K-means

3.1 EDK-Means Algorithm

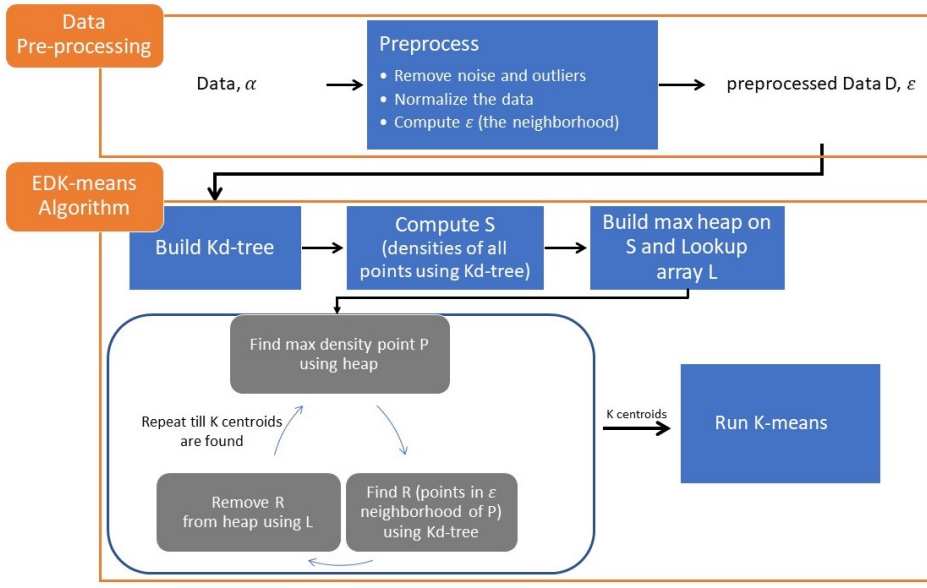


Figure 1. Workflow of EDK-means

EDK-means algorithm extensively relies on two data structures: KD-trees and max-heap to reduce the time complexity of DK-means. Figure 1 shows the workflow of the EDK-means algorithm. First, the data is pre-processed and the average distance between the data points is computed. During the pre-processing phase, the missing values, noise, and outliers are removed. Furthermore, the attributes are normalized using min-max normalization. This is beneficial, specifically when the attributes have very different ranges. Normalization ensures that no particular attribute dominates the clustering. Algorithm 3 shows some of the basic preprocessing steps. The neighborhood is defined as α times average distance. The value of α is set according to the characteristics of the dataset. We can speed up the preprocessing stage by calculating the average distance using a representative data sample rather than all the data points.

The pre-processed data and the ϵ (the neighborhood) are fed as input to the EDK-means algorithm. The neighborhood plays an essential role in discovering the density of a dataset. EDK-means algorithm constructs a KD-tree, also called as K-Dimensional Tree, on the given data. KD-tree is an advanced space partitioning data structure for organizing data points in a multidimensional space. It deploys the idea of a binary search tree to answer range queries efficiently. It is incredibly beneficial in our scenario for efficiently computing the density of each data point.

The density of a point is defined as the number of points in its ϵ neighborhood. With KD-tree, we find the density of each point in a specified neighborhood and store it in the set S. The algorithm constructs a max heap on the set S of densities and creates a lookup array that stores the location of each data point's density in the max heap.

EDK-means algorithm selects a data point with maximum density using max heap and sets it as an initial centroid. Then, it finds all points in ϵ neighborhood of the centroid using the KD-tree and removes the density of these points from the max heap using the lookup array. The algorithm repeats the above procedure till K initial centroids are found. After selecting the initial K-centroids, clusters are formed by assigning the data points to the nearest centroid according to the nearest distance formula, such as Euclidean distance. Update the cluster centers by taking the average of all points in the cluster. Repeat the clustering process until there is no change in any centroid.

Algorithm 4 outlines the pseudo-code of EDK-means algorithm. The algorithm gets the pre-processed data D, the number of clusters K, and neighborhood ϵ as input. It builds a KD-tree on given data in $O(n \log n)$, where n is the number of data points, and calculates the density of a data point using KD-tree. With KD-tree, we can count the data points in the ϵ radius of a specific point in $O(\log n)$ time. Thus, for loop in step 2 of Algorithm 4 takes $O(n \log n)$ time to compute densities of all points. In step 5 of the algorithm, a max heap on set S of densities is created along with the lookup array to store the location of each point's density in the max heap. This step takes $O(n)$ time. The algorithm extracts a maximum density point from the heap in $O(\log n)$ time and searches its neighborhood points using KD-trees in $O(\log n)$ time. Next, it removes density of the neighboring points from the heap and maintains heap order in $O(n)$ time. Thus, for loop on line 6-10 takes at most $O(Kn)$ time to identify K initial cluster centers. The overall time complexity of determining K centroid using EDK-means algorithm is $O(n \log n)$, while the DK-means takes $O(n^2)$ time for the same task. Algorithm 4 executes the basic K-means algorithm after determining the K-initial centroid to cluster the given dataset.

Algorithm 3 Data-Preprocessing

Input: Dataset, α the influence factor
Output: Preprocessed Dataset D, neighborhood ϵ

1: Handle missing and null values in data set
2: Remove noise and outliers
3: Normalize the data
4: Calculate *AvgDist*, the average distance between data points using Equation (2)
5: Set $\epsilon = \alpha \times AvgDist$

Algorithm 4 EDK-means Algorithm

Input: Preprocessed dataset D, number of clusters K, neighborhood ϵ
Output: K clustering partitions

KDtree = Build-KDtree(D)
▷ Calculate the density (no. of points in ϵ radius) of each data point using the KD-tree and store in set S
for each data point x in D **do**
    S($density_{\mathbf{x}}$) = KDtree.countPointsRadius(x, ϵ)
end for
Heap, L = Build-MaxHeap(S) ▷ Create a Max-Heap on set S and an lookup array L to store the location of each data-point's density in the Heap
for $j = 1, 2, \ldots, K$ **do**
    $c_j = extractMax(H)$
    R = KDtree.findPointsRadius(c_j, ϵ) ▷ Find data points in ϵ radius of centroid c_j using KD-tree
    Heap.Remove(R, L) ▷ Remove data points in R from Heap with help of lookup array L
end for
Get K initial centroids
repeat
    for each data point x in D **do**
        for $j = 1, 2, \ldots, K$ **do**
            Compute distance between x and c_j
        end for
        Assign x to nearest centroid
    end for
    for $j = 1, 2, \ldots, K$ **do**
        Update c_j to the average of all points assigned to it
    end for
until none of the centroid changes its value

4 EXPERIMENTS, RESULTS AND ANALYSIS

EDK-means provides a substantial speedup in time while maintaining the same clustering quality as the DK-means algorithm. The contribution of this study is two-fold: it significantly reduces the time taken by the recently proposed DK-means algorithm and rigorously evaluates the clustering quality of the density-based K-mean algorithm. We conducted computational experiments to compare the running time of EDK-means and DK-means algorithms on various real-world datasets. The experiments are performed on a machine with Intel dualCore™ i5 CPU, and code is developed in Python 3.6. We also carried out experiments to analyze the clustering quality generated by EDK-means and compare it with traditional K-means.

4.1 Data

The datasets are carefully selected from various sources [22, 23] to examine the time performance and accuracy of the EDK-means on the data with different specifications and properties. The details of the 22 selected datasets are given in Table 1. The selected datasets have a different number of sample points, attributes, and clusters. Some are chosen to observe the behavior of the algorithm on different cluster sizes and varying densities. Five datasets, namely, Unbalance, G2, S1, A3, and D15, are unsupervised and benchmark clustering datasets [22]. The rest of the datasets are supervised and obtained from UCI repository [23] except the dataset CC, which is acquired from Kaggle [24].

4.2 Evaluation Measure

Evaluating the quality of the clustering is a tough job and requires thorough investigation. It is quite a cumbersome task as compared to analyzing the results of a classifier. Different clustering indices exist to assess the clustering quality of supervised and unsupervised datasets. The indices like Adjusted Rand Index, Normalized Mutual Information, and Fowlkes-Mallows Index are used when class labels are known. On the other hand, measures such as Silhouette Coefficient, Calinski-Harabasz Index, and Davies-Bouldin Index are devised to assess the clustering results when no information regarding the ground truth is available. The evaluation measures used in this research work are briefly discussed below.

Adjusted Rand Index (ARI) calculates the similarity between predicted labels and actual labels, ignoring permutations and with chance normalization. ARI is computed as follows:

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI]} \tag{4}$$

where RI is the rand index and E[RI] is the expected rand index of clusters. The value of ARI ranges between -1 and 1 and 1 is the perfect match score.

Fowlkes-Mallows Index (FMI) is the geometric means of pairwise precision and recall. FMI is computed as follows:

$$FMI = \frac{TP}{\sqrt{(TP + FP)(TP + FN))}} \tag{5}$$

where TP, FP and FN stands for true positive, false positive and false negative, respectively. FMI values close to one indicate good clustering.

Normalized Mutual Information (NMI) measures the agreement between the predicted clusters X and actual classes Y. It ignores permutations but is not adjusted against chance. It uses mutual information (MI) and entropy (E)

to detect good quality of clusters. Values close to one indicate good clustering.

$$NMI(N,Y)) = \frac{MI(X,Y)}{meanE(X)E(Y)}. \tag{6}$$

Silhouette Coefficient (SCoff) is used when cluster labels are unknown. This measure calculates how well the clusters are defined, the formula is as follows:

$$s = \frac{b-a}{\max(a,b)} \tag{7}$$

where a is the mean distance between a data point and all other points in the same class and b is the mean distance between a data point and all other points in the nearest cluster. A higher Silhouette Coefficient score relates to a model with better-defined clusters.

Calinski-Harabasz Index (C-HI) is the ratio of the sum of between-clusters dispersion and of inter-cluster dispersion for all clusters. C-HI uses scatter matrix to assess the quality of cluster. It is defined as

$$CH\ index = \left(\frac{matrix(S_x)}{matrix(S_i)}\right)\frac{n-1}{n-k} \tag{8}$$

where n is the number of data points, k is the number of clusters, the $matrix(S\_i)$ is the scatter matrix of an inter-cluster, and $matrix(S\_x)$ is the scatter matrix of between-the-clusters. The higher score indicates good cluster quality, and the score is higher when clusters are dense and well separated.

Davies-Bouldin Index is used when cluster labels are unknown. It computes the average similarity of each cluster with respect to their centroids. It is defined as follows:

$$DB\ index = \frac{1}{k}\sum_{i=1}^{k}\max_{i\neq j}\left\{\frac{d(c_i)+d(c_j)}{d(C_i,C_j)}\right\} \tag{9}$$

where k is the number of clusters, $d(c_i)$ is the average distance of all the points from the centroid in cluster c_i and $d(C_i,C_j)$ is the distance between the centroids of the clusters. Zero is the lowest possible score. Values closer to zero indicate high quality clustering.

4.3 Time Performance of EDK-Means

Section 3, proves theoretically that the EDK-means algorithm significantly reduces the time taken by DK-means while maintaining the same cluster quality. This section presents the execution time comparison of DK-means and EDK-means algorithms for different datasets on a machine with Intel dualCore™ i5 CPU, and code is developed in Python 3.6. Table 1 shows the time taken by both the algorithms (in seconds) for different datasets.

Dataset	Size	Dimension	Optimal K	DK-Means Time	EDK-Means Time
Iris	150	4	3	0.25	0.11
Wine	178	13	2	1.15	0.40
Sonar	208	60	2	3.62	1.24
Flame	240	2	2	0.37	0.15
Ionosphere	351	34	2	9.87	2.51
Jain	373	2	2	0.96	0.35
Compound	399	2	6	1.45	0.85
Arrhythmia	452	278	16	354.00	66.68
Dress	500	13	2	10.51	2.48
Energy	768	9	12	23.39	9.04
Biodeg	1 055	41	2	122.23	15.64
Unbalance	1 281	2	8	42.53	3.90
G2	2 048	256	2	3286.36	1 048.21
D31	3 100	2	31	60.47	41.23
Abalone	4 177	8	3	929.30	257.90
S1	5 000	2	15	528.58	56.63
A3	7 500	2	50	975.27	133.67
CC	8 950	16	7	1 950.26	462.63
D15	10 126	15	9	4 810.79	1 570.01
Online	12 330	17	2	9 456.64	3 452.90
HTRU	17 898	8	2	12 980.45	4 164.74
TV	17 918	204	2	31 862.73	7 943.49

Table 1. Datasets and running time in seconds of DK-means and EDK-means

We compare the running time performance of EDK-means and DK-means on datasets with different sizes and dimensions. Figure 2 and Figure 3 compare the time taken (in seconds) by two algorithms on small and large datasets with many instances and dimensions. EDK-means algorithm has reduced the running time of DK-means almost by half in both scenarios. It is evident from Table 1 that EDK-means outperforms the DK-means algorithm on small as well as large and high dimensional datasets. As the number of dimensions increases, the time taken by both algorithms also increases. Consider the dataset Arrhythmia that consists of 452 instances and 279 attributes. DK-means algorithm takes a lot more time to cluster Arrhythmia than dataset Dress with 500 data points and 14 attributes. However, the time taken by the EDK-means algorithm for Arrhythmia is significantly less as compared to the DK-means. Similar behavior is observed for large datasets G2 and TV. Hence, we can conclude that advancement introduced in the EDK-means algorithm has made it practical for real-world datasets with many instances and attributes.

We examine the convergence rate of the EDK-means algorithm. It is the same as DK-means as both algorithms work on a similar principle to select initial cen-

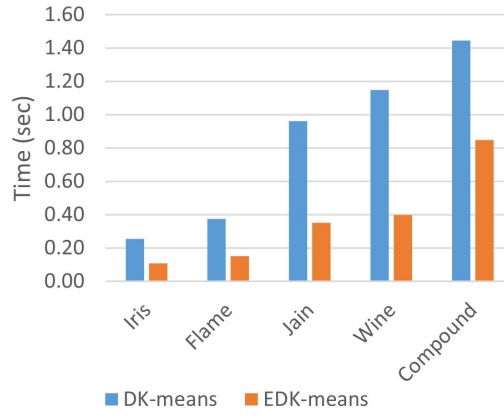


Figure 2. Time comparison for small datasets

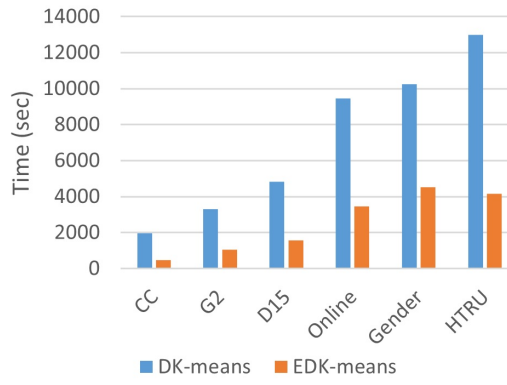


Figure 3. Time comparison for large datasets

troids. We recorded the number of iterations the EDK-means algorithm takes to converge to an optimal solution and compare it with traditional K-means. It is observed that mostly the EDK-means algorithm converges to the optimal solution quite fast as compared to K-means. The K-means algorithm randomly selects initial centroids, so it takes many iterations to converge in most runs. Moreover, K-means is not stable and takes a different number of iterations to converge for different runs. However, this is not the case for the EDK-means algorithm. We executed the K-means algorithm 100 times for each dataset and used the average number of iterations for comparison. Figure 4 shows the number of iterations taken by K-means and EDK-means for different datasets. In most cases, the EDK-means algorithm takes much fewer iterations to converge as compare to K-means. However, for some datasets like Iris and Abalone, K-means converges faster than EDK-means.

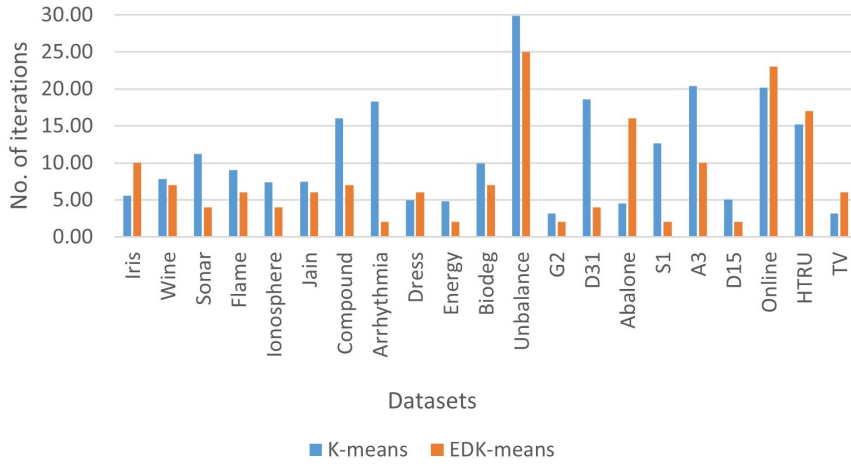


Figure 4. Convergence comparison between EDK-means and K-means

4.4 Clustering Quality Comparison of EDK-Means with Different Initialization Techniques

EDK-means maintains the same clustering quality as that of DK-means. In this section, we present the results of rigorous experiments performed on different datasets to examine the performance of EDK-means and compare it with the traditional K-means. The traditional K-means algorithm is executed 100 times and the average of all runs is used. For EDK-means different values of α are tried, and the best results are reported. Note that repeating the execution of K-means can be viewed as an alternative to using different initialization techniques and similarly using different alpha values in EDK-means gives us different initalizations. We compared the best result of EDK-means to the best result of K-means.

Different evaluation measures are used to evaluate the clustering quality for supervised and unsupervised datasets, as mentioned in Section 4.2. Table 2 shows the values of the α required in EDK-means for various datasets. The value of the alpha depends on the characteristics of the dataset at hand. In case we do not have much information about the dataset distribution, we can experiment with different values and select the best one.

Table 3 shows the results of the experiments on supervised datasets. It reports the values of three evaluation measures: FMI, ARI, and NMI. The results show that the EDK-means algorithm has outperformed the K-means for most of the datasets. For datasets such as Flame, Compound, Arrhythmia, Dress, and Online, the difference is enormous. For smaller datasets Iris, Wine, and Sonar, the performance of the algorithms is comparable. We analyzed the dataset D31 with two dimensions using plots to get insights. This dataset consists of the small dense clusters with small inter-cluster dispersion. Figures 5 a) and 5 b) show clusters generated by K-means

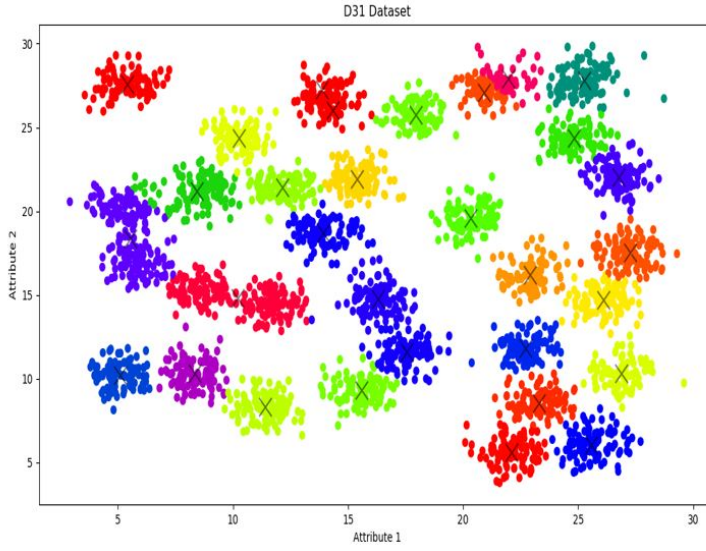
and EDK-means algorithms. It is clear from the figure that the K-means algorithm has merged few clusters with less separation and divided few other clusters into two. EDK-means, on the other hand, has recognized the clusters quite well. The datasets Flame, Sonar, Energy, and S1 have noise. From Table 3, it is observed that EDK-means is less susceptible to noise as compared to the K-means. In the Abalone and CC dataset, EDK-means achieves low-quality clusters compared to K-means, indicating that the EDK-means algorithm does not work well with datasets with sparse data points or maximum inter-cluster variance.

We conclude that EDK-means is stable and performs well for the datasets that have dense clusters or have clusters that are dense at the center. EDK-means merges the density concept with basic K-means to achieve the best of both worlds.

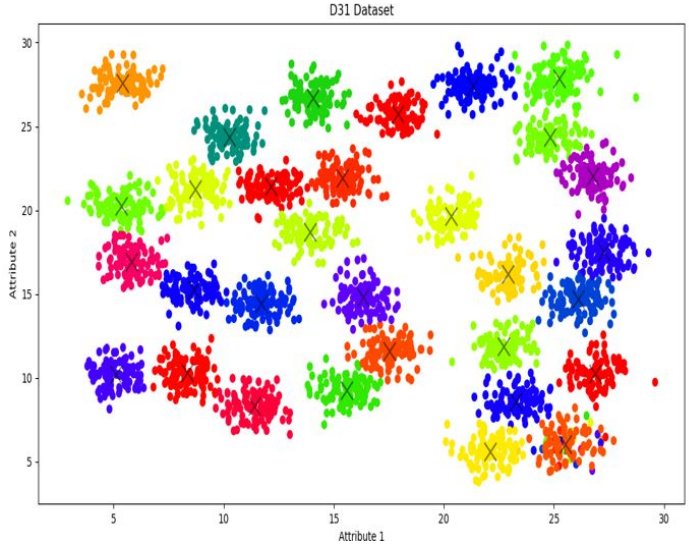
Datasets	**EDK-Means** α
Iris	0.15
Wine	0.27
Sonar	0.20
Flame	0.47
Ionosphere	0.04
Jain	0.08
Compound	0.15
Arrhythmia	0.28
Dress	0.01
Energy	0.05
Biodeg	0.01
D31	0.09
Abalone	0.03
CC	0.16
Online	0.03
HTRU	0.01
TV	0.15

Table 2. The best values of α for different supervised datasets

Analyzing the clustering quality of an unsupervised dataset is a tricky task as no prior information regarding class labels is available. We have used three measures: Silhouette Coefficient, Calinski-Harabasz Index (C-HI), and Davies-Bouldin Index (D-BI) to examine the cluster quality. Table 5 shows the values of the above three measures for five different unsupervised datasets. For a good clustering, the Silhouette Coefficient should be closer to 1. The higher values of C-HI and lower values of D-BI indicate better clustering performance. To determine the number of clusters in the unsupervised datasets, we have used VCVI, Variance-based cluster validity index [1]. VCVI uses the concept of variance to measure the dispersion of the data. It calculates the inter and intra cluster variance. The within-cluster compactness and between-cluster separation show the dispersion degree of clusters.



a) K-means



b) DK-means

Figure 5. Clustering of D31 dataset

Datasets	**K-Means FMI**	**EDK FMI**	**K-Means ARI**	**EDK ARI**	**K-Means NMI**	**EDK NMI**
Iris	0.82	0.86	0.73	0.75	0.76	0.78
Wine	0.58	0.61	0.37	0.40	0.43	0.47
Sonar	0.46	0.50	0.00	0.00	0.01	0.01
Flame	0.45	0.76	0.68	0.70	0.40	0.44
Ionosphere	0.61	0.64	0.14	0.17	0.13	0.15
Jain	0.70	0.70	0.32	0.32	0.37	0.35
Compound	0.64	0.81	0.74	0.88	0.72	0.80
Arrhythmia	0.25	0.43	0.07	0.10	0.24	0.20
Dress	0.52	0.65	0.00	0.01	0.00	0.01
Energy	0.90	0.97	0.89	0.97	0.96	0.97
Biodeg	0.59	0.64	0.00	0.00	0.00	0.00
D31	0.87	0.94	0.87	0.94	0.94	0.95
Abalone	0.45	0.43	0.12	0.09	0.11	0.11
CC	0.53	0.50	−0.05	−0.05	0.02	0.02
Online	0.70	0.81	0.08	0.08	0.02	0.02
HTRU	0.71	0.77	−0.08	−0.08	0.03	0.03
TV	0.58	0.64	0.02	0.54	0.03	0.62

Table 3. ARI, FMI and NMI values for K-means and EDK-means

Table 5 shows that EDK-means outperforms K-means for all the datasets. We plot graphs to examine the clusters generated by K-means and EDK-means for the S1 dataset (S1 has two dimensions). It is observed that the clustering of the K-means algorithm is not good as it merges two separate clusters and breaks a cluster into two parts. However, DK-means efficiently find all clusters as shown by Figure 6 a) and Figure 6 b).

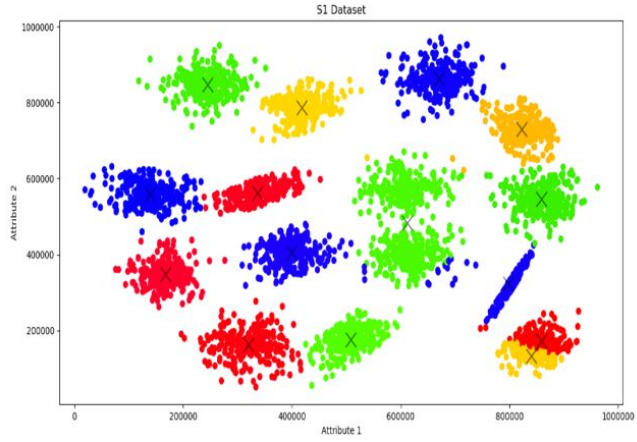
Datasets	**EDK-Means α**
Unbalance	0.05
G2	0.15
S1	0.05
A3	0.1
D15	0.05

Table 4. The best value of α for Unsupervised Datasets

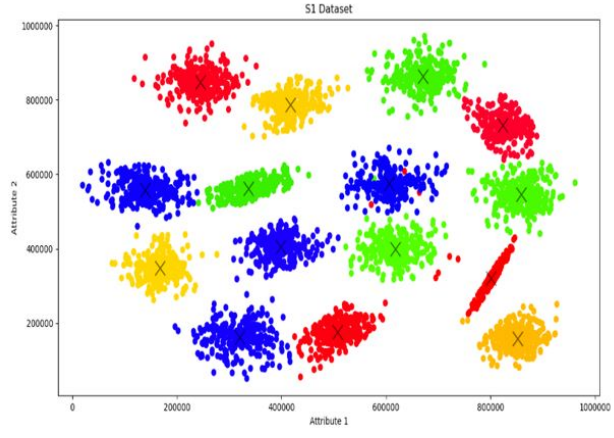
We examine the behavior of the EDK-means algorithm on datasets with different cluster sizes and densities. The datasets, namely, TV, HTRU, and Dress have clusters of varying sizes. Table 3 shows that EDK-means outperforms K-means for these datasets. Some of the supervised datasets given in Table 1 such as Jain, Compound, and Arrhythmia, have clusters of varying densities. The FMI value of EDK-means for Compound dataset is 0.81 while FMI of the K-means clustering for compound is just 0.64. Figure 7 a) and Figure 7 b) show the clustering of the

Datasets	K-Means Scoff.	EDK Scoff.	K-Means C-HI	EDK C-HI	K-Means D-BI	EDK D-BI
Unbal.	0.28	0.32	755	757	0.82	0.85
G2	0.19	0.21	634	637	1.76	1.79
S1	0.66	0.71	15 125	22 784	0.49	0.37
A3	0.54	0.58	16 937	19 907	0.68	0.59
D15	0.73	0.92	19 398	302 834	1.32	0.12

Table 5. Silhouette Coefficient (Scoff), C-HI and D-BI values for K-means and EDK-means

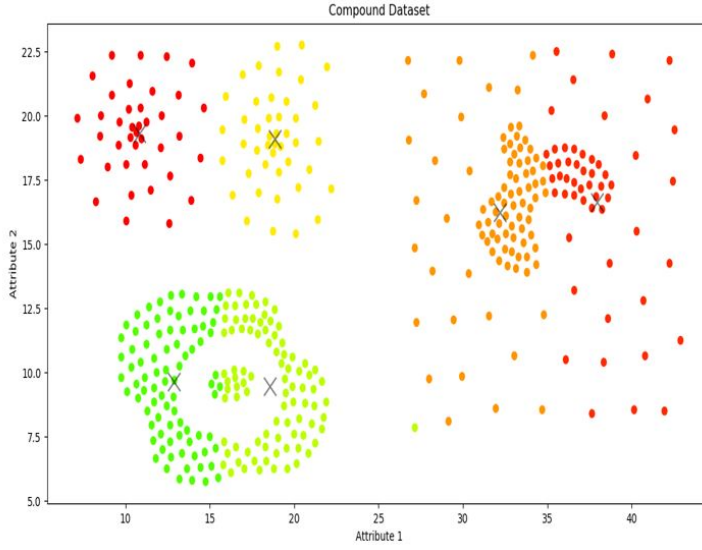


a) K-means

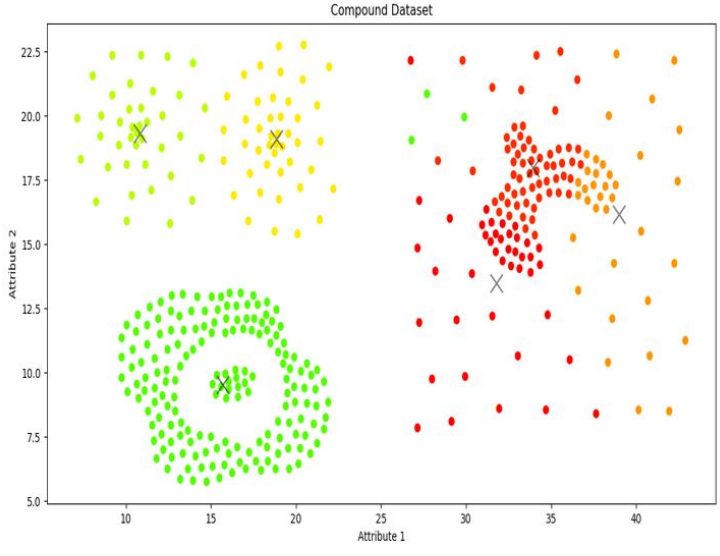


b) EDK-means

Figure 6. Clustering of S1 dataset



a) K-means



b) EDK-means

Figure 7. Clustering of Compound dataset

Compound dataset by K-means and EDK-means. It is observed that EDK-means cannot properly handle clusters of concave shapes and varying densities. However, it performs better than the traditional K-means.

5 CONCLUSION AND FUTURE WORKS

The clustering quality of the K-means clustering algorithm depends significantly on the selection of initial centroids. In this, we present an EDK-means algorithm that employs the concept of density to effectively find the initial centroids and used advanced data structures to reduce the computational time. The use of advanced techniques enhances the algorithm's speed and makes it feasible for large datasets with various features and instances.

We theoretically show that our algorithm is fast compared to the DK-means algorithm. We conducted computational experiments to evaluate our algorithm's runtime performance, convergence rate, and clustering quality on different datasets. It is observed that EDK-means have achieved higher accuracy compared to traditional K-means with different initializations. It can handle the dataset with different cluster sizes and is robust in the presence of noise. However, it cannot deal with the datasets with varying densities. As future work, we aim to modify the density parameters to handle datasets with varied densities. Furthermore, we aim to compare the EDK-means algorithm's performance with different variations of K-means such as K-means++, Min-Max K-means, and others on diverse and challenging datasets.

REFERENCES

[1] Zhu, E.—Ma, R.: An Effective Partitional Clustering Algorithm Based on New Clustering Validity Index. Applied Soft Computing, Vol. 71, 2018, No. 7, pp. 608–621, doi: 10.1016/j.asoc.2018.07.026.

[2] Aggarwal, S.—Singh, P.: Cuckoo, Bat and Krill Herd Based K-Means++ Clustering Algorithms. Cluster Computing, Vol. 22, 2019, No. 6, pp. 14169–14180, doi: 10.1007/s10586-018-2262-4.

[3] Arthur, D.—Vassilvitskii, S.: K-Means++: The Advantages of Careful Seeding. Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), 2007, pp. 1027–1035.

[4] Boudane, F.—Berrichi, A.: Gabriel Graph-Based Connectivity and Density for Internal Validity of Clustering. Progress in Artificial Intelligence, Vol. 9, 2020, No. 3, pp. 221–238, doi: 10.1007/s13748-020-00209-z.

[5] Ester, M.—Kriegel, H.-P.—Sander, J.—Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD '96), 1996, pp. 226–231.

[6] Fahad, A.—Alshatri, N.—Tari, Z.—Alamri, A.—Khalil, I.—Zomaya, A. Y.—Foufou, S.—Bouras, A.: A Survey of Clustering Algorithms for

Big Data: Taxonomy and Empirical Analysis. IEEE Transactions on Emerging Topics in Computing, Vol. 2, 2014, No. 3, pp. 267–279, doi: 10.1109/TETC.2014.2330519.

[7] Fränti, P.—Sieranoja, S.: How Much Can K-Means Be Improved by Using Better Initialization and Repeats? Pattern Recognition, Vol. 93, 2019, pp. 95–112, doi: 10.1016/j.patcog.2019.04.014.

[8] Hou, X.: A New Clustering Validity Index Based on K-Means Algorithm. Journal of Physics: Conference Series, Vol. 1187, 2019, No. 4, Art. No. 042040, doi: 10.1088/1742-6596/1187/4/042040.

[9] Kim, Y.—Shim, K.—Kim, M. S.—Lee, J. S.: DBCURE-MR: An Efficient Density-Based Clustering Algorithm for Large Data Using MapReduce. Information Systems, Vol. 42, 2014, pp. 15–35, doi: 10.1016/j.is.2013.11.002.

[10] Lakshmi, M. A.—Victor Daniel, G.—Rao, S. D.: Initial Centroids for K-Means Using Nearest Neighbors and Feature Means. In: Wang, J., Reddy, G., Prasad, V., Reddy, V. (Eds.): Soft Computing and Signal Processing. Springer, Singapore, Advances in Intelligent Systems and Computing, Vol. 900, 2019, pp. 27–34, doi: 10.1007/978-981-13-3600-3_3.

[11] Lattanzi, S.—Sohler, C.: A Better K-Means++ Algorithm via Local Search. Proceedings of Machine Learning Research (PMLR), Vol. 97, 2019, pp. 3662–3671.

[12] Li, Y.—Cai, J.—Yang, H.—Zhang, J.—Zhao, X.: A Novel Algorithm for Initial Cluster Center Selection. IEEE Access, Vol. 7, 2019, pp. 74683–74693, doi: 10.1109/ACCESS.2019.2921320.

[13] Li, Y.: Generalization of K-Means Related Algorithms. 2019, arXiv: 1903.10025.

[14] Ros, F.—Guillaume, S.: DENDIS: A New Density-Based Sampling for Clustering Algorithm. Expert Systems with Applications, Vol. 56, 2016 pp. 349–359, doi: 10.1016/j.eswa.2016.03.008.

[15] Saha, J.—Mukherjee, J.: CNAK: Cluster Number Assisted K-Means. Pattern Recognition, Vol. 110, 2021, Art. No. 107625, doi: 10.1016/j.patcog.2020.107625.

[16] Singh, P.—Meshram, P. A.: Survey of Density Based Clustering Algorithms and Its Variants. 2017 International Conference on Inventive Computing and Informatics (ICICI), IEEE, 2017, pp. 920–926, doi: 10.1109/ICICI.2017.8365272.

[17] Tzortzis, G.—Likas, A.: The MinMax K-Means Clustering Algorithm. Pattern Recognition, Vol. 47, 2014, No. 7, pp. 2505–2516, doi: 10.1016/j.patcog.2014.01.015.

[18] Yu, S. S.—Chu, S. W.—Wang, C. M.—Chan, Y. K.—Chang, T. C.: Two Improved K-Means Algorithms. Applied Soft Computing, Vol. 68, 2018, pp. 747–755, doi: 10.1016/j.asoc.2017.08.032.

[19] Mahajan, M.—Nimbhorkar, P.—Varadarajan, K.: The Planar K-Means Problem is NP-Hard. Theoretical Computer Science, Vol. 442, 2012, pp. 13–21, doi: 10.1016/j.tcs.2010.05.034.

[20] Ben-David, S.—Pál, D.—Simon, H. U.: Stability of K-Means Clustering. In: Bshouty, N. H., Gentile, C. (Eds.): Learning Theory (COLT 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4539, 2007, pp. 20–34, doi: 10.1007/978-3-540-72927-3_4.

[21] Bubeck, S.—Meilă, M.—von Luxburg, U.: How the Initialization Affects the Stability of the K-Means Algorithm. ESAIM: Probability and Statistics, Vol. 16, 2012, pp. 436–452, doi: 10.1051/ps/2012013.

[22] Fränti, P.—Sieranoja, S.: K-Means Properties on Six Clustering Benchmark Datasets. Applied Intelligence, Vol. 48, 2018, pp. 4743–4759, doi: 10.1007/s10489-018-1238-7.

[23] Dua, D.—Graff, C.: UCI – Machine Learning Repository. University of California, Irvine, School of Information and Computer Science. Available at: `http://archive.ics.uci.edu/ml`, 2020.

[24] Bhasin, A.: Credit Card Dataset for Clustering. Available at: `http://www.kaggle.com/plutosenthil/credit-card-dataset-for-clustering`, 2019.

Zareen Alamgir is Associate Professor at the Department of Computer Science, NUCES Lahore, Pakistan. Her research interests include data science, blockchain, data analysis, recommendation systems, and algorithms. She has written many articles in the field of data analysis and algorithms.

Hina Naveed has recently completed her Master degree in computer science and she is currently pursuing her Ph.D. Her research interests include data mining, machine learning and data science.