# University of Pretoria

## Computer Science

COS 710 - Assignment 2 Technical Report

Keoagile Dinake - u15041744

Implementation Repository

COMPUTER SCIENCE


UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

# Introduction

For this assignment, it was required that we use a Multi Objective PSO (MOPSO) in the application of clustering static data. Given that any clustering technique can be utilized, the application of MOPSO in the clustering process was left open for the student to decide. This brought up several concerns, some of which are as follows: which clustering approach to use, what is a suitable representation for a particle given the clustering approach, and how to measure the performance of the combined concepts. In this report I will provide background on the problem domain, the algorithms and approaches used in developing my solution. Thereafter, I will discuss my approach to implementing my solution, provide and explain the research results that I obtained, and finally layout my conclusion.

# Background

## Data Clustering

Data clustering is a common practice that is often used in Exploratory Data Analysis (EDA) and Data Mining (DM) for the benefit of gaining insight on your data. In a society where there is a vast amount of data to draw insight from, it is quite necessary for the knowledge extraction process to happen efficiently. Due to this many data scientists use Machine Learning (ML) algorithms, specifically unsupervised ML algorithms, because often times the data they wish to cluster does not have predefined labels.

Granted that the focus of this assignment is on the application of MOPSO, rather than data clustering, I have chosen the KMeans algorithm which is a relatively simple yet effective unsupervised ML algorithm that will, for the sake of this assignment, demonstrate how MOPSO can be applied to traditional static data clustering. The reader should be aware that other unsupervised ML algorithms such as Linear Vecor Quantization, Self Organizing Maps, etc. are commonly used could perhaps provide a more efficient solution than the one proposed in this report. However, that investigation is ommitted in this report and left for the reader to persue.

## KMeans Clustering Algorithm

The standard KMeans clustering algorithm operates by doing the following:

```
let S be the dataset, where each data sample is of I-dimensions
Randomly initialize K cluster centroids c[k] each of I-dimensions, where 1 <= k <= K
Repeat
        let s be a data sample not yet presented
        let c be the centroid closest to s
        for every cluster c[k]
                move that cluster to the average(mean) of points assigned to that cluster
Until stopping_condition
```

Note that for my implementation, I added slight variations to the standard KMeans algorithm to remove any possible biases in the order of presentation or in the initial value for the centroids. For this, I simply randomly shuffled the data samples after every epoch and initialized each centroid to uniformly distributed values within the ranges of each attribute of the data set.

## Multi Objective Particle Swarm Optimization

MOPSO is a class of Computation Intelligence, namely Swarm Intelligence, and often describes optimization problems with more than one, but less than four objectives to optimize.
For this assignment there were two objectives:

- minimize the intra-cluster distances, and
- maximize the inter-cluster distances.

In my approach, I viewed the task at hand as a traditional clustering task that required finding a set of cluster centroids, such that the intra-cluster distances are minimized and the inter-cluster distances are maximize. As a result my approach involves using two sub-swarms, each swarm aiming to solve the objective function that finds the centroids who's summated intra-cluster distance is minimal and who's summated inter-cluster distance is maximal over all. However, with this approach, rises a need for communication between particles of each swarm such that the global best solutions of each swarm can be utilized in satisfying the given objectives. To achieve this I chose to represent a particle in the following way:

- A particle has a set of centroids, i.e. the set of centroids that satisfy the objectives.
- A particle's position is a vector that contains the intra-cluster distances and the inter-cluster distances for the centroids belonging to that particle.
- A particle's personal best and velocity vectors are of the same dimensions as the particle's position vector.

The swarms of the MOPSO algorithm are also then represented in the following way: Each swarm contains a set of particles, a global best position vector and a global best centroids set, which will be the set of centroids that make up the global best position vector's elements.

For the case of dynamically determining the optimal number of clusters, I applied the above approach through several cluster sizes, and with each, measuring the silhouette score and finally determining the optimal number of clusters by taking the difference between adjacent cluster's silhouette scores and selecting the one with the highest difference, i.e. selecting the cluster with the steepest slope.

# Implementation

## Dataset Descriptions

### Iris Dataset

- 150 Data samples
- 4 Attributes (continuous-valued)
- 3 Classes
- 50 data samples per class
- Iris - UCI ML Repository

### Wine Dataset

- 179 Data samples
- 13 Attributes (continuous-valued)
- 3 Classes
- <59, 71, 48> data samples per class relatively
- Wine - UCI ML Repository

### Epileptic Seizure Recognition Dataset

- 11500 Data samples
- 178 Attributes (continuous-valued)
- 5 Classes
- 2300 data samples per class
- Epileptic Seizure Recognition - UCI ML Repository

## Algorithm Parameters and Run Conditions

### Priori Number of Clusters

- 50 Independent Runs
- 100 Epochs per independent run
- 2 Particle Swarms
- 30 Particles per swarm
- w = 0.729844
- c1 = c2 = 1.49618
- r1 ~ U(0,1)
- r1 ~ U(0,1)

Note that r1 and r2 have the same dimensions as a particle's position.

## Dynamic Number of Clusters

- Possible number of clusters ranges from 2 to 15
- 50 Independent Runs per possible number of clusters
- 100 Epochs per independent run
- 2 Particle Swarms
- 30 Particles per swarm
- w = 0.729844
- c1 = c2 = 1.49618
- r1 ~ U(0,1)
- r1 ~ U(0,1)

Note that r1 and r2 have the same dimensions as a particle's position. The stopping condition for each independent are when the max number of epochs are reached, i.e. 100 epochs.

## Algorithm Initialization

### Data Preprocessing

The dataset is read from a specified file. From there, the data is processed in such as way as to convert strings into numerical values.

### Swarms and Particles

To initialize the two swarms and the particles they contain the following was done:

Repeat the steps below for the number of specified particles, which in this case is 30.

- Get the bounds of each feature/attribute of the dataset.
- Create two particles (say p1 and p2), one for each swarm.
- For every cluster
    - generate two randomly distributed centroids within the bounds of each feature/attribute, and add one centroid to p1's centroids list, and the other to p2's centroids list.
- Set p1's velocity ~ U(0,1) and p2's velocity ~ U(0,1)
- Set p1's position to very large and very small real values, such that p1.position[0] = Double.MAX_VALUE and p1.position[1] = Double.MIN_VALUE
- Set p2's position to very large and very small real values, such that p2.position[0] = Double.MAX_VALUE and p2.position[1] = Double.MIN_VALUE
- Set the personal best of both p1 and p2 to their position vectors.
- Add p1 to Swarm 1 and add p2 to Swarm 2
- Set Swarm1's global best position to very large and very small real values, such that Swarm1.globalBestPosition[0] = Double.MAX_VALUE and Swarm1.globalBestPosition[1] = Double.MIN_VALUE
- Set Swarm2's position to very large and very small real values, such that Swarm2.globalBestPosition[0] = Double.MAX_VALUE and Swarm2.globalBestPosition[1] = Double.MIN_VALUE

## MOPSO applied to KMeans

In the below defined approach, the aim was to improve the training of the KMeans algorithms such that the optimal set of clusters that satisfied the objective function mentioned in the Background section. The implementation is as follows:

## # Optimization Step

- Repeat until max epochs reached
    - ## # KMeans Clustering Step
    - Repeat for each swarm s1 and s2
        - Repeat for each data sample ds not yet presented
            - find a particle from swarm s1 and s2 respectively, who's centroid is closest to ds (using euclidean distance measure)
        - For every particle p in swarm s1 and s2 respectively
            - For every centroid in particle p
                - move that centroid to the average(mean) of the data samples closest to that centroid
                - calculate the intra-cluster distance of that centroid (using euclidean distance measure)
            - calculate the inter-cluster distance of all the centroids in particle p (using euclidean distance measure)
    - ## # MOPSO Step
    - Repeat until max number of particles per swarm (i.e. 30)
        - update the personal best for each particle in swarm 1 -> minimize
        - update the personal best for each particle in swarm 2 -> maximize
        - update the global best position for swarm 1 -> minimize
        - update the global best set of centroids for swarm 1
        - update the global best position for swarm 2 -> maximize
        - update the hlobal best set of centroids for swarm 2
    - Repeat until max number of particles per swarm (i.e. 30)
        - update the velocity of every particle p in swarm 1 such that p.velocity[t+1] = w * p.velocity[t] + cognitiveComponent + socialComponent where,
            - cognitiveComponent = c1 * r1 * (swarm1.globalBestPosition[t] - p.position[t]) and
            - socialComponent = c2 * r2 * (swarm2.globalBestPosition[t] - p.position[t]) -> exchange global best information between swarms
        - update the velocity of every particle p in swarm 2 such that p.velocity[t+1] = w * p.velocity[t] + cognitiveComponent + socialComponent where,
            - cognitiveComponent = c1 * r1 * (swarm2.globalBestPosition[t] - p.position[t]) and

            - socialComponent = c2 * r2 * (swarm1.globalBestPosition[t] - p.position[t]) -> exchange global best information between swarms
        - For every particle p1 and p2 in swarm s1 and s2 respectively
            - update the position of p1 such that p1.position[t + 1] = p1.position[t] + p1.velocity[t + 1]
            - update the position of p2 such that p2.position[t + 1] = p2.position[t] + p2.velocity[t + 1]
    - re-initialize the values for r1 and r1
    - randomly shuffle the order of the data samples in the dataset

After running the Optimization Step 50 independent times the global best of either swarm is compared and the one with the smallest intra-cluster distance and the largest inter-cluster distance is chosen as the optimal result. The corresponding optimal set of centroids is the global best set of centroids of the winning swarm. From here, the average number of samples belonging to each centroid in the optimal set of centroids is computed and the execution ends. However, if a dynamic optimal number of clusters is to be determined then, after the average number of samples belonging to each centroid in the optimal set of centroids is computed the silhouette score of the optimal set of centroids is calculated and stored along with the optimal set of centroids for that particle cluster size for later usage.

Once all ranges of possible cluster sizes have been executed over 50 independent runs each, the difference between cluster size (i)'s silhouette score and cluster size (i + 1)'s silhouette score is then computed. The result optimal cluster size and it's corresponding set of optimal centroids is then the cluster size (i)  who's silhouette difference is maximal, i.e. the cluster size (i) who's silhouette score was subtracted by to obtain the steepest slope in silhouette differences. For futher elaboration consider the following:

cluster size (i) has silhouette score 3, cluster size (i + 1) has silhouette score 1 and cluster size (i + 2) has silhouette score 0.7
Then the silhouette difference scores are as follows:
silhouette_difference[0] = 3 - 1 = 2
silhouette_difference[1] = 1 - 0.7 = 0.3

Thus when comparing these silhouette difference scores the maximal value is 2, which is also the steepest difference in silhouette scores and belongs to the cluster(i + 1) who's value was subtracted by to obtain the steepest and/or highest silhouette difference.

The results of this above implementation to the 3 datasets are provided and discussed in the Research Results section

# Research Results

For each of the following dataset results, my solution was run over 50 independent runs for priori number of clusters and 50 independent runs for each possible optimal number of clusters in the range from 2 to 15. Therefore, the results showed below, are averages of those 50 independent runs.

## Iris Dataset Results

### Priori Number of Clusters

- Number of clusters: 3
- Average number of samples per cluster: {50, 52, 48}

Given that the actual number of samples per cluster is {50, 50, 50}, my implementation has an error rate of 2.667% (i.e. 4/150 samples were incorrectly clustered).

### Dynamic Optimal Number of Clusters

- Determined optimal number of clusters: 3
- Average number of samples per cluster: {53, 50, 47}

Given that the actual number of cluster is 3, my implementation was able to determine the correct optimal number of clusters each time using the approach mentioned in the Implementation section. For that optimal number of clusters, the average number of samples per cluster over those 50 independent runs was {54, 50, 47}. Given that the actual number of samples per cluster is {50, 50, 50}, my implementation has an error rate of 4% (i.e. 6/150 samples were incorrectly clustered).

## Wine Dataset Results

### Priori Number of Clusters

- Number of clusters: 3
- Average number of samples per cluster: {73, 71, 35}

Given that the actual number of samples per cluster is {59, 71, 48}, my implementation has an error rate of 15.084% (i.e. 27/179 samples were incorrectly clustered).

### Dynamic Optimal Number of Clusters

- Determined optimal number of clusters: 4
- Average number of samples per cluster: {46, 44, 46, 40}

Given that the actual number of cluster is 3, my implementation was off by 1 cluster each time using the approach mentioned in the Implementation section. This is due to the fact that the dataset was not normalized and thus skewed the results in favour of the attribute with the highest value. To try and mitigate this effect, one could preprocess the dataset such that all attributes are on the same scale, i.e. from 0 to 1. Also, one could apply Principal Component

Analysis (PCA) reduction on the dataset such that only "relevant" features of the dataset are utilized. For that optimal number of clusters, the average number of samples per cluster over those 50 independent runs was {46, 44, 46, 40}. Given that the actual number of samples per cluster is {59, 71, 48}, my implementation has an error rate of 45.810% (i.e. 83/179 samples were incorrectly clustered).

### Epileptic Seizure Recognition Dataset Results

#### Priori Number of Clusters

- Number of clusters: 5
- Average number of samples per cluster: {3267, 2202, 3204, 2668, 159}

Given that the actual number of samples per cluster is {2300, 2300, 2300, 2300, 2300}, my implementation has an error rate of 38.939% (i.e. 4478/11500 samples were incorrectly clustered).

#### Dynamic Optimal Number of Clusters

- Determined optimal number of clusters: 12
- Average number of samples per cluster: {1298, 800, 977, 982, 666, 395, 1198, 934, 736, 1190, 1253, 1066}

Given that the actual number of samples per cluster is {2300, 2300, 2300, 2300, 2300}, my implementation has an error rate of 47.783% (i.e. 5495/11500 samples were incorrectly clustered).

# Conclusion

## Solution Feasibility

The focus of this assignment was to determine whether a MOPSO could be used to solve a static data clustering problem, and In this report I have shown a method of how it could be achieved.

## Performance Observations

The research results in the section above show that on different classes of data with varying dimensions, the algorithm was able to produce some fairly accurate results. On low dimensions the results were near 100% accuracy each attempt at executing the solution, as in the case of the Iris dataset results. However, on the Wine dataset where the number of data samples were roughly the same, the difference factor came in on the dimensions of the data. Wine has 13 dimensions per data sample whereas Iris has only 4, and I believe that the difference in the performance on the results is largely due to the varying scale of the features and the chance that we're considering "non-relevant" features. So PCA reduction could be utilized, among other dimension reduction algorithms. This issue of dimensionality can also be seen with the

performance of the solution on the Epileptic Seizure Recognition dataset, which had a dimensionality of 6 and over 11000 data samples. It is worth noting that on 2/3 datasets, the optimal number of clusters was mis-classified, and thus the performance of the data clustering was slightly better than just randomly assignment of a data sample to a cluster. My understanding leads me to believe that Particle Swarm Optimization may not perform well with high dimensional data. Note that the above observations were consistent whether dynamic number of clusters were to be determined or whether they were known prior to execution.

## Final Words

In my humble opinion, this is probably not the best use case for MOPSO, considering that the training takes ages to complete. Also, it is worth mentioning that my approach has several flaws as pointed above and could possible be mitigated through choosing a different method of clustering, add data preprocessing techniques, and use several PSO performance enhancement methods to ensure better results. With that being said, I enjoyed getting to work on such an assignment and look foward to learning more about Swarm Intelligence.