KDB-Means

Grace Bai, Aneesh Pabolu
January 20, 2026
Dr. Yilmaz, Pd. 4, Machine Learning 1
Quarter 2 Project

Abstract

The K-Means algorithm is a widely used unsupervised learning algorithm due to its simplicity and efficiency. However, a major flaw of a typical implementation of the K-Means algorithm is high sensitivity to initialization of centroids. Random initialization of centroids, which K-Means uses, can lead to unnecessary time spent on calculating centroid positions and can drastically change clustering results when the algorithm is retrained on the same dataset. The K-Means++ algorithm aims to address this issue by selecting k initial centroids that are well separated in the feature space. While K-Means++ improves centroid initialization by choosing centroids that are well separated, it remains unaware of local data density and might select centroids in sparse regions. Our algorithm, which we will refer to as KDB-Means (short for K-Density-Based-Means) aims to mitigate the issue of picking inefficient initial centroids by initializing centroids deterministically in dense regions.

1. Introduction

In recent years, there has been a significant increase in interest around the field of cluster analysis, or the classification of data into similar groups (clusters) based on their similarities [1]. Cluster analysis has many uses, from applications in customer interest analysis in marketing, to usage in medical diagnoses in healthcare, to risk assessment in finance [2]. An especially popular cluster analysis algorithm is the K-Means algorithm [3], an algorithm widely known as one of the strongest unsupervised learning algorithms. Its strengths lie particularly in its time efficiency, a clear, guaranteed convergence where the algorithm will always finish, and its interpretability. Due to these strengths, K-Means can separate data into groups effectively, and the simplicity of the algorithm allows for more widespread use.

However, the most inefficient part of the K-Means algorithm comes from the repeated trials and calculations necessary to move the centroids and effectively classify the data into clusters. At the beginning of the K-Means algorithm, k centroids are randomly selected, often from the existing data points. As the algorithm progresses, these centroids will move in each iteration, or trial, of the algorithm to minimize the total squared distances between each point and its nearest cluster centroid [2]. When the algorithm concludes, we are left with k clusters of data. If the number of centroid-moving trials can be reduced, we can significantly improve the efficiency of this algorithm. In fact, the time complexity of the original K-Means algorithm is

$$O(k \cdot n \cdot t),$$

where t is the number of iterations, k is the number of centroids/clusters, and n is the number of instances, or data points [4]. From this formula, we can see that the algorithm becomes significantly less efficient with an increasing dataset size. Thus, in this paper, we introduce a novel variation on K-Means to significantly minimize this weakness.

2. Related Work

In past years, there have been many improvements to the original K-Means algorithm, as well as significant advancements in alternative clustering methods.

In the area of improvements to the original algorithm, the most notable is K-Means++ [5]. As aforementioned, K-Means++ helps mitigate the issue of picking weak initial centroids by initializing far spread centroids through a distance-based probabilistic seeding strategy. However K-Means++'s main drawback is not being able to take local data density into account when initializing centroids.

An algorithm that explicitly takes local data density into account when initializing centroids is the KNN-DP (k-Nearest Neighbor Density Peaks) [6]. While effective at locating density peaks, KNN-DP doesn't optimize the K-Means objective function and lacks refinement steps, limiting clustering accuracy.

Density-based clustering methods offer an alternative by explicitly modeling dense regions of the dataset. DBSCAN [7], one such density-based clustering method, is a popular algorithm that groups points based on neighborhood density and identifies noise without requiring a predefined number of clusters. Although DBSCAN performs well on datasets with arbitrary cluster shapes, it is highly sensitive to parameter selection and doesn't create centroid-based clusters, making it less suitable for applications requiring centroid refinement or datasets with high dimensionality.

Our algorithm, KDB-Means, aims to address these limitations by taking advantage of local data density when initializing centroids while retaining iterative refinement through the K-Means objective.

3. Dataset

We are using the User Knowledge Modeling (UNS) Dataset [8] from UCI Machine Learning Repository for performance analysis. The dataset is about students' knowledge status on Electrical DC Machines. The dataset has 5 features and 403 instances. Each instance represents one student. Each feature (STG, SCG, STR, LPR, and PEG) has continuous values that span the scale from 0 to 1. The feature STG (Study Time for Goal Object) represents the degree of time a user spends studying the target subject. SCG (The degree of repetition number of user for goal object materials) measures the frequency of repetition with which the user studies the target subject. This is reflective of how often a student revisits and practices the subject material. STR (Study Time for Related Objects) represents the degree of time the user spends studying subjects related to Electrical DC Machines. LPR (Performance for Related Objects) represents how the user has been performing on

exams in subjects related to Electrical DC Machines. PEG (Performance for Goal Object) measures the user's exam performance in the Electrical DC Machines subject. The class label, UNS, is an ordinal categorical variable with the following labels: very_low, Low, Middle, High. A student with a UNS of "very_low" means the student has very low knowledge on the target subject (Electrical DC Machines). A student has low knowledge on the target subject if their UNS is "low". A student has average knowledge on the target subject if their UNS is "Middle". A student has above average knowledge on the target subject if their UNS is "High".

The User Knowledge Modeling dataset originally came with a training dataset with 258 instances and a testing dataset with 145 instances. Because KDB-Means is an unsupervised learning algorithm, we combined both the original training and testing datasets into one single training dataset.

The original testing dataset contains inconsistencies in the capitalization of UNS class labels, where the value of "very_low" in the original training dataset was represented as "Very Low" in the original testing dataset. During preprocessing, these labels were standardized and encoded as an ordinal categorical variable with the following mapping: "very_low" = 0, "Very Low" = 0, "Low" = 1, "Middle" = 2 and "High" = 3. It is important to note that the UNS class label was not used during clustering, and that these preprocessing steps were taken only to make the class label easier to work with for evaluation. Aside from this standardization, no additional preprocessing steps were applied to the dataset.

Prior to clustering, all continuous features were normalized to prevent attributes with larger variances from dominating distance and density computations. The normalization method we used and its underlying purpose in our algorithm is discussed in detail in the Methodology section (Section 4).

In addition to this dataset, we also created a number of toy (example) datasets for visualization purposes. These datasets were created using Scikit-Learn's make_blobs[11] function, where the dataset had a random number of instances between 100 and 500 (inclusive) as well as a random number of clusters between 3 and 10 (inclusive).

4. Methodology

In this paper, we propose KDB-Means, a novel variation on K-Means. The core idea of our algorithm is to bias centroid initialization towards points that are well separated and representative of high-density regions. This is achieved by combining a local density estimate with distance-based separation. Our algorithm works especially well on unevenly distributed data, where K-Means and K-Means++ often show weaknesses.

In order for each feature to be represented equally with our density-based approach, all features are normalized using z-score normalization with this formula:

$$Z = \frac{(X-\mu)}{\sigma},$$

where Z represents the standardized data point, X represents the raw data point, μ represents the average of that feature across the entire dataset, and σ represents the standard deviation, or data spread, of that feature across the dataset. By performing z-score normalization on our data, all our features will have a mean value of 0 and a standard deviation of 1, making our density calculations more even by preventing features with different scales from being weighted differently.

Once the features are normalized with the z-score normalization method, the density radius r_0 is calculated for each node (instance). In order to do this, we first compute the measure of data spread across the entire dataset by finding the average distance between each pair of nodes. We then multiply this average distance value by a predefined value α = 0.2, the optimal value determined from trial-and-error from α = 0 to α = 1. Then, using a NearestNeighbors method imported from Scikit-Learn, the full dataset is stored to record neighbors for each node within the circle of radius r_0, and the indices of each node's neighbors are returned. The number of a node's neighbors in the circle of radius r_0 thus represents that node's density, ρ.

Once the densities for all nodes are calculated, we then select the node with the highest density value as our first centroid. After we select the first centroid, we then compute a value for each remaining node to determine the best placements for the following centroids using this centroid-selection formula:

$$\rho \cdot d^2,$$

where ρ is the calculated density value of that node, and d represents the Euclidean distance from that node to the nearest centroid, where Euclidean distance is calculated with this classic formula

$$\sqrt{\sum_{i=1}^{n} (p_i - q_i)^2},$$

where p_i and q_i represent feature values for two different nodes, and n represents the number of features, thus taking the square root of the summed squared distance between two nodes for each feature.

Our centroid-selection formula is a variant on the K-Means++ formula, which calculates the probability of a node being picked as the next centroid with

$$d^2,$$

where d similarly represents the distance from that node to the nearest centroid. However, unlike the K-Means++ algorithm, our centroid-selection algorithm is completely deterministic in order to eliminate the possibility of drastically different outcomes with different initializations. The next centroids are then picked by selecting the node with the max value from our formula, where the formula values are re-calculated each centroid selection to account for updated distances to the nearest centroids, until all k centroids are initialized. This method allows for the optimal selection of centroids, where initial positions are far from

existing centroids but also in high-density areas, thus allowing for a reduced number of centroid-moving iterations for clustering convergence through the typical K-Means algorithm. After this initialization, the typical K-Means algorithm proceeds with these pre-selected centroids using the Euclidean formula for distance.

We then evaluate the performance of each algorithm after convergence using a few metrics, which are the Adjusted Rand Index (ARI):

$$ARI(U, V) = \frac{2(N_{00}N_{11} - N_{01}N_{10})}{(N_{00}+N_{01})(N_{01}+N_{11})+(N_{00}+N_{10})(N_{10}+N_{11})}$$ [9],

where the ARI between the partial clusterings U and V is calculated with N_{00}, the number of pairs in different clusters in U and V, N_{01}, the number of pairs in the same cluster in U but different in V, N_{11}, the number of pairs in the same cluster in U and V, and N_{10} and N_{01}, disagreement indicators, the Adjusted Mutual Information (AMI),

$$AMI(U, V) = \frac{I(U,V) - E\{I(U,V)\}}{max\{H(U),H(V)\} - E\{I(U,V)\}}$$ [9],

where I(U,V) represents the mutual information of U and V, E is the expected value, and H is entropy, and the Silhouette Score:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i),b(i)\}}$$ [10],

where s(i), the Silhouette score, is calculated with a(i), the average distance between the node i and all the points in its own cluster, and b(i), the distance between i and the next nearest centroid. The ARI and AMI are external validation formulas [10], where the actual labels of the dataset are used to calculate the goodness value, while the Silhouette Score evaluates the formed cluster quality in an unsupervised way without the dataset labels [10]. In addition to these metrics, we also record the number of K-Means iterations required for each algorithm (KDB-Means, K-Means, K-Means++) to converge. The ARI measures the similarity between the predicted values through the clustering algorithm and the true labels by checking whether they're in the same clusters and ranges from -1 (random) to 1 (perfect). The AMI similarly measures likeness between predicted values and labels, but measures mutual dependence through shared information instead and varies from 0 (independence between the predictions and labels) to 1 (high agreement) [9]. Lastly, the Silhouette Score varies from -1 (very bad cluster formation; nodes are closer to other centroids than their own) to +1 (very good cluster formation) [10].

5. Experimentation

The above procedure was completed for both the toy datasets for visualization and the larger User Knowledge dataset. The toy datasets were used mainly for visualization purposes with 2D cluster graphs for each algorithm, since the lower dimensionality meant significant computational power was saved as compared to the User Knowledge dataset. With the toy datasets, 25 trials were performed. In each trial, the following occured:

1. A toy dataset was generated with the random parameters described above.
2. The methodology described above was performed, and each of the three algorithms (KDB-Means, K-Means, K-Means++) was executed with this dataset.
3. The performance metrics (ARI, AMI, Silhouette Score, and number of iterations) were recorded for each algorithm.
4. Visualizations of the trials were recorded for presentation purposes, which included the visualization of the original cluster formation, initial centroid instantiation, and final centroid and cluster results for each algorithm, as well as a Silhouette plot comparing the cluster formation resulting from each algorithm.

With the User Knowledge dataset, the methodology described above was performed, and the performance metrics were recorded. Due to the higher-dimensional nature of the User Knowledge dataset, no visualizations were created, as the toy dataset visualizations provided much more understandable and communicable results.

6. Results

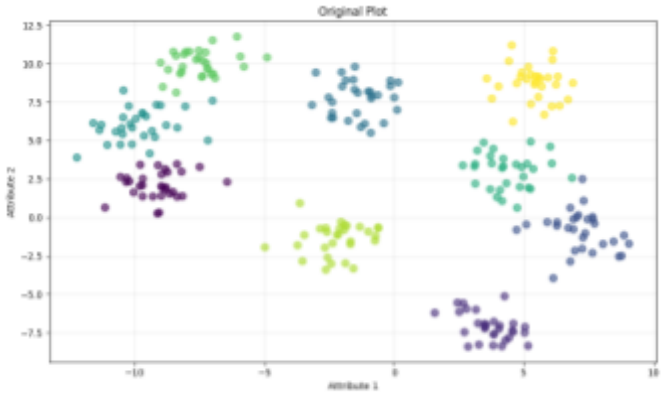

Fig. 1: Initial cluster formation in a 2D toy dataset with # nodes=271, k=9.

Fig. 2: DBK-Means initial centroid positions.

Fig. 3: DBK-Means final centroid positions with final clusters.

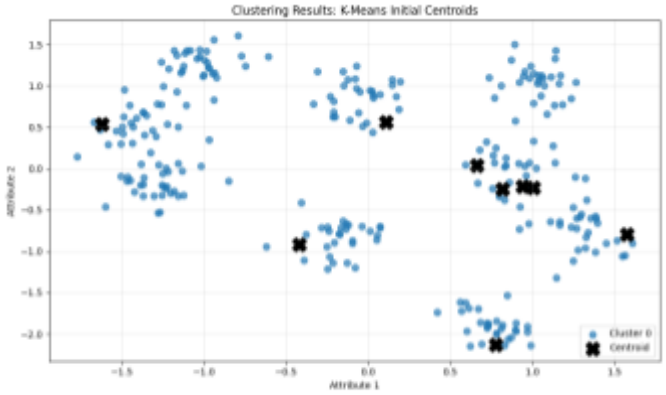

Fig. 4: K-Means initial centroid positions.

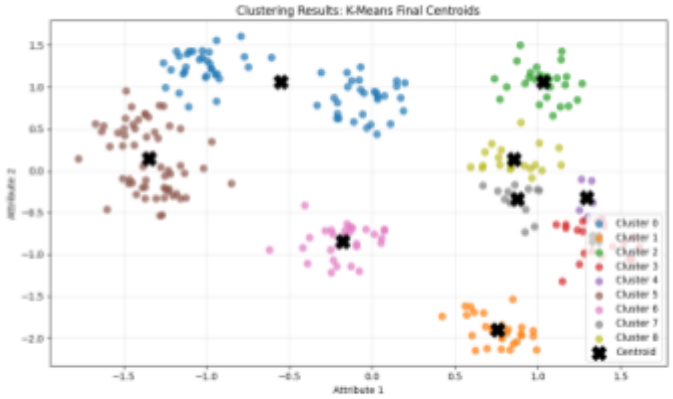

Fig. 5: K-Means final centroid positions with clusters.

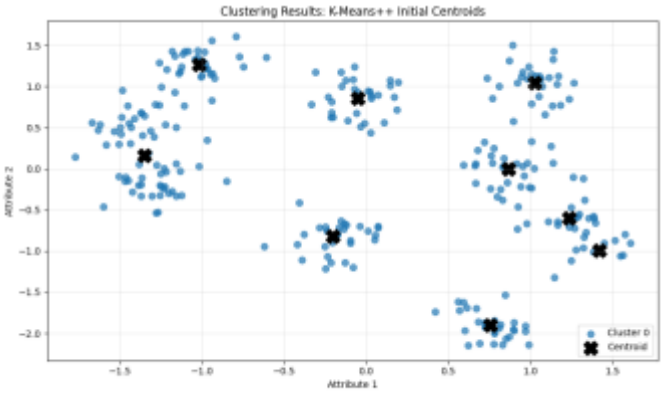

Fig. 6: K-Means++ initial centroid positions.

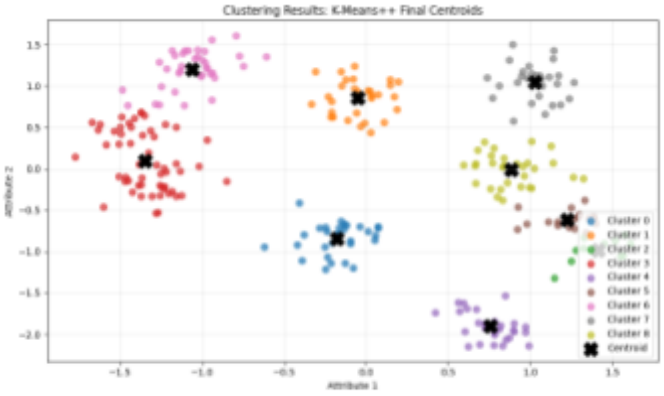

Fig. 7: K-Means++ final centroid positions with final clusters.

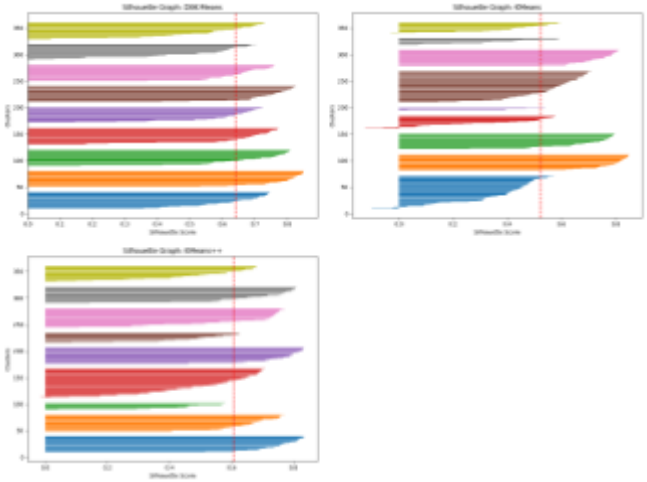

Fig. 8, 9, 10: Silhouette graphs of DBK-Means (top left), K-Means (top right), K-Means++ (bottom left)

As we can see from *Figs*. 2-7, the centroid initialization in DBK-Means from a toy dataset is significantly better than that of K-Means, and moderately better than that of K-Means++. Additionally, as we can see from *Figs*. 8-10, the Silhouette Scores of DBK-Means generally are better than that of K-Means and K-Means++, where the tall, uniform bars produced by the DBK-Means algorithm represent well-separated clusters.

	KDB-Means	K-Means	K-Means++
AMI	0.2930	0.2225	0.2217
ARI	0.2108	0.1579	0.1556
Silhouette	0.1755	0.1703	0.1697
Iterations	12.0000	18.8400	16.2000

Fig. 11: Performance metrics for the algorithms from User Knowledge dataset, averaged over 25 trials.

From *Fig. 11*, we can observe that KDB-Means also consistently outperforms K-Means and K-Means++ from the AMI, ARI, Silhouette Score, and number of iterations. Averaged over 25 trials on the User Knowledge dataset, KDB-Means' AMI and ARI values are higher than that of K-Means and K-Means++, illustrating a strong agreement between the predicted clusters of KDB-Means and the actual labels. Additionally, the higher average Silhouette Score shows more well-formed and cohesive clusters.

7. Conclusion and Future Work:

From this study, we can conclude that the integration of density weighting to the K-Means algorithm through KDB-Means is effective and beneficial to the efficacy of the algorithm itself through better-formed clusters and decreased convergence time. By combining our novel density weighting with existing clustering algorithms, we develop a more efficient and accurate algorithm. It can be noted, however, that KDB-Means will work better in lower dimensions due to the nature of the density calculations within a n-dimensional circle of radius r_0, where an increased number of dimensions will work less well. KDB-Means does outperform the existing clustering methods of K-Means and K-Means++, however, in both toy and real-world datasets. Additionally, KDB-Means will likely work better with datasets of unequal density and will outperform other clustering algorithms due to its density-based nature, whereas typical clustering algorithms such as K-Means and K-Means++ would struggle to produce clusters due to their inherent density discrepancies.

In the future, other applications can be considered with variants on K-Means or DBSCAN. In addition, the parameter α can be tuned to better fit datasets of high-dimensionality to fix the weaknesses of KDB-Means surrounding datasets with many features. To conclude, this study has proved the usefulness and potential of the intersection between density-based algorithms and the classic K-Means clustering algorithm through the better overall performance of KDB-Means.

8. Contributions

The code, figures, graphs, and parts 1, 4, 5, 6, & 7 of the paper were written and created by Grace Bai. The presentation, abstract, and parts 2 & 3 of the paper were written and created by Aneesh Pabolu.

References

[1] Henry, David, et al. "Clustering Methods with Qualitative Data: A Mixed-Methods Approach for Prevention Research with Small Samples." *Prevention Science*, vol. 16, no. 7, 7 May 2015, pp. 1007-16, https://doi.org/10.1007/s11121-015-0561-z.

[2] Shutaywi, Meshal, and Nezamoddin N. Kachouie. "Silhouette Analysis for Performance Evaluation in Machine Learning with Applications to Clustering." *Entropy*, vol. 23, no. 6, 16 June 2021, p. 759, https://doi.org/10.3390/e23060759.

[3] Hartigan, J. A., and M. A. Wong. "Algorithm as 136: A K-Means Clustering Algorithm." Applied Statistics, vol. 28, no. 1, 1979, p. 100, https://doi.org/10.2307/2346830.

[4] Zhao, YanPing, and XiaoLai Zhou. "K-Means Clustering Algorithm and Its Improvement Research." Journal of Physics: Conference Series, vol. 1873, no. 1, 1 Apr. 2021, p. 012074, https://doi.org/10.1088/1742-6596/1873/1/012074.

[5] Arthur, David, and Sergei Vassilvitskii. "k-means++: The Advantages of Careful Seeding." Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035, https://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf

[6] Liao, Jiyong, et al. "K-NNDP: K-Means Algorithm Based on Nearest Neighbor Density Peak Optimization and Outlier Removal." Knowledge-Based Systems, vol. 294, June 2024, p. 111742, https://doi.org/10.1016/j.knosys.2024.111742.

[7] Ester, Martin, et al. "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), AAAI Press, 1996, pp. 226–231, https://ui.adsabs.harvard.edu/abs/1996kddm.conf..226E

[8] "User Knowledge Modeling." UC Irvine Machine Learning Repository, 13 July 2013, archive.ics.uci.edu/dataset/257/user+knowledge+modeling.

[9] Vinh, Nguyen Xuan, Julien Epps, and James Bailey. "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance." Journal of Machine Learning Research, vol. 11, no. 95, 2010, pp. 2837–2854, http://jmlr.org/papers/v11/vinh10a.html

[10] Rousseeuw, Peter J. "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis." Journal of Computational and Applied Mathematics, vol. 20, Nov. 1987, pp. 53-65, https://doi.org/10.1016/0377-0427(87)90125-7.

[11] Pedregosa, Fabian, et al. "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, vol. 12, Oct. 2011, pp. 2825-30, jmlr.org/papers/v12/pedregosa11a.html

[12] Hunter, J. D. "Matplotlib: A 2D Graphics Environment." Computing in Science & Engineering, vol. 9, no. 3, 2007, pp. 90-95, https://doi.org/10.1109/MCSE.2007.55
[13] The pandas development team. pandas-dev/pandas: Pandas. Zenodo, Feb. 2020, https://doi.org/10.5281/zenodo.3509134
[14] Virtanen, Pauli, et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." Nature Methods, vol. 17, no. 3, 2020, pp. 261–72. Nature Methods. https://doi.org/10.1038/s41592-019-0686-2
[15] Harris, C. R., K. J. Millman, S. J. van der Walt, et al. "Array programming with NumPy." Nature, vol. 585, no. 7825, 2020, pp. 357–62. Nature, https://doi.org/10.1038/s41586-020-2649-2