Coursework 1:

An efficient approximation to the K-means clustering for massive data

Author:

González Béjar, Javier

Unsupervised Learning 2023/24

Master's degree in Artificial Intelligence

Barcelona, May 6, 2024

Contents

1 Introduction

This work consists on the implementation of the algorithm described in the paper **An efficient approximation to the K-means clustering for massive data**[1], by Marco Capó, Aritz Pérez and Jose A.Lozano. The algorithm proposes an iterative approximation strategy for the K-means, that is based on a sequence of recursive thinner partitions of the dataset, to address the computational and resource-intensive demands of applying the original K-means clustering technique to very large datasets.

2 Algorithm Description

The Recursive Partition-Based K-Means (RPK-means) algorithm introduced in the paper is an innovative approach to clustering that combines recursive data partitioning with an improved K-means clustering process to handle large and complex datasets efficiently. The idea behind it consists in recursively applying a weighted version of the K-means algorithm over a growing, yet small, number of subsets of the dataset. Each of these subsets is represented by a representative sample (center of mass) as well as its corresponding weight (cardinality).

In the approach, the authors introduce a method to handle clustering by initially dividing the dataset's space using a recursive quadtree structure. This structure splits the space into 2^d subspaces; for example, in a 2D space, the area would be segmented into four quadrants. This division process is iterative, where each quadrant is subsequently divided into smaller subquadrants, effectively managing the volume of data by ensuring the number of elements in each partition ($|P|$) remains smaller than the total number of instances in the dataset (n).

Each partition generated through this recursive splitting is then processed using a weighted Lloyd (WL) algorithm. The WL algorithm refines cluster centers iteratively: it starts with a set of initial centroids and recalculates them by considering the representative samples of each partition and their cardinalities. The refined centroids from each iteration then serve as initial centroids for the next level of partitioning, significantly reducing the computational the distance calculations required.

The operational flow of the WL algorithm, as described in the paper, is as follows:

Algorithm 1 Weighted Lloyd (WL)

Input: Set of representatives $\{S\}_{\text{sep}}$ and weights $\{|S|\}_{\text{sep}}$, for the partition P. Number of clusters K and initial set of centroids C_0.
Output: Set of centroids C_r and corresponding clustering pattern g_r.
Step 0 (Initial Assignment):
$g_0 \leftarrow C_0$; $r \leftarrow 0$.
while not StoppingCriterion **do**
    $r \leftarrow r + 1$;
    Step 1 (Update Step):
    $C_r \leftarrow g_{r-1}$; ▷ Clustering error $E_p(g_{r-1})$ (Eq. 4)
    Step 2 (Assignment Step):
    $g_r \leftarrow C_r$; ▷ Centroid error $E_p(C_r)$ (Eq. 2)
end while
return C_r and g_r;

1. **Initialization**: Input the set of representatives and their respective weights for the initial partition, along with the number of desired clusters and initial centroids.
2. **Iterating loop**: Update the centroids by computing the weighted mean of clusters based on the current partition's representative data points.
3. **Convergence Check**: The algorithm iterates until a stopping condition is met, which could be based on the convergence of centroids or reaching a maximum number of iterations.

Then, the complete RPKM algorithm, uses this approach across multiple recursive partitions:

Algorithm 2 RPKM algorithm

Input: Dataset D, number of clusters K, maximum number of iterations m.
Output: Set of centroids approximation C_i.
Step 1 Compute the set of weights and representatives of the sequence of thinner partitions, $P_1, \ldots, P_m$, backwards.
Set $i = 1$.
while not Stopping Criterion **do**
    Step 2 Update the centroid's set approximation,
    $C_i = \{c_{ij}\}_{j=1}^{K} : C_i = \text{WL}(\{S\}_{\text{sep}_i}, \{|S|\}_{\text{sep}_i}, K, C_{i-1})$
    $i = i + 1$
end while
return C_i;

1. **Initialization**: Initialize by determining the weights and representatives for a sequence of increasingly finer partitions.
2. **Iterative Refinement**: For each partition, update the centroid approximation by applying the

WL algorithm using the representatives and weights of the current partition and the centroids from the previous iteration.
3. **Convergence Check**: The process repeats until a stopping criterion, which might be a set number of iterations or a minimal change in centroid positions, is satisfied.

This method not only improves efficiency by reducing the number of distance computations needed but also leverages the hierarchical structure of data, potentially leading to better clustering by focusing on increasingly localized data features. Such a technique is especially advantageous in dealing with large datasets where traditional clustering algorithms might falter due to computational constraints or memory limitations.

3 Implementation

The algorithm is implemented in Python 3.10 following scikit-learn's API conventions.

The implementation consists in two main classes: **Subset** and **RecursivePartitionBasedKMeans**. The Subset class encapsulates the functionality required to manage subsets of the dataset partitions, including keeping their data indices, the maximum and minimum range for the dimensions, and threshold values for each subset. On the other hand, the RecursivePartitionBasedKMeans class implements the core functionality of the recursive partition-based k-means algorithm itself.

3.1 Subset Class

The class Subset class encapsulates the following:

1. **Indexes**: An array of integers representing the indices of the data points within the subset. This allows the subset to specifically identify and access its corresponding elements within the larger dataset.

2. **Maximum Values**: A NumPy array containing the maximum boundary values for the subset. These values define the upper limits of the subset in each dimension of the data space.

3. **Minimum Values**: A NumPy array containing the minimum boundary values for the subset. These values define the lower limits of the subset in each dimension, thereby helping in defining the bounding box of the subset.

4. **Thresholds**: A NumPy array of threshold values used for partitioning the dataset in the Recursive Partition-Based K-Means algorithm. These thresholds help in deciding the boundaries for further subdivision of the data.

5. **Representative Computation**: The class provides a method to compute the representative or mean of the data points within the subset, assuming the subset is non-empty. This mean serves as the centroid of the subset for the purposes of clustering analysis.

6. **Length Calculation**: A method to determine the number of indices or data points within the subset, for the assessment of the subset's size.

This class structure ensures that each subset is self-contained with all necessary information to perform its role in the partitioning and clustering process, facilitating efficient and effective data analysis in large-scale clustering tasks.

3.2 RecursivePartitionBasedKMeans Class

The Recursive Partition-Based K-Means (RPKM) algorithm is encapsulated within a class that inherits from scikit-learn's 'BaseEstimator', 'ClassifierMixin', and 'ClusterMixin' classes. The 'BaseEstimator' acts as the core class for all scikit-learn estimators, offering functionalities essential for managing estimator parameters and ensuring a consistent interface across different estimators. Their methods include 'set_params()' for setting parameters, 'get_params()' for retrieving them, '__repr__()' for generating a string representation, and 'validate_data(X)' for ensuring data is correctly formatted and managing feature-related attributes.

Additionally, the 'ClassifierMixin' improves functionality for classifiers by providing the 'score(X, y)' method, which predicts labels for data 'X' and calculates accuracy against the true labels 'y'. For clustering algorithms, the 'ClusterMixin' contributes the 'fit_predict(X, y=None)' method, which fits the estimator to the data and returns cluster labels.

We will explore now the specific implementation of the RPKM algorithm's 'fit' method, particularly how it methodically partitions the feature space into hypercubes as per the algorithm's design.

3.2.1 Fit

The 'fit' method in the RPKM algorithm is the central mechanism for executing the clustering process. It begins by validating the input dataset X to ensure it conforms to a two-dimensional structure and logs the total number of samples. Initially, it resets all internal state variables and sets up the first broad partition of the dataset using generic maximum, minimum, and threshold values. As the method progresses through each iteration, it refines these partitions by generating increasingly thinner sub-partitions based on the prevailing centroids and the distribution of the data. If centroids have not previously been established, the method randomly selects initial centroids from the dataset representatives. These centroids are then iteratively refined using sklearn KMeans algorithm, which considers the size of each subset for weighting purposes. The iterative process continues until the centroids reach a stable condition where changes between iterations fall below a specified threshold or until the predefined maximum number of iterations is achieved.

In order to build the partitions on the dataset, 'fit' uses some supporting methods. The 'create_recursive_partition' method is designed to iteratively refine the clustering granularity by subdividing existing partitions into more specific sub-partitions using the 'perform_partition' method. This method evaluates each data point within a partition against predefined thresholds, assigning them to new subgroups that better reflect their properties, thus increasing the granularity of the clustering. Following this, the 'generate_sub_partitions' method organises these data points into new partitions based on their hypercube indices derived from these comparisons, adjusting the maximum and minimum boundaries of each partition to encapsulate its data points more accurately. Lastly, the 'update_ranges' method fine-tunes these partitions by adjusting their boundaries—maxima and

minima—based on the binary outcomes of these partitioning processes, ensuring each new subset is optimally positioned within the feature space.

3.2.2 Predict

The 'predict' method, on the other hands, is used for applying the learned clustering model to new data. It assigns each sample in a given dataset X to the closest cluster that was identified during the model's training phase. Operationally, this method computes the Euclidean distance between each sample in X and all established centroids. Based on these distances, each sample is then assigned to the centroid it is closest to. This process not only categories new data based on the learned clustering patterns but also allows for the practical application of the clustering model to organise, classify, or label new, unseen datasets in an effective way.

4 Experimental Section

In this section, we replicate the set of experiments performed in the original paper, demonstrating the algorithm's effectiveness and efficiency. In total, six experiments were conducted, three of which used artificial datasets, which were then repeated using real datasets.

The artificial datasets were generated using the artificial dataset generator class as a d-dimensional mixture of K Gaussians. Specifically, we set $K \in \{3, 9\}$, $d \in \{2, 4, 8\}$, and $n \in \{1000, 10000, 100000, 1000000\}$. For each configuration, we generated 10 replicates of the dataset, with component overlapping lower than 5%.

In addition to these experiments, we evaluated the performance of the grid-based RPKM algorithm, KM++, and MB on a real-world dataset: the gas sensor array under dynamic gas mixtures dataset. This dataset contains time series data from 16 chemical sensors exposed to gas mixtures at varying concentration levels and includes 4,178,504 instances with 19 attributes, available in the UC-Irvine Machine Learning Repository. Using this real-world dataset, we generated various subsamples to analyze the algorithm features. We selected $d \in \{2, 4, 8\}$ random attributes and $n \in \{4000, 12000, 40000, 120000, 400000, 1200000, 4000000\}$ random instances, with the number of clusters set to $K \in \{3, 9\}$. For each setting, we generated 10 replicates of the dataset.

Three main experiments were performed with each dataset: the computation of distance comparisons, the standard error experiment, and the instance ratio experiment.

4.1 Distance Computations

The initial experiment conducted was the Distance Computations experiment, which involved evaluating the RPKM algorithm against scikit-learn's KMeans initialized with k-means++ and scikit-learn's MiniBatchKMeans with batch sizes of 100, 500, 1000. RPKM was tested for iterations ranging from 1 to 6. This experiment was replicated for both artificial and real datasets. For the artificial datasets, distance computation values were averaged over 10 replicates, with each replicate being a randomly generated dataset with a consistent number of clusters K, dimensions D, and instances N. For real datasets, 10 replicates involved randomly selecting D features and N instances

from the dataset.

The number of distance computations for the RPKM algorithm, d_{RPKM}, is calculated by summing the products of the number of iterations for the WL algorithm at each iteration (n_{iteri}), the number of representatives at that iteration ($|R_i|$), and the number of clusters (K).

For the KMeans algorithm, the total distance computations, d_{kmeans}, are the sum of the initialization distances (d_{init}) and distances calculated during Lloyd's algorithm iterations ($dlloyd$), defined as:

$$d_{kmeans} = d_{init} + dlloyd$$

$$d_{init} = \frac{K \times (K+1)}{2} \times N$$

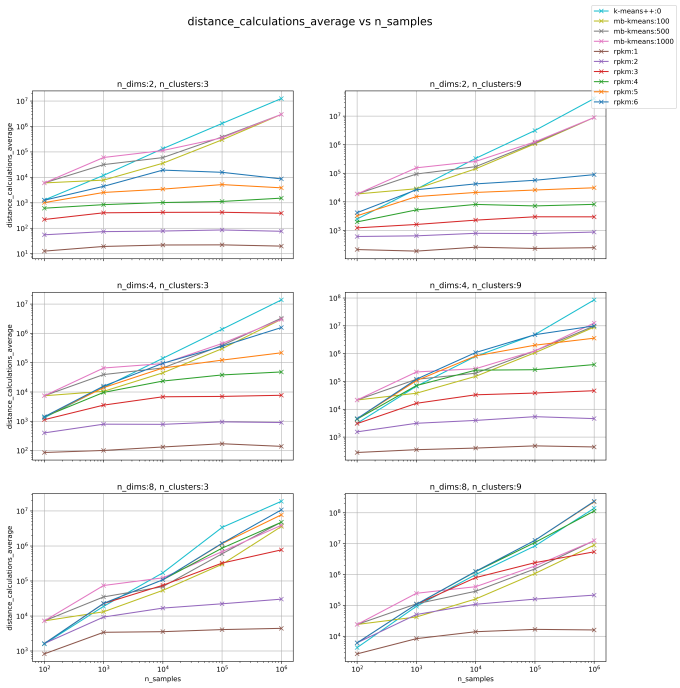
$$dlloyd = n_{iter} \times K \times N$$

Here, the initialisation computation involves calculating the distances for each new center being added during the k-means++ initialisation, which increases sequentially with each iteration up to K clusters. This results in a cumulative sum of all natural numbers up to K multiplied by the dataset size N. Lloyd's algorithm involves computing a $K \times N$ distance matrix each iteration to assign each instance to a cluster.

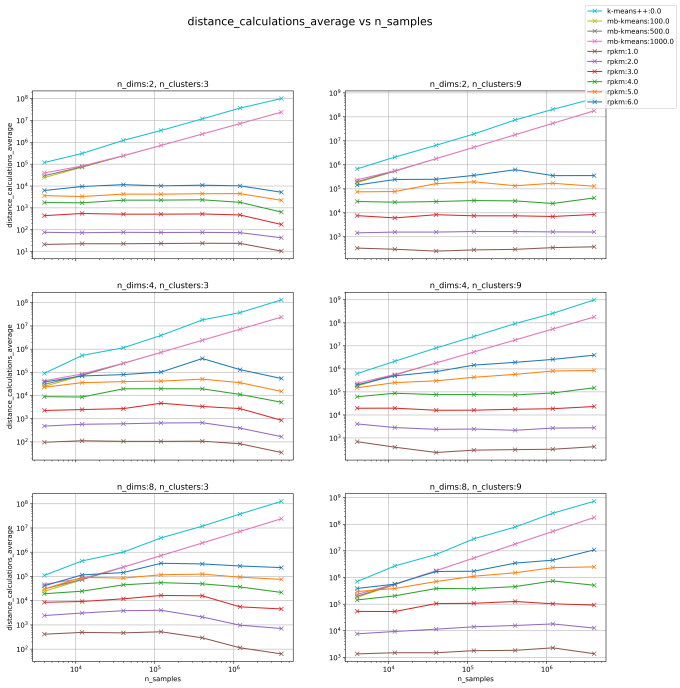
The MiniBatchKMeans algorithm follows a similar formula but differs in that $dlloyd$ is calculated with a batch size b, not the entire dataset size:

$$dlloyd = n_{iter} \times K \times b$$

This computational approach was applied across various settings, with $K \in \{3, 9\}$ and $D \in \{2, 4, 8\}$. For artificial datasets, N ranged from 100 to 1,000,000, while for the real dataset, N was set between 4,000 and 4,000,000, as specified in the study. The results, presented in the plots below, showcase the results for artificial and real datasets, respectively.



(a) Distance Calculations results in artificial datasets



(a) Distance Calculations results in real gas-sensor dataset

The analysis reveals that the number of distance computations for RPKM is generally lower than that for k-means++. In the case of the artificial dataset with high dimensionality ($D = 8$) and cluster count ($K = 9$), the results show some overlap, indicating similar performance levels. Conversely, the real dataset's results are notably promising, with RPKM's distance computations not exceeding those calculated by MiniBatchKMeans and KMeans across all tested combinations of K and D.

4.2 Quality of the approximation

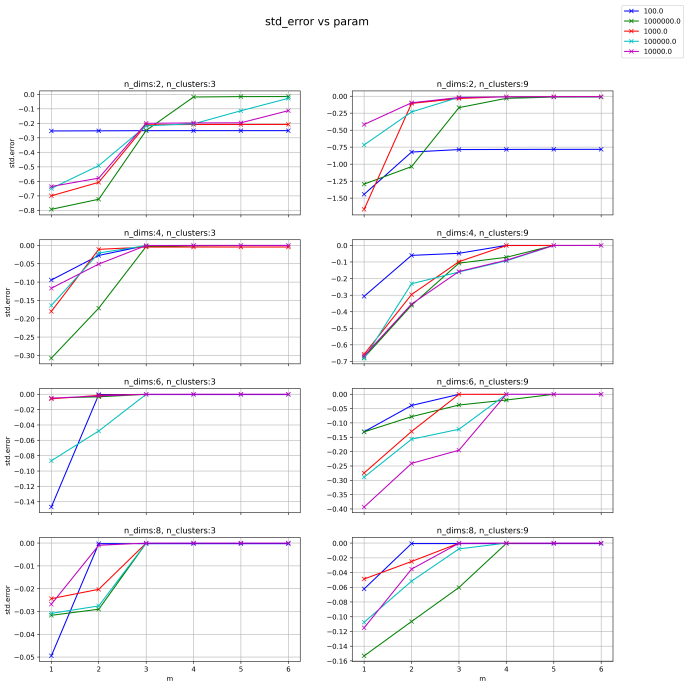
In this section, we will explore the standard error metric introduced in the study, focusing on an experimental analysis designed to evaluate the discrepancies in cluster centers identified by the RPKM algorithm versus those determined by the k-means++ algorithm. The study defines clustering error with the following formula:

$$E(C) = \sum_{x \in D} \min_{k=1,\ldots,K} \|x - c_k\|^2$$

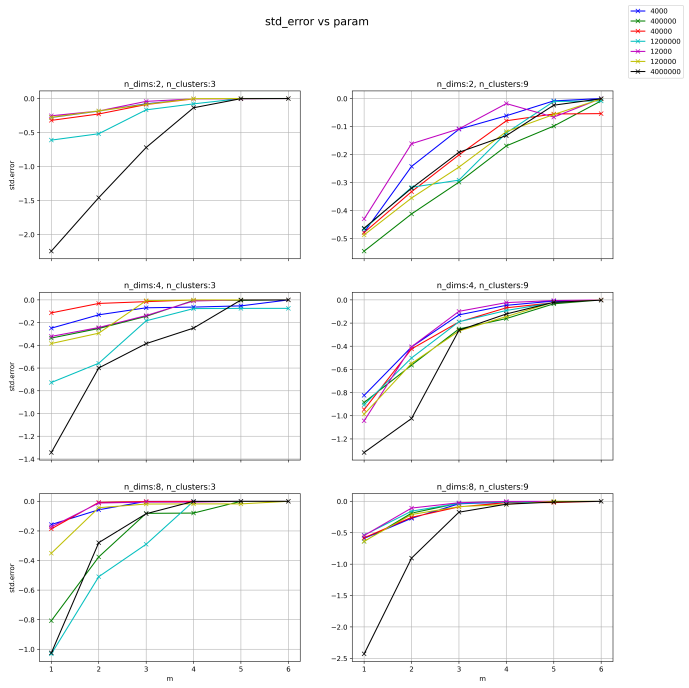
Using this formula, we compute the clustering errors for both RPKM and KMeans algorithms. The difference between these two is expressed as the ratio:

$$\rho = \frac{E_{km++} - E_{RPKM}}{E_{km++}}$$

Typically, ρ will be negative, indicating that more negative values signify a poorer performance of the RPKM in approximating cluster centers. The results of this evaluation are documented below for artificial and real datasets, respectively.



(a) Standard error results in artificial datasets



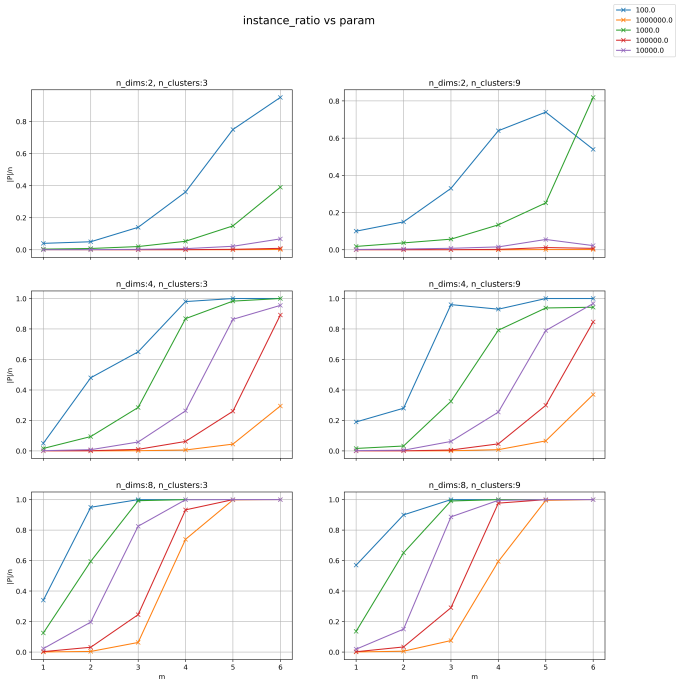
(a) Standard error results in real gas-sensor dataset

The data from these images suggest that the error reduces as the iteration number m and the number of representatives increase. Notably, as the cluster count escalates, the discrepancy in performance relative to the k-means++ algorithm becomes more evident.

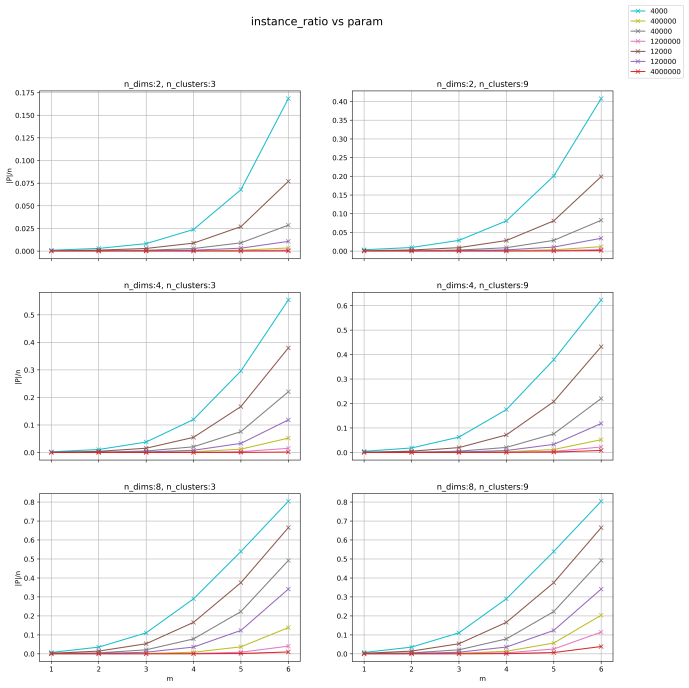
4.3 Instance Ratio

Now we evaluate the instance ratio, defined as the size of the partition relative to the total number of instances in the dataset $|P|/n$. This experiment aims to illustrate how the data points processed by the WL algorithm decrease with each iteration of the RPKM algorithm. Both artificial and real datasets were employed, similar to the previous experiment, with ten replicas and the same values for the number of instances (N), clusters (K), and dimensions (D) maintained. Only the RPKM algorithm was assessed in this context because the concept of partitions does not apply to KMeans and MiniBatchKMeans.

The instance ratio was calculated for each maximum iteration value (m) for the RPKM and recorded at the conclusion of the fitting process as the number of subsets in the mth partition divided by the total number of instances in the dataset. The results of this experiment are shown below:



(a) Instance Ratio results in artificial datasets



(a) Instance Ratio results in real gas-sensor dataset

From the graphs, it can be seen that in the initial iterations of the RPKM algorithm, the count of representatives (or subsets) is notably small relative to the dataset's sample size, which is expected. While there is clear exponential growth in the number of subsets in the real dataset, this pattern is less pronounced in the artificial dataset, where the ratio approaches 1 more swiftly compared to the real dataset. Notably, in datasets with fewer samples, the ratio increases much more rapidly than in those with a larger number of instances.

The findings are quite impressive, particularly for large datasets, as the algorithm appears to manage the data efficiently, utilising a considerably reduced number of instances. Additionally, it is observed that the ratio escalates rapidly with respect to the number of dimensions, given that the number of hypercubes grows exponentially with dimensionality, leading to a larger number of partitions.

5 Additional Experiments

In this section we are presenting additional experiments to the ones replicated from the original paper. First, we are comparing internal criteria between scikit-learn's KMeans and the RPKM algorithm. Then, we are evaluating and discussing external criteria, assessing how these models perform on specific metrics.

5.1 Internal Criteria

We evaluate three internal clustering metrics to compare the clustering outcomes of the Recursive Partition-based K-means (RPKM) algorithm against scikit-learn's KMeans. The metrics we assess include the silhouette score, Calinski-Harabasz index, and Davies-Bouldin index. The silhouette score assesses cluster separation quality, ranging from -1 to 1, where higher values indicate better-defined clusters that are distinct from each other. The Calinski-Harabasz index measures the ratio of between-cluster dispersion to within-cluster dispersion, with higher values suggesting better-clustered data. This index is particularly effective for datasets resembling spherical clusters. The Davies-Bouldin index evaluates the average similarity between each cluster and its closest cluster, with lower values indicating better separation.

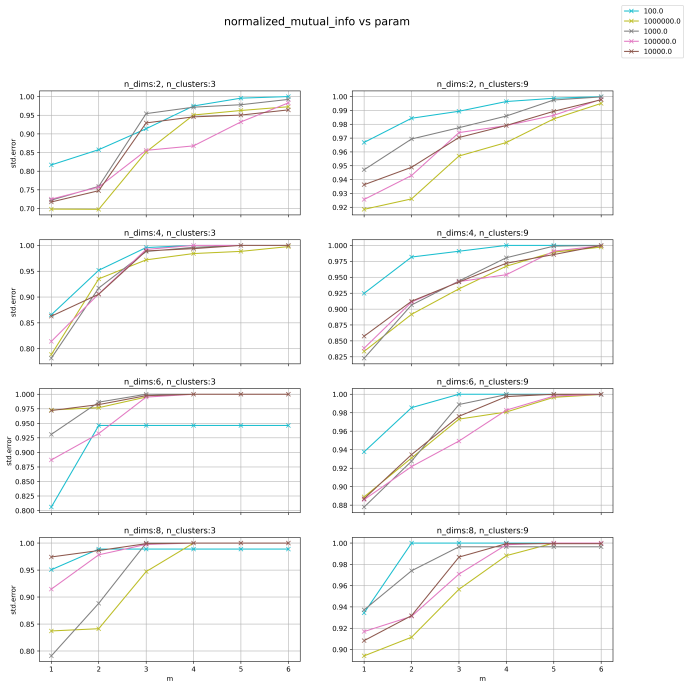
For our experiments, we utilised an artificially generated dataset comprising Gaussian mixtures. The parameters configured included the number of clusters $K \in \{3, 9\}$, dimensions $D \in \{2, 4, 8\}$, and dataset sizes $N \in \{100, 1000, 10000\}$. To ensure stability in metric measurement, each configuration was replicated 10 times. These metrics were analysed across various cluster configurations and dimensions.

The analysis focused on comparing each configuration of the RPKM algorithm against the clusters generated by KMeans++. Due to the broad range of metric values, we employed a similarity measure similar to the standard error used previously, calculating the absolute standard error as the absolute value of the difference between the RPKM and KMeans++ metrics, normalised by their sum. This scale-invariant measure ranges from 0 (identical metrics) to 1 (completely different metrics), providing a clear quantification of similarity.

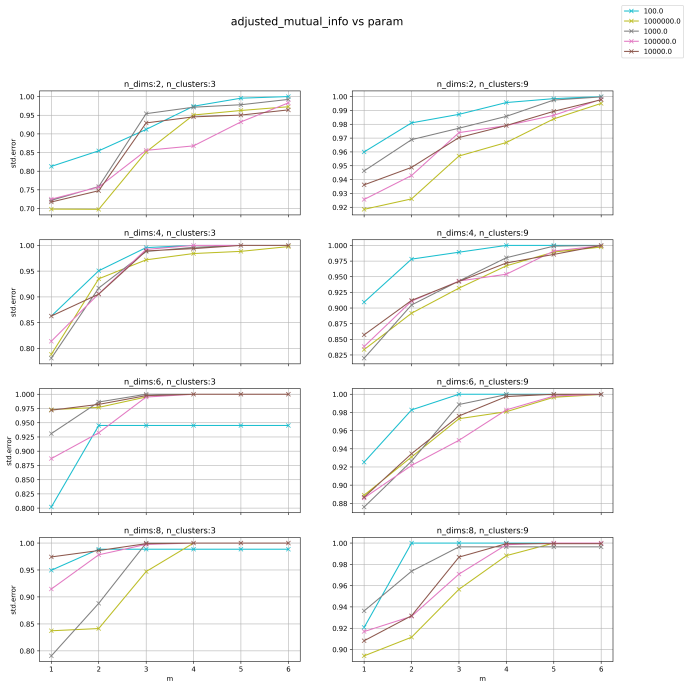
The results, presented in a detailed table, show that the RPKM algorithm's approximation to KMeans++ improves as the number of iterations increases, indicating a higher similarity in cluster outcomes. Notably, the silhouette scores generally show better similarity than the other metrics, suggesting that while the clusters are well-defined, they might not be as distinctly separated as expected, especially in datasets with overlapping clusters.

Table 1: Absolute standard error of internal criteria metrics

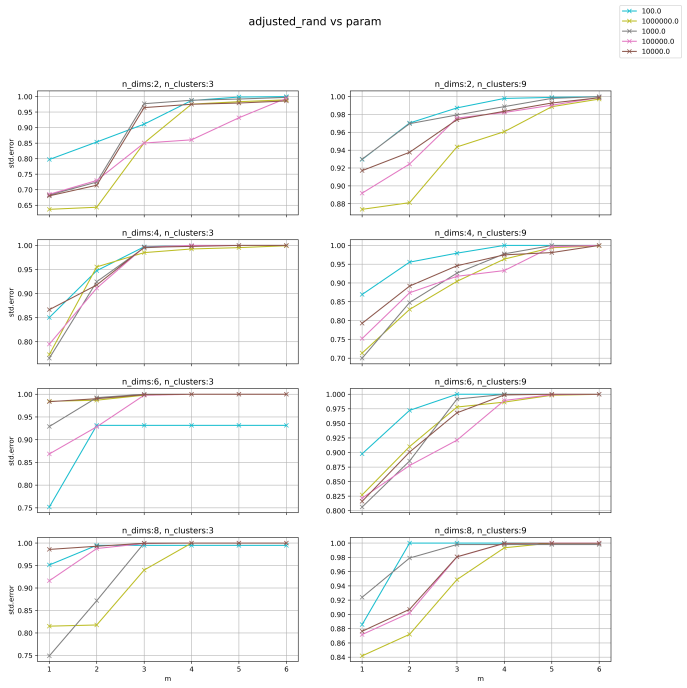
Metric	**Max Iterations**					
	1.0	**2.0**	**3.0**	**4.0**	**5.0**	**6.0**
Silhouette	0.02358	0.01655	0.00997	0.00902	0.00855	0.00837
Calinski Harabasz	0.1710	0.1423	0.1234	0.1161	0.1121	0.1112
Davies Bouldin	0.1045	0.0844	0.0687	0.0670	0.0661	0.0642



(a) Calinski Harabasz score results in artificial datasets



(a) Davies Bouldin score results in artificial datasets



(a) Silhouette score results in artificial datasets

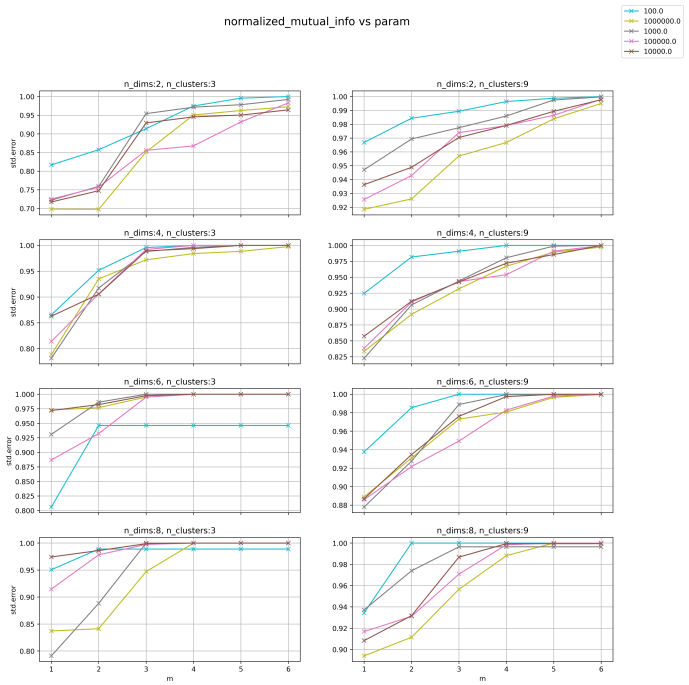
5.2 External Criteria

We assess the clustering performance of the Recursive Partition-based K-means in comparison to KMeans++ using several external metrics: Normalised Mutual Information (NMI), Adjusted Mutual Information (AMI), and Adjusted Rand Score (ARS). These metrics evaluate the similarity between two sets of clustering labels, offering insights into the relative performance of the algorithms without referring to ground truth labels. Instead, labels generated by KMeans++ are used as a reference to measure how closely RPKM's outputs resemble them.

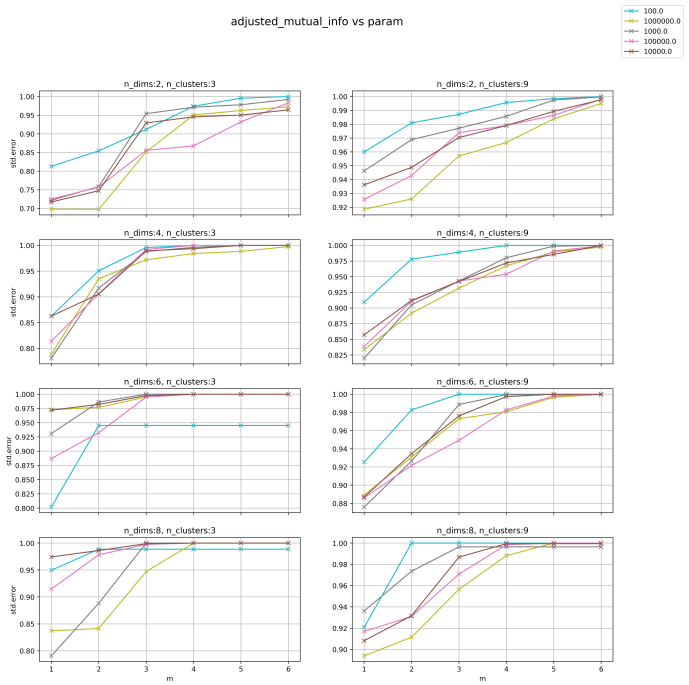
NMI normalises the Mutual Information (MI) score to scale between 0 (no mutual information) and 1 (perfect correlation), making it robust against the absolute values of labels, as label permutations do not affect the score. Similarly, AMI adjusts the MI to account for chance, providing a more accurate measure of information shared between label sets, especially when the number of clusters is large. This metric also scales from 0, indicating random labelling, to 1, denoting perfect agreement.

ARS, on the other hand, evaluates the similarity between two clustering by considering all pairs of samples and counting those that are assigned similarly across both predicted and reference label sets. The ARS value ranges from 0 for random labelling to 1 for perfectly consistent labelling.

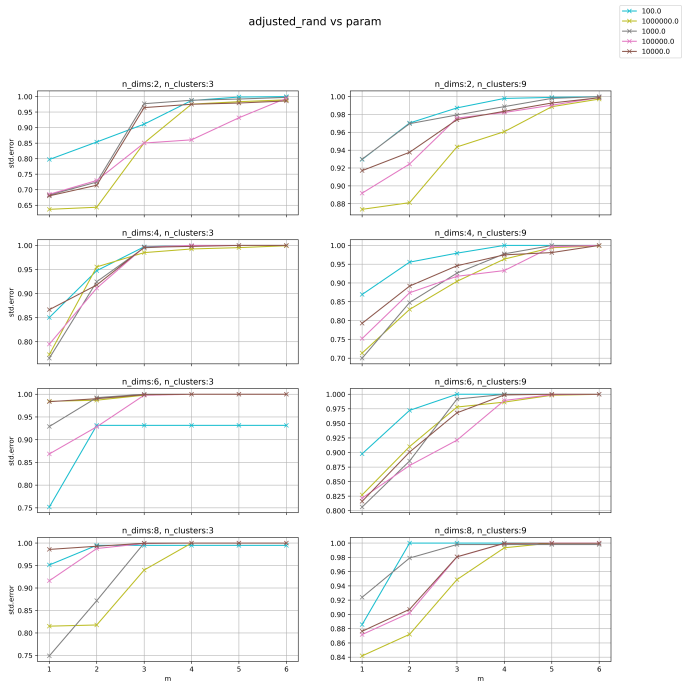
The clustering metrics were computed using an artificially generated dataset with varying parameters: number of instances $N \in \{100, 1000, 10000, 100000, 1000000\}$, number of clusters $K \in \{3, 9\}$, and number of dimensions $D \in \{2, 4, 6, 8\}$, averaged over 10 replicas to ensure statistical reliability. The results are displayed below for the NMI, AMI, and ARS metrics, respectively.



(a) Normalized Mutual Info score results in artificial datasets



(a) Adjusted Mutual Info score computation results in artificial datasets



(a) Adjusted Random score results in artificial datasets

Initial observations from the plots indicate that NMI and AMI scores are very close, reflecting the non-random, heuristic-based nature of the cluster assignments by both algorithms. Given their similarity and the moderate cluster size used in the experiments, AMI and NMI provide equivalent insights. Furthermore, the ARS scores, which are conceptually akin to the other two metrics, corroborate the findings that RPKM effectively approximates the KMeans++ clustering results, especially as the number of algorithm iterations increases. These findings reinforce the claims of the paper that RPKM is a viable alternative to KMeans++ for clustering large datasets.

6 Conclusions

In this study, we implemented the PRKM algorithm, originally introduced by Marco Capó, Aritz Pérez, and Jose A. Lozano, detailed in their research publication "An efficient approximation to the K-means clustering for massive data." Through a series of experiments, we have shown that our implementation performs on par with that of the authors. Additionally, we achieved results that closely match those documented in their study, thus validating their findings and conclusions.

The RPKM algorithm serves as an effective alternative to scikit-learn's KMeans and MiniBatchK-Means, especially in scenarios involving large datasets and high-dimensional features. It significantly reduces the number of distance computations while still maintaining a reliable approximation of cluster error. Based on the outcomes of our experiments, we are confident in the algorithm's capability to efficiently handle large-scale clustering tasks, a claim supported by the empirical evidence provided.

References

[1] Marco Capó, Aritz Pérez, and Jose A. Lozano. "An efficient approximation to the K-means clustering for massive data". In: *Knowledge-Based Systems* 117 (2017). Volume, Variety and Velocity in Data Science, pp. 56–69. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2016.06.031. URL: https://www.sciencedirect.com/science/article/pii/S0950705116302027.