School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

COMS4040A & COMS7045A Project – Report
K-Means and Fuzzy C-Means Clustering
Parallel Machine Learnig Algorithms

Shameel Nkosi, 1814731, Coms Hons
Siraj Motaung,1390537, BDA Hons

July 3, 2021

Contents

1 Introduction

In machine learning, clustering is a technique of grouping objects of similar characteristics into the same groups called clusters. The similarity between any two objects is measured by their distance from each other.

In this project, we have implemented the K-Means clustering algorithm and the Fuzzy C-Means algorithm. Traditionally, we aim to classify objects into separate clusters. The K-Means algorithm allows us to achieve this type of clustering, where an object belongs to only one cluster. On the other hand, Fuzzy C-Means allows us to classify an object as a member of different clusters. The K and C in K-Means and Fuzzy C-Means respectively are hyper-parameters that specify the number of clusters or groups we wish to have. In this project we are going to implement these algoriths using three different implementations, namely, serial implementation in C++, parallel implementation in Cuda and in MPI.

1.1 Problem Statement

In this project, we have taken a dataset from the UCI Repository. This data set describes the chemical composition of the wine. We want to use these chemical compositions to find the origins of the wine. These wines are from 3 different cultivars in the same region in Italy. We, therefore, aim to find out which of these wines belong to which cultivar.

2 Methodology

2.1 Algorithms

This subsection describes the algorithms we implemented in-depth.

2.1.1 K-Means Clustering

As mentioned above, K-Means partitions all objects into K clusters or subsets of the data set. Each object is an observation in our data set, which belongs to the nearest mean among the K means. Initially, we need to initialize the K mean as points in the same space as all the data points. The easiest way is to choose k random points as the initial means. Mathematically we can describe the problem as follows: Given a set of observations in a d-dimensional space, K-Means partitions these observations into $k \leq n$ sets $S = \{S_1, S_2, .., S_k\}$. We then want to find the minimum distance between each observation and the means. Formally, we to find:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S} \|\mathbf{x} - \mu_i\|^2 \tag{1}$$

K-Means is an iterative algorithm. We iteratively update the means until there is stability i.e. the means aren't changing anymore. In our implementation, however, we set several epochs so that we can measure the time it takes for each implementation. The algorithm runs as follows:

- choose k random observations as your initial centroids or means
- Repeat for a specified number of epochs or until convergence:
  - Calculate the distance of each data point with all the means.
  - Assign each data point to a cluster with the least distance
  - recalculate the means by assigning the mean to the average means of the data points that belong to that specific cluster

Upon termination of the algorithm, each observation will belong to a specific cluster, in the case of our project, these observations will belong to one of three cluster.

2.1.2 Fuzzy-C Means

The difference between Fuzzy-C Means and K-Means is that K-Means partitions the data into different clusters, Fuzzy C-Means on the other hand assigns membership weight of each observation to a cluster. For example, let **x** be an observation, after running the K-Means algorithm, **x** can have the following classification: $[0, 1, 0]$, which means it belongs to second class or cluster. After running Fuzzy C-Means however, **x** can take an infinite possibilities, e.g

$[0.12445, 0.64587, 0.24654]$ this means that the observation weighs more to the middle class then the left class then the right class.
The algorithm works as follows:

- Choose a number of classes **C**, in our case this will be 3.
- Randomly assign coefficients of observations being belonging to a cluster, let's call this the coefficients matrix **W**.
- Repeat until convergence or for several iterations:
  - compute the centroids or means for each cluster as follows

$$\mathbf{c_k} = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m} \tag{2}$$

  - for each observation, computer it's coefficient of belonging to a cluster as follows:

$$w_{ij} = \frac{1}{\sum_{k=1}^{c}\left(\frac{\|\mathbf{x}_i - \mathbf{c}_j\|}{\|\mathbf{x}_i - \mathbf{c}_k\|}\right)^{\frac{1}{m-1}}} \tag{3}$$

$m \geq 1$ in the above equation is the Fuzzy measure. m is a hyperparameter, the bigger m gets, the fuzzier the values. The smaller it is, the less fuzzy the coefficients will be.If m is equal to one, then the algorithm becomes K-Means, this means that there isn't any fuzziness in the algorithm and clusters are disjoint. This means that there is a perfect partition of the dataset into clusters.

2.2 Solution implementations

The results of both algorithms are dependent on their initializations. The K-Means initializes centroids and Fuzzy C-Means initializes the coefficients matrix. For the K-Means, we initialized the clusters as the first 3 data points in the datasets, this helps to measure the correctness of all three implementations. As for the Fuzzy C-Means, we wrote a utility file that creates initializations for the coefficients and stores these coefficients into a csv file. All three implementations then read the dataset as well as the coefficients into their environment. The approaches above aid in validating the correctness of the implemented algorithms.

2.2.1 Serial Implementation

The serial was exactly as described in the the algorithms section above.

2.2.2 MPI Parallel Implementation

To optimize for performance, we need to reduce the number of communications among processors. We can achieve this either by avoiding communications which can be difficult to do or by running the algorithm on a small number of processors.

K-Means

We began by dividing the data set almost equally across the data points. If the dataset can not be evenly split among the processors, then the last processors take the remainder of the data points. Each processor then calculates the number distance of its data points with all available clusters and assigns the data point to the nearest cluster. Upon distance calculations, each processor except the MASTER processors sends the distances as well as the assigning of clusters to the MASTER thread. The MASTER thread then broadcasts these results to every thread in the program. This is the first phase of the algorithm.

In the second phase, only c number of threads run, c is the number of clusters. Each running thread recalculates the position of the centroid assigned to it. The processors send these recalculated centroids to the MASTER thread and the MASTER thread broadcasts these centroids to every thread in the program. The above-mentioned process happens for several iterations.

Fuzzy C-Means

The process here is the opposite of that in K-Means. We start with c number of threads among all available threads. These compute the centroids since we already have the initialized coefficient matrix. Each thread in the first phase computes the centroids and sends these centroids to the MASTER thread. The MASTER thread then broadcasts these to all threads in the program.

In the second phase, every thread in the program does approximately the same amount of work depending on whether the data points could be split evenly across the available threads. If the data can not be split evenly, the last thread takes the remainder of the data points. Each thread then calculates the coefficients of each data point. Upon calculation, every thread sends its chunk to the MASTER thread and the MASTER thread broadcasts the results back to all the threads. At this point, the program is ready to move on to the next iteration.

2.2.3 CUDA Parallel Implementation

The beauty of CUDA is that we have the power to assign each independent piece of work to a dedicated thread or processor. In addition to the above superpower, We have the privilege of updating a shared memory buffer. We, therefore, hope that the CUDA implementation will give us the best performance.

K-Means

Initially, we wanted to dedicate a block of threads with threads equalling the number of observations in our dataset. Each block would then be dedicated to each cluster. Because we couldn't find the problem we were facing, we resorted to using one block running c times, where c is the number of clusters. For each cluster c, there are n threads, each thread calculates the distance of its assigned data point to all clusters and assigns that data point to the appropriate cluster. Since the updates are on the shared memory, no communications are done here. This marks the end of phase one.

In the second phase, we calculate the centroids, again, we would have loved to dedicate a separate block for each cluster, due to time we couldn't, this was, however, not a complete train smash as we were still able to parallelize the second phase. Since we are iterating through the buffer that stores data about data points being assigned to clusters, We only use c threads among thousands available to us. Each thread recomputes the centroid. Upon completion, the program is ready to move to the next iteration.

Fuzzy C-Means

You probably already know what goes on in this section. In the first phase, we dedicate c number of threads to compute the centroids given the coefficient matrix. Upon centroid recomputation, we move to the next phase.

In the second phase, we make use of n threads, where n is the number of observations. Each thread calculates the coefficients of the assigned data point for all clusters. Upon coefficient recomputation, the program is ready to move to the next iteration.

2.3 Evaluation Methods

We tested for two things in all our implementations, correctness and performance. We measured the performance in milliseconds. The measure of correctness was dependent on initializations. I believe we've discussed the approach we took in trying to measure correctness.

3 Experimental Setup

3.1 Dataset Description

The dataset used here is used here is taken from UCI Repository,

The data set contains 178 observations and 13 features. This Dataset is owned by Forina, M. et al, PARVUS, and donated by Stephen Aeberhard. It was donated in the year 1991. The data consists of no missing values and it contains integers and real numbers. The features describe the quality of the wine. The features are alcohol level, Malic acid, Ash, Alkalinity of the Ash, Magnesium level, total phenols, non-flavonoid phenols, Proanthocyanins, Color intensity, Hue, property of diluted wines, and Proline. The data set is compiled to specifically classify the origin of the wine out of 3 regions. To stay as professional as possible, we kept the number of clusters fixed to 3.

3.2 Performance Evaluation Approaches

We measured performance primarily as a function of the number of epochs. One interesting factor we could have considered was the number of clusters we could have used, but we however stuck to using only three as specified by the providers of the dataset.We only measure the performance of the parts that run the algorithm. The initializations, as well as the finalizations, are not included in the results that we are going to discuss below.

4 Results and Discusions

4.1 K-Means

Below are the results stored in the table format for different implementations.

K Means Performance Measure in ms			
algorithm	10 Epochs	100 Epochs	500 Epochs
Serial	8.248	83.23	383.636
MPI(8)	2.708	13.107	57.255
CUDA			

Above is the table for result the algorithm after different epochs. The 8 brackets for MPI means that it was run on 8 threads.

4.2 Fuzzy C-Means