CS403/603 - ML Project Proposal

Faster *k*-Medoids Clustering: Improving the PAM

Submitted

In partial fulfillment of the requirements of

CS 403 Machine Learning

by

Arjun Singh (200001007)
Ayush Sinha (200001012)
Prashant Kumar (200001062)
Shubham Pednekar (200001073)

Submitted to

Prof. Dr. Kapil Ahuja

Department Of Computer Science and Engineering

Indian Institute of Technology Indore

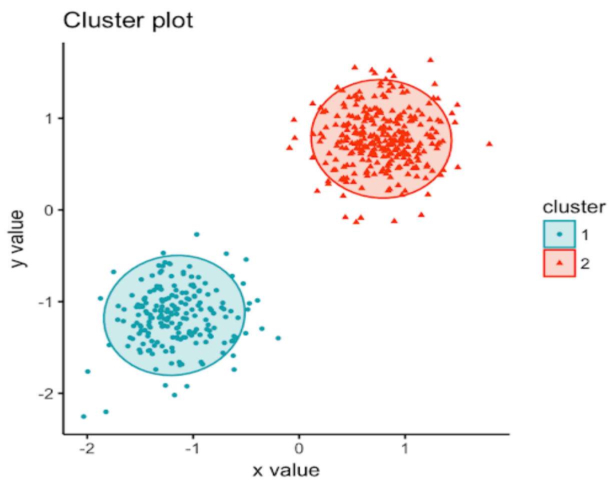
A.Y 2022-2023

Introduction:

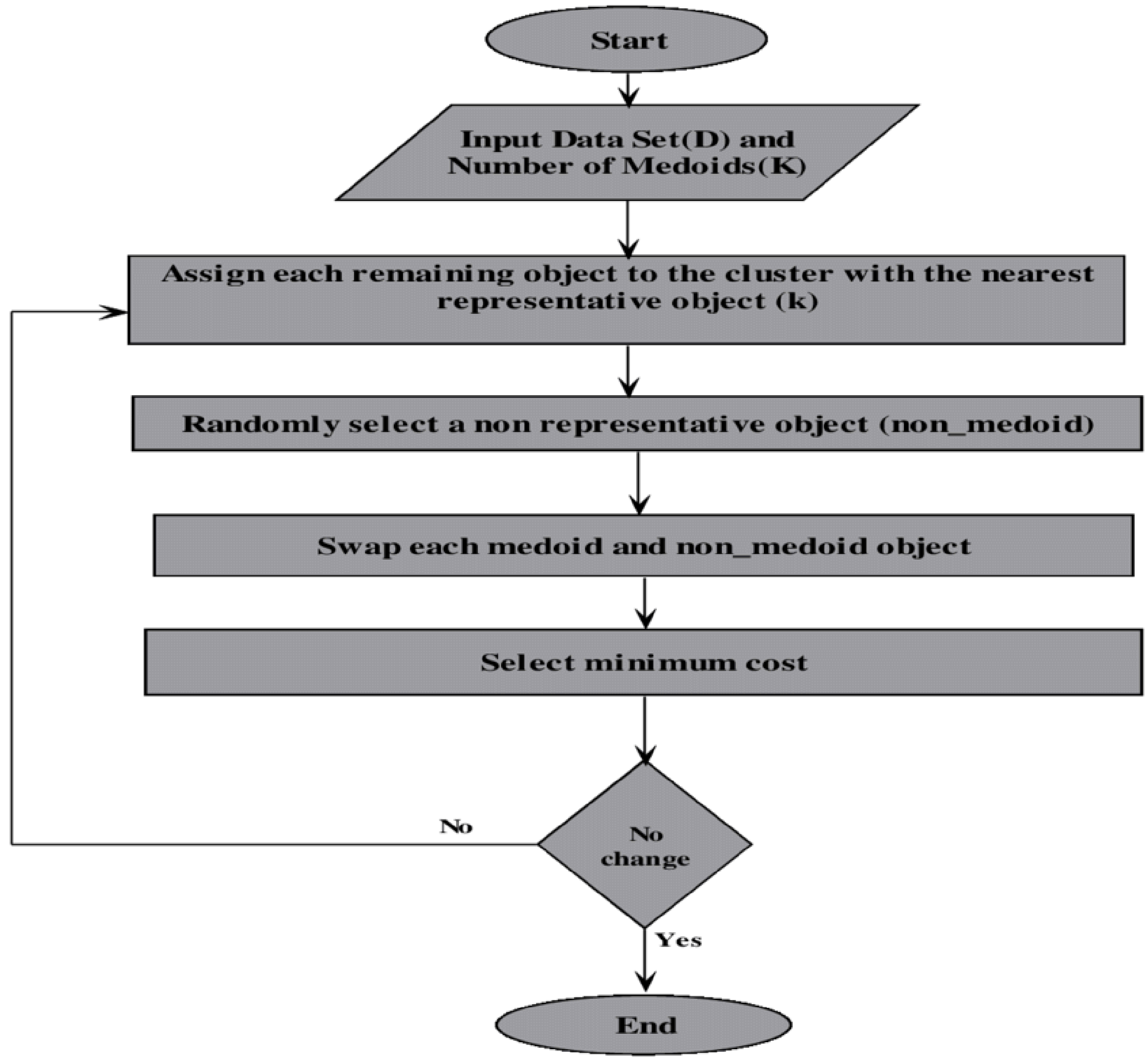
Clustering is a fundamental task in machine learning, aiming to partition datasets into cohesive groups based on similarity automatically. Among the plethora of clustering algorithms, K-means, we group data points by calculating the average position of each cluster, known as the mean, and assigning points to the nearest mean. This process iterates until the clusters stabilize, aiming to minimize the sum of squared distances between points and their cluster means.

On the other hand, in k-medoids clustering, instead of using means, we choose representative points from the dataset called medoids. These medoids act as prototypes for the clusters and are selected to minimize the total deviation of points to their respective medoids. Unlike k-means, where any point can be a mean, in k-medoids, medoids must be actual data points. This method is more flexible, allowing for different dissimilarity measures and input domains.

The Partitioning Around Medoids (PAM) algorithm is commonly used for this purpose, aiming to find a good partitioning of the data using medoids. However, a vital issue with PAM is its high run time cost, particularly in its second phase i.e. swapping. This project proposal seeks to address this issue by proposing modifications to the PAM algorithm to enhance its efficiency without compromising on result quality.



PAM Algorithm Workflow


Start
Input Data Set(D) and Number of Medoids(K)
Assign each remaining object to the cluster with the nearest representative object (k)
Randomly select a non representative object (non_medoid)
Swap each medoid and non_medoid object
Select minimum cost
No change
No
Yes
End

Problem Statement

The Partitioning Around Medoids (PAM) algorithm, a widely-used method for clustering non-Euclidean data, faces a significant challenge due to its high computational complexity, particularly in its second phase involving medoid swapping. This computational overhead limits the algorithm's scalability and impedes its practical applicability to large-scale datasets. The critical issue is reducing the PAM algorithm's runtime complexity without compromising the quality of clustering results by many factors, ensuring that it remains adequate for real-world data analysis tasks. This project aims to address this problem by proposing modifications to the PAM algorithm that enhance its run time while not changing its accuracy by much. We propose modifications to the PAM algorithm that achieve an $O(k)$-fold speedup in the second ("SWAP") phase of the algorithm, but will still find the same results as the original PAM algorithm.

Objective

- To implement the PAM algorithm.
- To apply medoid clustering using euclidean distance as the parameter of loss function.
- To optimize the runtime performance of the PAM algorithm by finding the Best Swap.
- To make the PAM swap faster by a speedup on the order of $O(k)$.

Formulation and WorkFlow

Medoid clustering

In medoids clustering, also known as k-medoids, the objective is to partition a dataset into k clusters by selecting representative objects called medoids. The Total Deviation criterion calculates the total dissimilarity of each data point in its respective cluster to the medoid of that cluster. This measure is used as an objective function to guide the clustering process, aiming to minimize the total dissimilarity across all clusters.

$$TD := \sum_{i=1}^{k} \sum_{x_j \in C_i} d(x_j, m_i)$$

where:

- k is number of clusters
- C_i is ith cluster
- x_i is datapoint in cluster Ci
- m_i is medoid in cluster Ci
- $d(x_j,m_i)$ is dissimilarity between datapoint and medoid.

The medoid of a cluster C is defined as the object with the smallest sum of dissimilarities to all other objects in the set:

$$\text{medoid}(C) := \arg\min_{x_i \in C} \sum_{x_j \in C} d(x_i, x_j)$$

Finding The Best Swap

The algorithm SWAP evaluates every swap of each medoid $\boldsymbol{m_i}$ with any non-medoid $\boldsymbol{x_j}$ recomputing the resulting ***TD*** every time requires finding the nearest medoid for every point, which causes many redundant computations. Instead, PAM only computes the change in ***TD*** for each object $\boldsymbol{x_o}$ if we swap $\boldsymbol{m_i}$ with $\boldsymbol{x_j}$:

$$\Delta TD = \sum_{x_o} \Delta(x_o, m_i, x_j)$$

In the function ***Δ(x$_o$, mi$_i$, x$_j$)*** we can often detect when a point remains assigned to its current medoid (if c_k != c_i) and this distance is also smaller than the distance to x_j) , and then immediately return 0. Because of space restrictions, we do not repeat the original "if" statements used in [1], but instead condense them into the equation:

$$\Delta(x_o, m_i, x_j) = \begin{cases} \min\{d(x_o, x_j), d_s(o)\} - d_n(o) & \text{if } i = nearest(o) \\ \min\{d(x_o, x_j) - d_n(o), 0\} & \text{otherwise} \end{cases}$$

where ***d$_n$(o)*** is the distance to the nearest medoid of **o**, and ***d$_s$(o)*** is the distance to the second nearest medoid. Computing them on the fly increases the runtime by a factor of $O(k)$, but we can cache these values, and only update them when performing a swap.

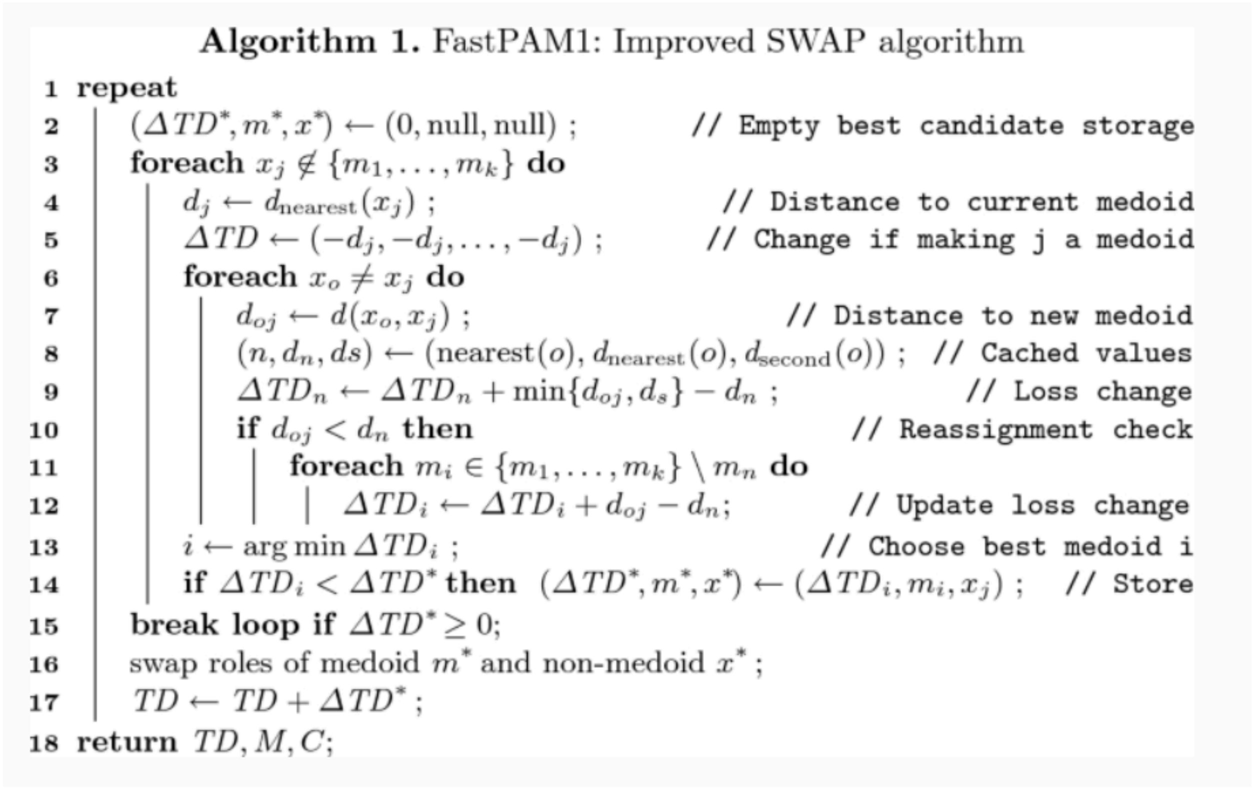
Reynolds et al. [2] note that we can decompose ***ΔTD*** into: (i) the loss of removing medoid ***m$_i$***, and assigning all of its members to the next best alternative, which can be computed as

$$\sum_{o \in C_i} d_s(o) - d_n(o)$$

(ii) the (negative) loss of adding the replacement medoid ***x$_j$***, and reassigning all objects closest to this new medoid. Since (i) does not depend on the choice of ***x$_j$*** , we can make the loop over all medoids ***m$_i$*** outermost, reassign all its points to the second nearest medoid (cache the distance to the now nearest neighbor), and compute the resulting loss. We then iterate over all non-medoids and compute the benefit of using them as the missing medoid instead. In the ***Δ*** function, we no longer have to consider the second nearest now (we virtually removed the old medoid already). The authors observed roughly a two-fold speedup using this approach.
Our approach is based on a similar idea of exploiting redundancy in these computations (by caching shared computations), but we instead move the loops over the medoids ***m$_i$*** into the innermost for loop. The reason for this is to further remove redundant computations. This becomes apparent when we realize that in Eq. 4, the second case does not depend on the current medoid i. If we transform the second case into an if statement, we can often avoid iterating over all k medoids.

Making The PAM SWAP faster : FASTPAM1

Pseudocode:



Algorithm 1. FastPAM1: Improved SWAP algorithm

```
1  repeat
2      $(\Delta TD^*, m^*, x^*) \leftarrow (0, \text{null}, \text{null})$ ;        // Empty best candidate storage
3      foreach $x_j \notin \{m_1, \dots, m_k\}$ do
4          $d_j \leftarrow d_{\text{nearest}}(x_j)$ ;                              // Distance to current medoid
5          $\Delta TD \leftarrow (-d_j, -d_j, \dots, -d_j)$ ;                      // Change if making j a medoid
6          foreach $x_o \neq x_j$ do
7              $d_{oj} \leftarrow d(x_o, x_j)$ ;                                   // Distance to new medoid
8              $(n, d_n, ds) \leftarrow (\text{nearest}(o), d_{\text{nearest}}(o), d_{\text{second}}(o))$ ;  // Cached values
9              $\Delta TD_n \leftarrow \Delta TD_n + \min\{d_{oj}, d_s\} - d_n$ ;      // Loss change
10             if $d_{oj} < d_n$ then                                              // Reassignment check
11                 foreach $m_i \in \{m_1, \dots, m_k\} \setminus m_n$ do
12                     $\Delta TD_i \leftarrow \Delta TD_i + d_{oj} - d_n$;            // Update loss change
13         $i \leftarrow \arg\min \Delta TD_i$ ;                                    // Choose best medoid i
14         if $\Delta TD_i < \Delta TD^*$ then $(\Delta TD^*, m^*, x^*) \leftarrow (\Delta TD_i, m_i, x_j)$ ;   // Store
15     break loop if $\Delta TD^* \geq 0$;
16     swap roles of medoid $m^*$ and non-medoid $x^*$ ;
17     $TD \leftarrow TD + \Delta TD^*$ ;
18 return $TD, M, C$;
```

Algorithm 1 shows the improved SWAP algorithm. In lines 4–5 we compute the benefit of making ***x**$_j$* a medoid. As we do *not yet* decide which medoid to remove, we use an array of ***ΔTD***s for each possible medoid to replace. We can now for each point compute the benefit when removing its current medoid (line 9), or the benefit if the new medoid is closer than the current medoid (line 10), which corresponds to the two cases in Eq.1 . Because the second case does not depend on *i*, we can replace the min statement with an if conditional *outside* of the loop in lines 10–12. After iterating over all points, we choose the best medoid, and remember the overall best swap. If we always prefer the smaller index *i* on ties, we choose *exactly the same* swap as the original PAM algorithm. Assuming that the new medoid is closest in *O*(1/*k*) cases on average, we can compute the change for all *k* medoids with ***O(K · 1/K)=O(1)*** effort, by saving on the innermost loop. Therefore, we expect a typical speedup on the order of *O*(*k*) compared to the original PAM SWAP (but it may be hard to guarantee this for any *useful* assumption on the data distribution; the worst case supposedly remains unaffected) at the slight cost of storing one ***ΔTD*** for each medoid m_i (compared to the cost of the distance matrix and the distances to the nearest and second nearest medoids, the cost of this is negligible).

Reference

Schubert, E., Rousseeuw, P.J. (2019). Faster *k*-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. In: Amato, G., Gennaro, C., Oria, V., Radovanović , M. (eds) Similarity Search and Applications. SISAP 2019. Lecture Notes in Computer Science(), vol 11807. Springer, Cham. https://doi.org/10.1007/978-3-030-32047-8_16

-by
Ayush Sinha (200001012)
Arjun Singh (200001007)
Prashant Kumar (200001063)
Shubham Pednekar(200001073)