Unsupervised-based Distributed Machine Learning for Efficient Data Clustering and Prediction

Vishnu Vardhan Baligodugula

Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/2791

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact library-corescholar@wright.edu.
UNSUPERVISED-BASED DISTRIBUTED MACHINE LEARNING FOR EFFICIENT DATA CLUSTERING AND PREDICTION

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science

By

VISHNU VARDHAN BALIGODUGULA
B.Tech., Koneru Lakshmaiah University, 2021

2023
Wright State University

Fathi Amsaad, Ph.D.
Thesis Director

Thomas Wischgoll, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination:

Fathi Amsaad, Ph.D

Travis E. Doom, Ph.D

Subhashini Ganapathy, Ph.D

Vincent Schmidt, Ph.D

Shu Schiller, Ph.D.
Interim Dean, College of Graduate Programs & Honors Studies
Abstract


Machine learning techniques utilize training data samples to help understand, predict, classify, and make valuable decisions for different applications such as medicine, email filtering, speech recognition, agriculture, and computer vision, where it is challenging or unfeasible to produce traditional algorithms to accomplish the needed tasks. Unsupervised ML-based approaches have emerged for building groups of data samples known as data clusters for driving necessary decisions about these data samples and helping solve challenges in critical applications. Data clustering is used in multiple fields, including health, finance, social networks, education, and science. Sequential processing of clustering algorithms, like the K-Means, Minibatch K-Means, and Fuzzy C-Means algorithms, takes a long time, especially with many data samples, regardless of whether the results obtained may be accurate or not.

This thesis proposes parallel and distributed computing unsupervised ML techniques to improve the execution time of different ML algorithms. The application of different ML techniques on each system and their specific variations is outlined. Various parallelized unsupervised ML models are developed, implemented, and tested to demonstrate the efficiency, in terms of execution time and accuracy, of the serial methods as compared to the parallelized ones. For that, parallel K-Means, parallel
Minibatch K-Means, and Fuzzy parallel C-Means using an MPI model are developed. A distributed time estimation approach is created that utilizes the AWS cloud computing architecture. The Sequential, Parallel, and distributed approaches of K-Means, Minibatch K-means, and Fuzzy C-Means are investigated to enhance the outcome of the developed models. The strengths and weaknesses of various ML-based algorithms are analyzed.

As a case study, a country dataset for multiple organizations is used to provide financial assistance to nations based on socioeconomic and health factors and use K-Means, Minibatch K-Means, and Fuzzy C-Means sequential Parallel and distributing techniques like AWS to analyze the data. We developed a serial, parallel, and distributive computing technology based on ML AWS architecture to determine the most efficient method through comparative analysis and our research investigations to provide K-Means, Minibatch K-Means, and Fuzzy C-Means execution timings.

Our results reveal that Minibatch K-Means outperforms the other two clustering methods in sequential execution while outperforming them in parallel execution. It is observed that all the developed models perform better in the sequential model than in the parallel model. This work concludes that execution times reduce when these models are implemented on distributed platforms, i.e., Amazon SageMaker, a cloud computing platform, with no noticeable impact on the accuracy of the developed models.
Contents

1 INTRODUCTION 1

1.1 Outline ................................................. 1

1.2 Chapter Overview ...................................... 3

2 BACKGROUND AND RELATED WORK 4

2.1 Machine Learning ....................................... 4

2.1.1 K-Means Algorithm .................................... 6

2.1.2 Minibatch K-Means .................................... 11

2.1.3 Fuzzy C-Means ........................................ 12

2.2 Message Passing Interface .............................. 13

2.2.1 Features of the Message Passing Interface that are advantageous ........................................... 14

2.2.2 Flexibility ........................................... 14

2.2.3 Terminology used by MPI: Key terms and instructions ........ 15

2.2.4 MPI’s Past and Iterations ............................. 17
2.3 AWS Sage Maker ............................................. 18
  2.3.1 Overview .................................................. 18
  2.3.2 AWS .......................................................... 18
  2.3.3 About AWS Sage Maker .................................. 18
  2.3.4 Attributes of Sage Maker .................................. 19
  2.3.5 Utilizing AWS Sage Maker and machine learning ........... 20
  2.3.6 Modern Developments ..................................... 23
  2.3.7 Conclusion .................................................. 23

2.4 Related Work .................................................. 23

3 PARALLELIZATION STRATEGIES 36

4 CONTRIBUTION OF RESEARCH TO K-MEAMS, MINIBATCH
  K-MEANS AND FUZZY C-MEANS 38
  4.1 Sequential Process of K-Means ............................... 38
  4.2 Minibatch K-Means ............................................ 41
  4.3 Fuzzy C-Means .................................................. 43
  4.4 Parallel Techniques using Message Passing Interface ........ 44
    4.4.1 Steps involved in Parallel Techniques using MPI .......... 46
    4.4.2 MPI methodology consists of several stages ............. 47
4.5 Amazon Sage Maker ................................................. 48
  4.5.1 Working of parallel clustering techniques using MPI in AWS . 48
  4.5.2 Steps to generate a Sage Maker notebook example .......... 50
4.6 Implementation ..................................................... 51
  4.6.1 Data Collection .................................................. 51
  4.6.2 Data Pre-Processing .......................................... 52
  4.6.3 Data Transformation ........................................... 53
  4.6.4 Implementation of Clustering Algorithms ..................... 53
  4.6.5 Pattern Information .......................................... 57
  4.6.6 Comparison of various Techniques ................................ 58
4.7 Setup of Exploration ............................................... 58

5 Evaluation Matrices .................................................. 59
  5.1 Confusion Matrix .................................................. 59
  5.2 Precision ............................................................ 60
  5.3 Recall ................................................................. 60
  5.4 F1 score ............................................................. 60
  5.5 Support .............................................................. 61
  5.6 Accuracy ............................................................ 61
6 RESULTS AND CONCLUSION

6.1 Elbow method

6.2 Discussion

6.3 Conclusion

6.4 Future Work

7 APPLICATIONS

7.1 Customer Segmentation

7.2 Document Classification

7.3 Identifying crime hotspots

7.4 Delivery store optimization

7.5 Ridesharing data analysis

7.6 Analysis of call record details

7.7 Detecting insurance fraud

7.8 Criminals who use cyber-profiling

7.9 Fully automated clustering of IT notifications

7.10 Prediction for the stock market

7.11 High dimensional text data

REFERENCES
<table>
<thead>
<tr>
<th>A</th>
<th>Appendix</th>
<th>89</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Appendix</td>
<td>93</td>
</tr>
<tr>
<td>C</td>
<td>Appendix</td>
<td>100</td>
</tr>
<tr>
<td>D</td>
<td>Appendix</td>
<td>104</td>
</tr>
</tbody>
</table>
## List of Figures

2.1 Basic Model of Machine Learning ........................................ 5  
2.2 Before K-Means and after K-Means ................................... 7  
2.3 Clusters formation ............................................................ 9  
2.4 Elbow Method Graph Representation .................................. 12  
2.5 The rank assigned to every CPU is demonstrated using an MPI COMM process with numerous endpoints distributed across four clusters. . . . 17  
2.6 Model of AWS sage maker. ..................................................... 19  
4.1 Flowchart of the of K-Means Algorithm. ............................. 39  
4.2 A description of how to put the divide and combine plan into practice of Parallel clustering techniques using MPI. ............................. 45  
4.3 Flow of Parallel Techniques using MPI. ............................... 46  
4.4 How K-Means works in Amazon Sage Maker. ........................ 49  
4.5 Implementation Process. ..................................................... 51  
6.1 Distortion Score Elbow Curve for Clustering. .......................... 62
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2 Clusters formation for Sequential K-Means</td>
<td>63</td>
</tr>
<tr>
<td>6.3 Clusters formation for Parallel K-Means using MPI</td>
<td>63</td>
</tr>
<tr>
<td>6.4 Clusters formation for Parallel K-Means using MPI in Amazon Sage Maker</td>
<td>64</td>
</tr>
<tr>
<td>6.5 Clusters formation for Sequential Minibatch K-Means</td>
<td>64</td>
</tr>
<tr>
<td>6.6 Clusters formation for Parallel Minibatch K-Means using MPI</td>
<td>65</td>
</tr>
<tr>
<td>6.7 Clusters formation for Parallel Minibatch K-Means using MPI in Amazon Sage Maker</td>
<td>65</td>
</tr>
<tr>
<td>6.8 Clusters formation for Sequential Fuzzy C-Means</td>
<td>66</td>
</tr>
<tr>
<td>6.9 Clusters formation for Parallel Fuzzy C-Means using MPI</td>
<td>66</td>
</tr>
<tr>
<td>6.10 Clusters formation for Parallel Fuzzy C-Means using MPI in Amazon Sage Maker</td>
<td>67</td>
</tr>
<tr>
<td>6.11 Confusion matrix of Sequential K-Means</td>
<td>68</td>
</tr>
<tr>
<td>6.12 Confusion matrix of Sequential Minibatch K-Means</td>
<td>68</td>
</tr>
<tr>
<td>6.13 Confusion matrix of Sequential Fuzzy C-Means</td>
<td>69</td>
</tr>
<tr>
<td>6.14 Comparison of Sequential Clustering techniques</td>
<td>70</td>
</tr>
<tr>
<td>6.15 Comparison of Parallel Clustering techniques</td>
<td>71</td>
</tr>
<tr>
<td>6.16 Comparison of Cloud Clustering techniques</td>
<td>72</td>
</tr>
<tr>
<td>6.17 Comparison of Execution times in all techniques</td>
<td>73</td>
</tr>
</tbody>
</table>
6.18 Help needed per country (World)
List of Tables

5.1 Confusion matrix .......................................................... 59

6.1 Evaluation Matrix of Sequential Clustering .......................... 69

6.2 Evaluation Matrix of Parallel Clustering using MPI ............... 70

6.3 Evaluation Matrix of Parallel Clustering using MPI in Amazon Sage Maker ......................................................... 71

6.4 Comparing execution time of all techniques ......................... 73
Acknowledgment

I would like to express my heartfelt gratitude to the individuals who have contributed to my knowledge through this thesis. Firstly, I would like to thank the Department of Computer Science and Engineering at Wright State University for fostering an environment that promotes collaboration between students and faculty to push the frontiers of research forward every day.

Moreover, I extend my sincere appreciation to my thesis advisor, Dr. FATHI AMSAAD, for his guidance in teaching me the intricacies of distributed computing and inspiring me with his unwavering optimism and positive energy. His support and encouragement have been instrumental in the successful completion of this thesis.

I would like to express my heartfelt dedication to my family BALIGODUGULA PULLAIAH, BALIGODUGULA LAKSHMAMMA, and BALIGODUGULA VAMSI SAI KRISHNA, for their unwavering support throughout my academic journey. Their constant guidance, encouragement, and sacrifices have been the cornerstone of my success, and I am immensely grateful for their love and support.

In addition, I would like to express my appreciation for the immense support and encouragement that my dear friends, DEEPIKA GADIRAJU and AKHIL SABBELLA, have given me. Their unwavering presence and companionship have been a source of comfort and strength, especially during challenging times. I feel grateful for their genuine friendship and the positive impact they have had on my life.
1 INTRODUCTION

1.1 Outline

In machine learning, one of the primary goals is to extract valuable insights or patterns from large datasets. One widely adopted technique for grouping similar data is the K-Means algorithm. K-Means works by evaluating the similarity among distinct objects within a dataset and subsequently grouping them based on these similarities. With the growing volume of data being amassed using scientific data collection techniques, it has become increasingly imperative to ensure that machine learning algorithms are effective, efficient, and scalable. One way to improve machine learning algorithms’ performance is to parallelize them, thereby leveraging modern computing infrastructures. However, the efficacy of parallelization is heavily contingent upon the data organization and choice of parallelization approach.

The K-Means algorithm is one of the first and most commonly utilized techniques. There are a few issues related to the K-Means algorithm. They are as follows: one is the initial centers initialized randomly, which is a significant issue. The other issue is that there will need to be some idea of the clusters in the dataset. Minibatch K-Means is a clustering algorithm that splits a dataset into a fixed number of groups or clusters. Instead of processing the entire dataset, Minibatch K-Means works on smaller, randomly chosen subsets of data known as mini-batches. This approach makes it more efficient and capable of handling larger datasets that may not fit into memory. The algorithm assigns each data point to the nearest centroid and updates
centroid positions based on the mean of the assigned data points. While Minibatch K-Means is faster and more scalable, it may only sometimes produce optimal results compared to the standard K-Means algorithm, mainly when the mini-batch size is too small. Fuzzy C-Means (FCM) is an effective clustering algorithm that groups data points into distinct clusters based on their similarities. This unique algorithm enables overlapping clusters by ascribing a fuzzy membership value to each point, signifying its membership degree to each cluster. FCM continually calculates and updates the cluster centers and membership values until a predefined stopping point is reached. Many scholars have concentrated on parallelizing these techniques based on these issues.

Nevertheless, all these algorithms have a few drawbacks, such as their parallel systems having constrained programming models that automatically parallelize the data line by line. They assumed all the data would have to be kept in memory line-by-line simultaneously, which is highly impossible when dealing with extensive data collections with millions of records. Sometimes, speed is achieved, but efficiency is missed, and vice versa. For better performance, both efficiency and accuracy should be achieved.

This research investigated multiple Parallel and distributing approaches and implementation methods for the K-Means, Minibatch K-Means, and Fuzzy C-Means algorithms. We created a tool ”Non-Governmental Organizations(NGOs)” may use to provide developing countries with funding for necessities and assistance during disasters and environmental harm. We divided the nations into categories based on the economic and medical factors that will contribute to their overall development. Organizations can determine which country needs support with the aid of analysis. K-Means, minibatch K-Means, Fuzzy C-Means serial approach, open MPI parallel technique, and AWS distribution strategy were all used in the investigation.
In this study, we implemented the task using a parallelized MPI-based algorithm and an implementation of AWS for computing distribution from scratch, as well as serial K-Means, Minibatch K-Means, and Fuzzy C-Means utilizing the elbow approach in Python. For several reasons, including obtaining the precision and speed with which the result is found, MPI is one of the best parallel computing approaches. We tested the speed and accuracy of the algorithms using the three different ways to make the comparison, and we then presented the findings.

1.2 Chapter Overview

The chapters are arranged as follows: Chapter 2 provides background information on machine learning algorithms, Message Passing Interface, Amazon SageMaker, and related work. Chapter 3 focuses on parallelization strategies, describing the methods and requirements necessary for parallelization. Chapter 4 details the research contributions to K-Means, MiniBatch K-Means, and Fuzzy C-Means, as well as the understanding of parallelization and cloud algorithms, and the experimental setup. Chapter 5 covers the evaluation matrix. Chapter 6 presents the results, discussions, and conclusions regarding the performance of cloud and parallel clustering distribution techniques versus sequential clustering. Chapter 7 explores potential applications. Finally, the References and Appendix are included.
2 BACKGROUND AND RELATED WORK

This chapter will cover four major topics: Machine Learning models, Parallel Message passing interface, AWS Sage maker, and related work.

2.1 Machine Learning

Machine learning is a subclass of artificial intelligence which allows computer systems to improve performance by learning from experiences without human instruction. It entails using large datasets to train algorithms capable of detecting patterns, making predictions, and enhancing decision-making abilities. Machine learning is classified into three main categories: supervised learning, unsupervised learning, and reinforcement learning, each having its unique techniques and areas of application.

Supervised learning deals with labeled data wherein the algorithm learns to recognize patterns, relationships, and correlations between inputs and outputs to produce accurate predictions or classifications. Unsupervised learning, on the other hand, works with unlabeled data, which means that the algorithm must identify and extract insights from the data by itself without any predefined labels. Reinforcement learning, often used in robotics and game development, relies on trial-and-error to learn the best approach to solve a particular task.

As machine learning continues to evolve, it has found applications across several industries, including healthcare, finance, retail, and transportation. It has enabled
organizations to analyze diverse data sources, make informed decisions, and increase operational efficiencies. With the growing availability of data and processing power, the potential for machine learning is enormous, and its impact on various industries is expected to be transformative and here I am presenting basic model of ML Figure 2.1.
We are using unsupervised learning and these are the algorithms we used in this study: -

- K-Means Algorithm.
- Minibatch K-means Algorithm.
- Fuzzy C-Means Algorithm.

2.1.1 K-Means Algorithm

These grouping issues in machine learning or information science are resolved using unsupervised learning K-Means Clustering. In this chapter, we will study the K-means clustering method, how something operates, and how to apply it in Python.

Unsupervised training method K-Means Cluster divides a piece of unlabeled information into various clusters. Thus, K describes how several clusters that were not previously present should be generated due to the procedure; for instance, if K=2, the outcome will primarily be two types. If K=3, the outcome will be three clusters, and so on.

This iterative method separates the unlabeled information into k distinct clusters, each containing one dataset and sharing a set of features. It allows us both to categorize all data into different groups. It offers a workable technique for quickly and effectively determining the groups within the unlabeled dataset without having to do any training.

Every cluster has a corresponding centroid in this centroid-based approach. The principal objective of such an approach is to reduce the distance between the data points and the groups they belong.
This method starts with an intake of an unlabeled dataset, separates this into
k clusters, and continues the procedure until no better clusters are found. In this
method, the k value must be known in advance. Its two main functions of the k-means
suggested method are:

- Determines the optimal quantity of K points or centroids via an iterative
  algorithm.
- Each piece of information is paired with the closest k-center. Those information
  fragments near a certain k-center combine to form a cluster.

Each cluster is unique and has samples that only share a small amount in a joint.
A K-Means Clustering Method is shown in the Figure below Figure 2.2:

![Figure 2.2: Before K-Means and after K-Means](image)

The following stages illustrate how the K-Means method functions:

1. Choose the number of groups, K.
2. Pick K locations or cluster centers randomly. (These may not be from the
   supplied data.)
3. Allocate every piece of data to its nearest centroid to create the predetermined
   number of clusters.
4. Determine the variation and relocate every cluster’s centroid.

5. Reassign every piece of data to an updated centroid for each cluster by repeating the previous step.

6. If there are still data reassignments, go back to step 4. Otherwise, proceed to step 7.

7. The clustering process is finished.

Now let us analyze the graphical plots to comprehend the steps mentioned earlier:

Below points is the explanation of graphs each point representing each graph formation Figure 2.3.

- Consider that there are two independent variables, P1 and P2.

- To identify the data set and divide it into various clusters, let us use a value of k for clusters or K=2. This implies that we will divide those data sets into two separate groups. To create a cluster, k points or the centroid must be chosen randomly. These coordinates may be taken from the dataset or in different locations. Since these two do not appear in the dataset, it was choosing them as the k points in this instance.

- Currently, we would identify the nearest K-point and centroid for every data point on the scatter plot. To find the distance separating two points, we will estimate it using the math we studied. To create a midpoint between both, the two centroids shall do such.

- It is “evident from Figure 2.3” from the previous picture that locations along the line to the left were near a K1 center, which is red, and white spots along the
lines to the right are near a red centroid. For better comprehension, let us color both red and green. We will repeat the procedure by picking a new centroid to locate the nearest cluster. Let us determine those centroids centers of gravity ordered to select their fresh centroids.

- We would then reallocate every piece of data here to a new centroid. "conduc" the same procedure for locating a mean line in this, and your average should look such as the illustration of 5 one in Figure 2.3. There is one red point on

Figure 2.3: Clusters formation
the left end of the line, plus two green points on the lines. Thus, fresh centroids will be assigned to such three locations.

- Since reassign has occurred, we once more go to step 4, locating fresh centroids called K-points. Determining the exact gravitational centers of a new centroid would allow us to continue the operation.

- We will construct the average lines once more and reassign all pieces of data because we have the updated cluster centers.

- This is established due to the need for more consistent data points along both sides of the boundary.

- Now that our model is complete, we may exclude the two remaining groups and the presumptive centroids.

**Elbow Method**

The Elbow Method is widely used for determining the ideal number of clusters. A WCSS-valued idea is applied in this technique. The term “total variations inside a cluster” is abbreviated as ”WCSS” and means Within Cluster Sum of Squares. This Formula 2.1 is used to determine the number for WCSS (for 2 clusters):

\[
WCSS = \sum_{Pi \text{ in Cluster 1}} \text{distance}(Pi C1)^2 + \sum_{Pi \text{ in Cluster 2}} \text{distance}(Pi C2)^2
\]

From the formula,

\[\sum_{\text{Pi in Cluster 1}} \text{distance}(Pi C1)^2: \text{The remaining two parameters are the}\]
product of the squares of an interval between every data point and its centroid inside a cluster.

We can use any method, including the Manhattan distance or even the Euclidean distance, to calculate that separation between the data points and the centroid. The elbows approach goes through the procedures listed below to determine the best value of groupings:

- On such a given data, it does K-Means grouping with various Values of k (ranges from 1-10).
- Determine the WCSS values for each K value.
- Traces a curve from the estimated WCSS numbers to the K-cluster count, and when a bend’s sharp point or plot point resembles an arm, this location has the maximum value.

Its elbows technique is so named because the diagram depicts a steep bend that resembles an elbow. A graph for the elbows approach resembles what illustration Figure 2.4

2.1.2 Minibatch K-Means

Minibatch K-means is a widely used clustering algorithm that divides a given dataset into K clusters by minimizing the distances between the data points and their associated cluster centroid. In contrast to the standard K-means algorithm, Minibatch K-means processes the data’s small, randomly selected subsets, or Minibatches, to expedite convergence while maintaining comparable clustering accuracy. It is considered efficient and scalable for large datasets that exceed the available memory capacity. The algorithm assigns each data point to its closest
cluster center and updates the centroids iteratively until convergence is achieved. The number of Minibatches and the batch size are hyperparameters that can be optimized to enhance the clustering performance and accuracy of the resulting partitions.

2.1.3 Fuzzy C-Means

Fuzzy C-means is a prominent unsupervised clustering approach used to group data points into distinct clusters based on their similarity. Unlike complex clustering methods such as K-means, Fuzzy C-means assigns each data point a degree of membership, represented as a fuzzy value, for every cluster. This allows a data point to be part of multiple clusters at the same time with different membership degrees. The algorithm adapts the cluster centers and their fuzzy memberships iteratively based on the data point similarity and the level of overlap among the clusters. The hyperparameter ‘fuzziness’ regulates the degree of blurriness of the memberships, with higher values leading to more fuzziness and an increased number of clusters. Fuzzy C-means can be utilized to address various clustering problems and provide soft classification outcomes for the data points.
2.2 Message Passing Interface

A standard way to communicate throughout many computers that run concurrent programs all over distributed storage is indeed the message-passing interface (MPI).

The term "node" refers to a group of processors, or even a group of core processors on a single computer, in parallel Computing. Generally, every node inside a parallel configuration focuses on a distinct aspect of an immense processing challenge. It thus becomes difficult to coordinate the activities of every simultaneous network, transfer data across nodes, or exert control over the entire parallel cluster. The message-forwarding interface defines a standardized set of commands for such activities. When a message is sent to an entity, parallel process, subroutine, function, and thread, it is usually called "passing a message." That message is again utilized to begin a different technique.

Although no formal reference implementation, such as the International Organization for Standardization or the IEEE Institute of Electrical and Electronics Engineers, had supported MPI as a benchmark, it is widely regarded as an industry norm. It serves as the base for most network systems used by parallel computing programming. Researchers also produced several MPI versions.

Fortran, C, C++, Python and Java procedures or libraries have usable syntax defined by MPI.
2.2.1 Features of the Message Passing Interface that are advantageous

Standardization

Previous message queue packages have been supplanted by MPI, which is now a widely recognized mainstream technology.

A large group formulated it

"MPI was not designed through a nationalized standard process", a committee of manufacturers, integrators, and users came up with it.

2.2.2 Flexibility

Since MPI has already been developed for numerous distributed memory designs, users can move their software to other systems which the MPI standard supports without changing the source code.

Efficiency

Usually, functionality is tailored to MPI’s device, and vendor implementations could be tuned to take advantage of built-in hardware characteristics.

Feature set

Maximum performance on parallel processing systems and groups is a crucial feature of MPI. Furthermore, over 100 specified routines make up the fundamental
MPI-1 architecture.

### 2.2.3 Terminology used by MPI: Key terms and instructions

A few fundamental MPI ideas or "instructions include Figure 2.5:"

**Comm**

This MPI communication classes link several stages. An enclosed consultation processes an individual identifier from a communication instruction, which arranges it into an orderly topology. For instance, MPI COMM WORLD is a signal for global communication.

**Color**

This means that a process is assigned a color, and all activities with that coloring are housed within the same communication. "MPE MAKE COLOR ARRAY" is a color-related instruction that modifies the offering of the product.

**Key**

A key determines the ranking or ordering of a procedure in communication. The order is established by the application’s position inside the communication if two methods are assigned the same key.

**New comm**

Fresh communication can be created using this instruction.
Categories of data collected

A description of the kind of data exchanged among tasks is required for MPI functions. These variables can be predefined with the help of MPI INT, MPI CHAR, or MPI DOUBLE.

Refer

It communicates among two particular operations. Two popular delaying techniques for juncture messaging were MPI Send and MPI Recv. Block is the method by which the transmitting and receiving systems hold off on transmitting and finishing a communication until a whole signal has been appropriately transmitted and received.

Cooperative foundations

All activities in a particular process must communicate to perform these group tasks. However, one method is MPI Bcast, which distributes information from a single node to each process in a process.

One-sided

Usually, when this phrase refers to external communications like MPI Put, MPI Get, or MPI Accumulate. Publishing to storage, receiving to recollection, or minimizing operations on a single recollection between activities are specially mentioned.
2.2.4 MPI’s Past and Iterations

Around 1991, a tiny research group from Austria started talking about a message-carrying interface. The Center for Studies in Parallel Computing supported a workshop about message-passing protocols inside a distributed memory system that occurred in Williamsburg, Virginia, per year afterward. As a result, a work was organized in order to develop the standard approach.

The prototype of MPI-1 was developed in Nov 1992, and the specification was published in 1993 just at Supercomputing ’93 conference. Around 1994, MPI version 1.0 was published after receiving more comments and modifications. Since then, MPI has been open to everyone working in the powerful computational field, with more than 40 companies currently involved.

About 115 capabilities are offered under the more dated MPI 1.3 specification, sometimes known as MPI-1, and more than 500 features and a high degree of older systems using MPI-1, a subsequent MPI 2.2 specification, or MPI-2, are available.
Unfortunately, a complete MPI-2 version is not offered by all MPI implementations. Along with variable system integration and remote storage functions, MPI-2 introduced novel concurrent I/O. The Nov. introduction of the MPI-3 specification results in increased speed, multicore or cluster capabilities, and improved scalability, with availability and high interoperability. MPI 4.0 was published by The MPI Forum around 2021. It featured innovative features like Permanent Communes, segmented networking, and a new product layout. Now, MPI 5.0 is being created.

2.3 AWS Sage Maker

2.3.1 Overview

Most data analysts use the hosted platform to develop, train, and publish machine learning algorithms, and they were regrettably still unable to change resources as required. To go live quicker and with less expense, AWS Sage Maker makes it simpler for programmers to design and retrain models. This section review the capabilities, use applications, and advantages of AWS Sage Maker and Computer Vision using AWS Sage Maker.

2.3.2 AWS

A cloud-based system that offers services that are ordered over the web is known as Amazon Web Services. Every public cloud type may be developed, monitored, and deployed using AWS services, which is where AWS Sage Maker is useful.

2.3.3 About AWS Sage Maker

Amazon manages the machine learning services offered as Sage Maker. Sage Maker enables data researchers or programmers to create machine learning models quickly, train those, and afterward instantly release them into a virtual machine that
is prepared for use in operation. It usually includes methods for machine learning designed for application in a distributed scenario using extensive data sets. Sage Maker provides flexible, dispersed education options that ”fit existing operations as shown in Figure 2.6”.

![Figure 2.6: Model of AWS sage maker.](image)

### 2.3.4 Attributes of Sage Maker

Sage Maker that Amazon has developed since its debut in 2017. Accessibility to functionality is made possible via AWS Contributory factor Studios, an Integrated Development Environment that combines all abilities. One of two methods can be utilized to build a Jupyter notebook:

- Sage Maker Studio’s internet version of the IDE.
• A machine learning server in Amazon Sage Maker that Amazon EC2 hosts.

2.3.5 Utilizing AWS Sage Maker and machine learning

Let us examine the development, testing, fine-tuning, then deployment of a model for machine learning utilizing AWS Sage Maker.

Builds

• There are approximately 15 popular learning algorithm skills and making included.

• It enables us to select the configuration settings needed for the notebook instances.

• To start coding, a user may use a notebook example (for building model training tasks).

• Choose and enhance the necessary methods, such as: K- Means, Linear Regression and Logic-based Regression.

• AWS Sage Maker’s Jupyter notebook API allows programmers to alter Supervised Learning servers.

Tune and Test

• All required modules should be made or imported. During Amazon model development, establish and control just a few configuration settings.

• The model is taught and tuned using Sage Maker.
• Sage Maker achieves a set of parameter tuning by combining some algorithm parameters.

• Sage Maker uses Amazon S3 for keeping records as it is a secure protocol.

• Sage Maker uses extensible ECR to control Container technology.

• Docker setup, management, and storage are made more accessible with ECR.

• The developed skills source is preserved in ECR, whereas the supervised learning is separated and stored in Amazon S3.

• Next, Sage Maker generates, learns, and stores a given input cluster in Amazon S3.

**Implement**

• Upon adjusting, objects could be released to Sage Maker destinations.

• Just in the end, a forecast is made in real-time.

• To determine whether the strategy has achieved your company goals, one must evaluate it.

**Uses of Sage Maker**

Many different businesses use AWS Sage Maker. Machine learning groups use Sage Maker for the below task:

• Accessing and exchanging codes.

• Accelerate the creation of production-ready AI components.
• Enhance data interpretations and build more exact database schemas iteratively.

• Streamline data both in and out.

• Much information is to be processed.

• Codes for modeling exchange.

Advantages

• Deep learning methods are easier to use.

• More people will be able to innovate using machine learning thanks to integrated tools like computer scientists and also no potential input for analysts.

• Huge data preparation.

• Collect, categorize, and analyze vast quantities of organized (tabular) and unorganized (photos, videos, and audio) information via computer vision.

• Accelerate the advancement of computer vision.

• Without improved facilities, learning can be completed in just a few moments instead of days.

• Using specially designed tools, employees to comply might rise as much as ten times.

• Enhance the machine training cycle to be more efficient.

• Streamline or unify machine learning operations practices throughout a company to design, train, publish, and maintain algorithms at volume.
2.3.6 Modern Developments

Since the initial release of reinvent 2021, Amazon has added a feature for its Sage Maker Python SDK that offers abstractions to speed up models’ deployments, plus Model Registry, which makes it easier to connect virtualized inferences destinations and MLOps processes. One can utilize Sage Maker Cloud Hosting Deduction with high-traffic applications now that Amazon has increased the provisions requests per endpoints limitation from 50 to 200.

2.3.7 Conclusion

AWS charges every sage Maker user for Computing, memory, or information computer resources used to create, test, deploy, and log machine learning algorithms or predictions. The expenditures of S3 were related to maintaining train and prediction data sets. The sage Maker application’s design is flexible and responsive, enabling the whole lifespan of machine-learning applications through modeling creation to model implementation. This implies that the sage Maker can be used independently for the proposed model, retraining, or distribution.

2.4 Related Work

Mohanavalli, Jaisakthi, and Aravindan [1] developed a parallel K-Means model based on both a distributed memory system using MPI programming and a memory version using OpenMP programming. They also tested a hybrid implementation that combined OpenMP with MPI. To evaluate the speedups and efficiency of the parallel algorithms, they used Amdahl’s law. The hybrid technique was found to be 50 percentage faster than MPI alone and performed better under a balanced load. [1]. They have demonstrated that this approach is added to increase the effectiveness of parallel K-Means. Both the Enslaver/Slave concept and the parallel processing
method are used. The tests revealed that this technique is more effective and applicable to Yufang Zhang’s situation[2]. Showed this is suggested to use an initiation technique for K-Means parallel processing, establish the first cluster centers, that not only speeds up performance but also produces consistent results conducted by Swamy, Raghuwanshi and Gholghate[3].

Boukhdhir, Lachiheb and Gouider’s [4]research focuses on increasing K-Means’ capacity to deal with large datasets by speeding up its execution, suggesting a map Reduce-based approach. In addition we will suggest two additional algorithms. The first eliminates unnecessary, and the other dynamically chooses the starting cluster center to stabilize the outcome. The suggested technique is demonstrated to be significantly quicker than three different known algorithms from the literature when implemented just on the Hadoop framework[4]. Applied Parallel MapReduce-based K-Means clustering approach because it is a straightforward yet effective method for parallel programming. These results of experiments indicate that the suggested algorithm can process massive datasets on affordable equipment while scaling up and down effectively research evaluated by Weizhong, Huifang and Qing [5].

The focus of this work is on improving a modified version of the K-Means clustering algorithm using both OpenMP and CUDA for CPU and GPU acceleration, respectively. The algorithm is tested on various datasets, including photos with varying amounts of data, with a particular emphasis on large datasets. The experiments cover various combinations of features and clusters to test the algorithm’s performance under different conditions.[6]. So, to speed up K-Means, the study presents two novel methods for transmitting data. Expanded Vector is the initial method (KMMR-EV). K-Means over MapReduce utilizing Boundaries File is indeed the name of the second technique (KMMR-BF). Compared to the simple MapReduce implementation of K-Means, both methods achieve speed up had undergone
considerable experimental investigation using actual and artificial data, coupled with an excess assessment to demonstrate the efficacy between both methods[7].

One goal of the researcher would be to create a K-Means method version that can run on a standard PC using NVIDIA graphics cards or tackle more extensive data sets using CUDA[8]. It concentrates upon cluster method performance problems, and a more sophisticated initial centroids core approach is used. The topic of a single CPU computer’s operational limit while dealing with large data sets also is investigated, and a parallel K-Means technique is investigated. The analysis shows that such advancements can significantly increase performance, allowing for the group of many data sets less precisely and fast analysis shown by Tian, Zhu, Zhang, and Liu[9].

Using GPU clusters, we demonstrate the conception and execution of an effective parallel K-Means algorithm. They dynamically use load balancing to evenly spread demand across the many GPUs inside the cluster to improve the overall cross-performance of the parallel K-Means. Additionally, they utilize software distributed shared memory to facilitate inter-node cooperation and communication. By keeping load balance on GPU clusters, the evaluation’s findings demonstrate the parallelism K-Means’ enhanced results.[10].

The latest update of K-Means separates overall issues into minor issues handled separately with one or more GPUs’ Broadcast Multiprocessors. They had developed the Graphics card K-Means (CUDA) authors[11]. It creates a buffer among collected data that did not alter their clusters over the following group, which might dramatically lower the burden of large data sets. Its operation time was reduced by up to 70 Percentage inside the prototype system[12].

On such a CPU-GPU heterogeneous system, they provide a quick implementation
of the spectral clustering technique. My method uses the multicore CPUs processing capability and the GPUs huge multithreading and SIMD abilities. They suggest a concurrent technique to generate a sparse refers to a network expressed in a common sparse codified form; the data points are given as input in high-dimension space. Next, using the ARPACK program, the CU SPARSE library’s backward transceiver k is ordinarily quite large—where the smallest k eigenvectors were computed by three of the Channel matrix. Additionally, we use GPUs to create a rapid parallel processing K-Means method, and the solution is much quicker[13]. A technique known as the enhanced K-Means method was used to expand that standard K-Means strategy. Our experimental findings demonstrate that such an expanded K-Means algorithm may effectively organize the data when the number of components for each grouping has to be constrained. This method changes the number of attributes in every category by utilizing a greedy strategy by faliu and inkyu[14].

This genetic algorithm is utilized to optimize K-Means grouping to ensure that the shortcomings of K-Means can be overcome. K-Means using the GA algorithm suggested innovative products in this field of research when the outcomes of conventional K-Means clustering versus genetics K-Means grouping were contrasted and analyzed[15].

In this work, three different programming paradigms are compared for accelerating the K-Means clustering algorithm: MPI for distributed memory, OpenMP for shared memory, and CUDA-C for heterogeneous computing on NVIDIA GPUs. The first investigation focuses on comparing these different approaches, while subsequent work has focused on optimizing the K-Means algorithm itself for speed. To select the most effective method, we compare several thermally activated processes in serial and choose the most efficient one for our method. We test our approach on various photos ranging from small (300x300 pixels) to large (1164x1200 pixels).
Results show that all three distributed programming paradigms yielded speedups, with OpenMP being more effective for smaller pictures and CUDA-C for larger pictures, according to Janki, Miriam, and Ningfang.[16]. provided a hybrid algorithm that decides when to employ the triangle inequality in an improved way. Numerous studies show that our approach performs better than the Computer or Multicore Kmeans[17].

The examination of share price pricing data from the prior years., with findings interpreted following intensive training using an algorithm for machine learning on CUDA, keeping in mind the time restrictions of trading. With the aid of machine learning approaches, the program’s performance has been significantly increased. In this research, a massively parallel methodology is applied to hasten the generation of findings. Compared to typical ways of employing a solitary Processor Core Unit, the execution time is significantly decreased due to the recent high achievements of CUDA parallel computing technology (CPU). By accurately anticipating stock prices in advance, it reduced computation time by a significant margin and resulted in net profits, which is the end objective of trade. Just characterized by three groups as projected utilizing algorithms, traders could choose to maintain the share, sell it, acquire new equities, or stay impartial. If a user makes a neutral choice, that indicates he must hold onto any stock he already has and refrain from purchasing any more. This suggested approach is appropriate for trading stocks based on stock price[18].

Its efficiency of a program in a concurrent environment was enhanced by adding a K-Means clustering algorithm to an MPI4py module. This report examines the effectiveness of executing the K-Means method consecutively versus using a Message Passing Interface (MPI) parallel design for grouping information in the form of operational costs and processing time[19]. This study presented a parallelization K-Means method for sparse, elevated text (PKHT). This suggested technique provides
an 11x shorter runtime by utilizing both GPUs with MPI by xiaolei, yanking, and yuxin[20].

Maximum performance, flexibility, and portability are all characteristics of MPI as a message-passing type system. This led to the motivation for this research to present MKmeans, a similar K-Means clustering technique with MPI. The approach allows efficient use of the clustering algorithm inside a parallel setting. Research using experimental statistics shows that MKmeans operates well on a variety of large amounts of data with very few time resources[21]. Researchers investigate leveraging NVIDIA GPUs written with CUDA C to enhance the speed of K-Means clustering. Various optimization methods are being used, including using constant memory for cluster data, shared memory, or picture data. Its outcomes were assessed on various images, ranging in size from tiny (256x256 pixels) to large (1024x1024 pixels) and with 4 to 256 clusters. For four clusters, we observe that now the serial version is typically 9x slower than the parallel version. Even as the number of clusters rises to 256, the speedup jumps to 57x[22].

We suggested a parallel implementation of the conventional K-Means algorithm to run it on the Hadoop distributed framework. The research findings show that when clustering a vast volume of data, our suggested K-Means method works better than conventional K-Means[23]. In terms of overhead expenses and execution, the overall efficiency of K-Means grouping the information is examined among parallel and sequential implementation in the Message Passing Interface Infrastructure by Ragunthar, Ashok, and Gopinath[24].

The study utilizes NVIDIA’s Compute Unified Device Architecture (CUDA) to create the GPU-based Harmony K-Means Algorithm in content clustering. During this testing, compared with CPU-based software, the GPU-based application could achieve a speedup of increase than Twenty times[25]. An approach suggested in this
research is based upon the modified K-Means algorithm of a Hadoop platform; initially, a preliminary clustering is obtained using the canopy algorithm [25], after which the pinpoint of the exact or cluster numbers is also adapted using the new ISODATA algorithm[25]. Lastly, integrate the MapReduce distributed computation architecture with the K-Means method to achieve the ideally planned CI-K-Means technique to improve the convergence point list and K calculated value by the Canopy method. The experiments show that now the CI-K-Means technique resolves the issues associated with the Canopy K - means algorithm’s inconsistent result in a more excellent and also the complexity of choosing the Centre for the K-Means approach. Its correctness, overall speedup proportion, and grouping efficiency have already been vastly enhanced compared to both approaches used before improvement[26].

They examine two examples of calculating dense similarity matrices to cluster using vast data sets. By sparsifying a vector, contrast one technique using the Nyström methodology. Next, decide to sparsify the matrices while keeping their closest neighbors, plus we look into their parallelization. On dispersed computers, we parallelize both memory usage and computations. We demonstrate the practicality of our parallel approach by conducting an empirical investigation on two colossal data sets: a large-scale photo dataset with 2,121,863 examples and a document data set with 193,844 examples[27].

The subsequent work has developed a probabilistic theory for grouping using a random point scientific process. This approach perfectly mimics Bayes decision theory regarding classification: Bayes clustering procedures available with the least predicted errors provided known underlying processes and a defined objective function. As a result, clustering becomes an ongoing method rather than a subjective one. In this study, we start to comprehend such an algorithm by presenting the circumstances where the given opportunity used in traditional K-Means clustering becomes efficient.
in the novel Bayes clustering concept[28].

The description of methodological improvements may allow for high computation savings in average squared data clustering. A parallel statistics program P-CLUSTER, which runs on a desktop system, now includes additional upgrades. Unsupervised categorization of photos with a familiar texture was the subject of investigations. A 96 decreasing trend in the calculation was made for some data sources[29].

The K-Means technique is suggested in this study as a single special instruction multiple data (SIMD) architecture processors (GPUs) driven algorithm. Both objects and the reconfiguration of k-centroids are assigned to the GPU concurrently inside this approach to minimize the computationally costly parts of the entire unit. With the most current development of GPUs with computational integrated system architectures, they have built such Graphics K-Means algorithm (CUDA). The numerical tests showed that the efficiency of GPU-based K-Means might be up to 40 times quicker than that of CPU-based K-Means[30].

Using the K-Means technique for OpenCL, they would demonstrate a well-parallelized environment that’s also dependable for analyzing or extracting massive datasets. They also recommend comparing the three most effective K-Means method deployments: The execution of the sequential Lloyd-Forgy approach, an OpenMP-based parallel version aimed at the CPU, and ultimately the most complex solutions that used the OpenCL system Efficiency is usually measured using various data sizes. These findings indicate an excellent addition to contrasting the GPU-based parallelism technique to the CPU-based serial approach for large datasets using OpenCL. Conversely, an OpenMP solution is the ideal option for tiny data sets[31].

The enhanced K-Means grouping optimization technique just on the Hadoop cluster was developed to address the issues with the classic K-Means segmentation
method in big data processing, including speed and selection of the center. That technique maximizes the selection through one core using the K-Means algorithm, computes the canopy technique [32] and similarity measure, and employs a distributed processing framework to scale the method in parallel. Can adapt here to the processing of massive data. The findings of the study demonstrate that the enhanced K-Means grouping optimization technique here on the Hadoop platform offers greater clustering effectiveness along with a substantial increase in speed or stability while computing a vast amount of data[32].

A K-Means technique analyzes the visual data in a Written programming environment. At the same time, deep convolution networks are built to classify both the actual and the transformed photos. The previous empirical results show that processing an image using the K-Means method takes 20 to 30 seconds less than processing any image using a neural network constructed using convolution. It really can significantly increase how quickly images are processed[33]. This parallel K-Means method solution using FPGA can accelerate the system’s computer performance and enable practical examples. Unlike most systems that the literature suggests, this approach lacks sequential phases, a computing performance constraining element, and even parallel versions. These metrics—throughput (or processing time) and FPGA area occupancy (or hardware resources)—are studied for various factors, with results exceeding 53 million datasets handled per second. Moreover, measurements of the current technology are shown, with speedups above a partially serial approach of much more than 15573x[34].

Highly enhanced MapReduce-based asymmetric anonymity K-Means technique. This suggested technique chooses the starting center point using Canopy but implements unequal privacy and security using the Laplace technique. The simulation demonstrates that the suggested application's clustering outcomes beat the
conventional DP K-Means in terms of usability or closure speed[35]. They propose a MapReduce-based methodology for such a K-Means similarity measure for window frame versions of Windows[36].

The canopy method [25], the highest and lowest distances, and just a K-mean two structures are suggested. The canopy method creates the k-value, so avoid artificially altering it. To lessen the effect of outliers on clustered findings, the weighted concentration approach was used to choose the grouping center set. In order to prevent the number of clusters from settling into a local optimum, the center point is again chosen using the highest and lowest separation. On Spark, the method is multithreading. Last but not least, the experimental findings from the UCI dataset demonstrate that the revised K-Means algorithm improves the grouping efficiency and speeds up the algorithm's median repetition times. According to research observations, the proposed method may significantly boost its parallel computing power and effectiveness[37].

By adapting the link prediction K-Means technique to be expandable and capable of operating on big data, one such work seeks to address this issue. We make use of the system’s distributed processing area, HPCC Technologies. The described K-Means technique uses a hybrid parallelism technique to achieve scalability in parallel K-Means. With us, the method can help scholars and machine learning experts who train thousands of algorithms daily to save significant time. The effectiveness of the scalable parallel K-Means technique is assessed using datasets and clusters of various sizes, and the findings demonstrate its considerable scalability[38].

The purpose of this paper is to show a hybrid method for optimizing the gene transcription site method using distributed as well as parallel computing capacity. The findings show a success rate for accurate, precise, sensibility, particular, and
corrected exactness of 91.15, 39.83, 89.11, and 88.93, but also 89.02 these are in percentages for such a Homo sapiens collection. This is a speedup of up to 23 publicizing the pecking order theory. This same precision, sensibility, and sensitivity, but also modified accuracy for such a Drosophila melanogaster database, had been 95.22 percent, 43.1 percent, 90.83 percent, 90.47 percent, and 90.64 percentage, sequentially. The performance boost was 18.33 times. Both groups of findings are significant. As a result, the answer suggested in this paper proved to be workable for the issue at hand[39].

The GPU-based K-Means classification technique is enhanced by using multiprocessor parallelism (ILP) at low demand. Once numerical results indicate the sum of the data within every cluster, for example, several separate instructions per thread are executed to accomplish ILP on the K-Means-based approach. Higher efficiency can be attained by adding more workloads to threads with reduced occupancy. An exercise with multiple data clustering demonstrates that the suggested approach can accelerate K-Means clustering significantly more quickly than other massively parallel approaches. K-Means application on the CPU, but also K-Means clustering[40].

With the help of such an approach, we now have a novel approach for designating dynamic groupings for the K-Means algorithm. However, we may also employ Parallel Libraries to process data simultaneously. The developed Gaining ability and the K-Means method are used in the analyses of that 2-D primary data. Thus, the findings confirm that now the Modified approach is even more successful and productive than K-Means in considerations of cluster creation, even as the Modified aspect designs the necessary clusters requested for parallelization, which enhances processor and cuts down on process time. K-Means and an updated approach are used to determine the outcomes for the various datasets in the library. The findings suggest a 20–50 percent
speedup when analyses are performed for the same sequence length.[41]

They examined the clustering application’s K-Means technique in depth. This study used convex hull and, indeed, the heel point answer to answer the initial two grouping points to modify the method by screening massive datasets. The complete parallelization process was made possible by the map-reduce programming paradigm. The effectiveness of the recommendation techniques was then evaluated in tests using the serial, parallel method, various cluster nodes, and various distance measurement techniques. The dependability and computational effectiveness of an enhanced parallel program are also improved with the growth of cluster nodes but also data size[42]. Researchers have already discussed a technique that employs Hadoop to perform the K-Means method while varying this same data collection and network nodes. We then compare parallel but also sequential processing while maintaining another parameter. The testing findings show that our algorithm can handle large datasets on the Hadoop platform effectively[43]. Using the Map - reduce paradigm, researchers developed and tested a parallel version of the K-Means technique and contrasted the results to sequential K-Means for clustering text datasets of various sizes. The outcome shows that recommended K-Means exceed sequential K-Means in terms of efficiency when clustering papers.[44].

Utilizing the most current development of GPUs to compute unified device Architecture, researchers have developed one such GPU that uses K-Means (CUDA)[45].This same hybrid K-Means classification technique combines the inner, globally optimized scanning capabilities of genetic algorithms with the ease and energy accuracy of a K-Means method. The instability of something like the K-Means clustering method has been successfully solved by creating a self-adaptive fitness function, multi-point crossover system of genes, and parallel transformation strategy, defecting that such an approach readily falls into an optimal local state. Lastly,
comparative studies using three large UCI datasets are conducted to demonstrate the viability and efficiency of this program.[46].

This study aims to create a parallel clustering algorithm to cluster vast data. It was proposed to enhance the K-Means technique by employing a novel heuristic technique to generate the best initial centroids but by applying MapReduce to big data sets. When contrasted with similar techniques already in use, the suggested approach is correct[47]. One such article utilizes the Java Join and Fork Technique to evaluate this same Parallel Genetic K-Means method and the Modified Parallel K Means algorithm’s efficiency. The fork/join technique fixes multiprocessing execution’s flaws[48]. We found early centroids with PSO but instead improved groupings with K-Means. Hadoop is used to analyze big datasets quickly and in parallel[49]. The entire Unit Means classification method has been updated by us utilizing better starting centers. Researchers have put forth several computation techniques and contrasted the outcomes[50].
3 PARALLELIZATION STRATEGIES

The parallel method for data mining utilizes a parallel architecture, and for implementation purposes, a parallel algorithm is used. A standard parallel clustering procedure includes at least three phases. The first phase is Partition, where we divide the information into smaller datasets. In the second phase, Computing, we run the clustering method on each processor’s local dataset. It’s worth noting that each processor’s clustering method may differ.

The third and final phase is Integration, where we merge all the information obtained from each processor to get the overall result. Two data techniques are commonly used. The first technique involves finding reasons for the highly changeable components of the information set or locating and explaining outliers. This technique is primarily used in science applications. The second technique aims to comprehend the variances of the overwhelming bulk of the given dataset components without any concern for the extremes. This technique is typically used in business applications.

For first-class applications, Computing is necessary, as we are still determining how beneficial sampling from a vast dataset will be in implementations of the latter type. Thus, parallel Computing has a big future as a platform for data processing. However, it’s not yet apparent if this is the direction data mining will take. By looking at the applications in which the most extensive data centers are used for data gathering, we can understand how the technique is already employed.

The parallelization of data involves various techniques, with the first method
being data parallelism. The aim of this method is to distribute the data appropriately and divide it into processors, each of which computes the given data in parallel. Data parallelism is classified into two types: record-based and attribute-based. The second method is the parallelization of tasks. Each record has the same likelihood of performing a data mining strategy when redistributed, making equally distributing information achievable. Data partitioning based on records is commonly used in real-world parallel data mining applications. If the Partition characteristic is incorrect, the information connection breaks, and the processing quality decreases.

The second strategy is Task Parallel, which breaks the entire solution method into many sub-procedures and sends them to separate processors. This method includes two ways of achieving task parallelism. One uses a divide-and-conquer technique, assigning the work and the subtask to a designated processor. The other is based on a scheduler, which dynamically assigns data to available processors. The execution method involves utilizing many CPUs to perform the same job and several CPUs to complete a separate one. The one selected is determined based on the area we are implementing and the data structure. Regardless of the parallel method used, the load-balancing problem must be addressed. Unequal burden may arise if data is misallocated.

The third alternative is a hybrid method, which involves breaking down the data and transmitting it to each processing unit. Jobs are assigned to a designated processor based on the data properties assigned to each processing unit. After properly breaking down the information and transmitting it to each processing unit, tasks are assigned to a designated processor by the attributes assigned to each processing unit. The data generated by the chain is combined when each job executing on each processor is finished. Task and data parallel techniques are used in this situation, and data interchange occurs before and after job execution.
4 CONTRIBUTION OF RESEARCH TO K-MEANS, MINIBATCH K-MEANS AND FUZZY C-MEANS

This chapter discusses a study focused on improving the quality of the information in a country data set using K-Means, Minibatch K-Means, and Fuzzy C-Means methods. The study uses parallelization techniques to speed up the execution times of all methods and obtain more accurate results. With the help of the elbow method, the optimal number of clusters is perceived using this method.

The study discusses different techniques for parallelizing: sequential K-Means, Minibatch K-Means, and fuzzy C-Means, parallelization using MPI (Message Passing Interface), and a cloud-based method. These techniques are implemented from scratch in Python, a popular programming language for machine learning and data processing. The results of this study may be helpful for organizations interested in aiding impoverished countries, as it provides more accurate information about the country data set.

4.1 Sequential Process of K-Means

Calculating centroids and cluster formation is done progressively with sequential K-means. Here is a detailed guide and breakdown of how K-Means works.
Figure 4.1: Flowchart of the K-Means Algorithm.
Algorithm 1 K-Means Algorithm

1: $K$ is cluster
2: $N$ is data objects
3: $\text{clusters}[K]$ cluster centers in array
4: $\text{objects}[N]$ data objects in array
5: $\text{membership}[N]$ objects memberships in array
6: Set threshold $\Theta$
7: repeat
8: \hspace{1em} $G \leftarrow 0$
9: \hspace{1em} for $a = 0$ to $N - 1$ do
10: \hspace{2em} for $b = 0$ to $K - 1$ do
11: \hspace{3em} $\text{distances} \leftarrow \text{objects}[a] - \text{clusters}[b]$
12: \hspace{3em} if $\text{distances} < d_{\text{min}}$ then
13: \hspace{4em} $d_{\text{min}} \leftarrow \text{distances}$
14: \hspace{4em} $n \leftarrow b$
15: \hspace{3em} end if
16: \hspace{2em} end for
17: \hspace{1em} if $\text{membership}[a] \neq n$ then
18: \hspace{2em} $G \leftarrow G + 1$
19: \hspace{2em} $\text{membership}[a] \leftarrow n$
20: \hspace{1em} end if
21: \hspace{1em} $\text{newcluster}[n] \leftarrow \text{newcluster}[n] + \text{objects}[a]$
22: \hspace{1em} $\text{newclustersize}[n] \leftarrow \text{newclustersize}[n] + 1$
23: \hspace{1em} end for
24: \hspace{1em} for $b = 0$ to $K - 1$ do
25: \hspace{2em} $\text{clusters}[b][*] \leftarrow \text{newcluster}[b][*]/\text{newclustersize}[b]$
26: \hspace{2em} $\text{newcluster}[b][*] \leftarrow 0$
27: \hspace{2em} $\text{newclustersize}[b] \leftarrow 0$
28: \hspace{1em} end for
29: \hspace{1em} until $G/N \leq \Theta$

The code is a Python implementation of the K-Means clustering technique. The objective of the method is to divide a collection of $N$ objects into $K$ clusters according to how identical they are, where $K$ is the number of clusters.

The K-Means algorithm uses this code to cluster data. It initializes the cluster centers and places data objects into clusters. Then, it refines the cluster centers and data object membership iteratively until reaching a threshold value that terminates the algorithm. To manage data, it uses variables to handle cluster centers, data
objects, and membership information. Cluster centers are an array of size K, while
data objects and membership data are arrays of size N.

The code computes the distances between objects and each cluster center during
each iteration, assigning objects to the nearest cluster based on these calculations.
If the object is assigned to a new cluster, a variable counter increments to reflect
changes in its membership data. The new cluster center updates by summing its
data object values and dividing by the cluster size.

Finally, the algorithm checks if the termination threshold is met. If not, it
updates the cluster centers and repeats the process.

4.2 Minibatch K-Means

Algorithm 2 Minibatch K-Means

```
1: Begin
2: Initialize: k, mini-batch size b, iterations k, data collection X
3: Set: Every s ∈ C with an x drawn at random to X
4: a ← 0
5: for j ← 1 to k do
6:   M ← b examples picked randomly from X
7:   for x ∈ M do
8:     d[x] ← f(C, x)
9:   end for
10: for x ∈ M do
11:   s ← d[x]
12:   a[s] ← a[s] + 1
13:   n ← 1/a[s]
14:   s ← (1 - n)s + nx
15: end for
16: end for
17: Stop
```
Here is an explanation of the Algorithm 2 code, step by step:

For each point \( x \) in dataset \( X \), a centroid \( s \) is initialized randomly from the set of possible centroids \( C \). The algorithm then runs for \( k \) iterations, each with the following steps.

- A mini-batch \( M \) of \( b \) examples is randomly sampled from the dataset \( X \).
- For each example \( x \) in the mini-batch \( M \), the distance between \( x \) and each centroid in \( C \) is computed using the function \( f \).
- For each example \( x \) in the mini-batch \( M \), the closest centroid \( s \) is identified based on the computed distances, and the count of examples assigned to that centroid \( a[s] \) is incremented.
- For each example \( x \) in the mini-batch \( M \), the position of the assigned centroid \( s \) is updated using a weighted average of its previous position and the new example \( x \). The weight given to the previous position is determined by the number of examples already assigned to that centroid, i.e., \( s \leftarrow (1-n)s+nx \), where \( n = 1/a[s] \).
- At the end of \( k \) iterations, the final positions of the centroids are returned.

The main idea behind this code is to iteratively improve the positions of the centroids by adjusting their position based on the examples in each mini-batch. By randomly sampling the examples and updating the centroids incrementally, the algorithm can scale to large datasets while producing high-quality clustering results.
### 4.3 Fuzzy C-Means

**Algorithm 3 Fuzzy C-Means Algorithm**

1. Initialize $U = [u_{ij}]$ matrix, $U^{(0)}$
2. At $k$-step: calculate the center vectors $C^{(k)} = [c_j]$ with $U^{(K)}$
3. $C_i = \frac{\sum_{j=1}^{n} u_{ij} x_j}{\sum_{j=1}^{n} u_{ij}^m}$
4. Update $U(k), U(k+1)$
5. $u_{ij} = \frac{1}{\sum_{k=1}^{n} \left( \frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{2/(m-1)}}$
6. $\|U^{(k+1)} - U^{(k)}\| < \epsilon$ then STOP
7. Else
8. Return to Step 2;

Here is an explanation of the Algorithm 3 code, step by step:

- To perform Fuzzy C-Means clustering, we first initialize the membership matrix, $U$, by assigning random values to it or by setting it to a predefined initial value.

- At each iteration, denoted by the $k$-step, we calculate the center vectors of the clusters, representing the centroids of the clusters. The center vectors are calculated using the membership matrix $U$ from the previous iteration.

- The center vectors are calculated using the membership matrix $U$ from the previous iteration. Specifically, we calculate the center vectors, $C_i$, by taking a weighted average of the data points, $x_j$, according to their membership values, sloppy $u_{ij}$, $ij$ should be subscripts of $u$.

- A higher membership value for a data point implies a more prominent weight for that data point. Once we have calculated the center vectors, we update the membership matrix for the next iteration by recalculating the membership values, $u_{ij}$, for each data point, $x_i$, based on their distance from the center vectors, $c_j$.  

---

43
• The new membership value, \( u_{ij} \), depends on the sum of the distances from the data point \( x_i \) to all the center vectors, \( c_j \), raised to the power of \( \frac{2}{m-1} \), where \( m \) is a fuzziness parameter that controls the level of fuzziness or overlapping among clusters.

• We then check if the difference between the membership matrices at the \( k \)-step and the \( (k+1) \)-step is lower than a certain threshold, epsilon (\( \varepsilon \)). If the difference is smaller than epsilon, the algorithm terminates and returns the final membership matrix and center vectors.

• Otherwise, the algorithm returns to step 2 to recalculate the center vectors and update the membership matrix for the next iteration. The algorithm repeats steps 2 to 7 until convergence, which implies that the difference between the membership matrices at two consecutive iterations is less than or equal to epsilon.

4.4 Parallel Techniques using Message Passing Interface

MPI is one of the best methods for implementing parallel computing. The apparent reason for that is because of the standard libraries it has. The motivation is to aim for exactness and efficacy. With the help of open MPI, we can parallelize the K-Means, Minibatch K-Means, and Fuzzy C-Means. MPI is supported in many languages. In this, the datasets were divided equally among many of the MPI processes, and Using OpenMP instructions, data labeling for each local data within the process was carried out.

Each attribute has its dimension in the result and is created in the space based on the number of attributes. The result generated using this methodology will be assimilated with the basic version of the technology to determine whether the
approach is superior and how much performance measurements, such as time for completion, are generated. This method was evaluated on the same dataset, termed “Country Data Set.” The information we will validate is based on the testing dataset, and we broke that complete information into train and test for training and testing the model. Here we used eighty for training and twenty for testing, i.e., 80:20.

Figure 4.2: A description of how to put the divide and combine plan into practice of Parallel clustering techniques using MPI.

To implement this, we use Jupyter Notebook. Where the ”divide and combine” scheme for parallel K-Means, Minibatch K-Means, and Fuzzy C-Means clustering using the Message Passing Interface (MPI) library, the process can be divided into several steps. Firstly, the data points should be divided into smaller subsets that different processes can independently process. Each process would then compute the centroid of a subset of data points using all algorithms until it converges to a stable value. After that, the partial centroids obtained from each process must be combined to obtain the final cluster of centroids. This can be achieved using the MPI reduce() function, which combines the partial centroids to ensure that all processes receive the same result Figure 4.2.

Finally, the results should be verified by comparing the final centroids with the expected values or by measuring the clustering quality using a performance metric
such as the Sum of Squared Errors (SSE). The divide and combine scheme in parallel clustering using MPI provides an efficient way to process large datasets by dividing them into smaller subsets and processing them in parallel, resulting in faster execution times and increased scalability.

4.4.1 Steps involved in Parallel Techniques using MPI

Figure 4.3: Flow of Parallel Techniques using MPI.
The algorithm starts by choosing the number of clusters, k, and giving each node one of the k cluster centers. It is crucial to remember that the clusters must be distributed evenly among the nodes, i.e., k/n, where n is the total number of nodes.

After assigning the cluster centers, each node actively computes the mean centers of the k/n groups, disseminates the centers to all nodes using MPI (Message Passing Interface), receives the cores from other nodes, computes the distance between each data point and its nearest collection. When the cluster assignations stop changing, it actively achieves convergence. Finally, the process is actively repeated until reaching convergence.

A crucial element of the clustering algorithms is convergence. It ensures the algorithms terminate once the cluster assignments have stabilized and no more adjustments are required. This enhances the clustering process’ effectiveness and precision. In conclusion, a clustering algorithm is a valuable tool for grouping data into groups, and convergence is essential to the algorithm’s precision and effectiveness.

### 4.4.2 MPI methodology consists of several stages

- We first determine the centroid based on the number of nodes.

- At this point, the master component sends data chunks based on centroid to various nodes. These broadcasts are made possible by comm = MPI.COMM WORLD. And communication bias.

- The means of each node is used to calculate centroids, which are then broadcast to other nodes.

- Data points are dispersed using communication scatter. This process is repeated until convergence occurs.
4.5 Amazon Sage Maker

Amazon SageMaker is a cloud-based machine learning platform that enables developers to build, train, and deploy machine learning models. It offers a range of tools for data labeling, training, experimentation, model deployment, and monitoring. Amazon SageMaker uses a pay-per-use pricing model, which makes it more cost effective for small-scale projects and large-scale enterprises.

4.5.1 Working of parallel clustering techniques using MPI in AWS

Parallel clustering techniques using MPI in AWS SageMaker work by distributing the clustering algorithms across multiple compute nodes using the Message Passing Interface (MPI) protocol.

The input data is divided into various subsets. Each subset is given its own compute node when a training job is started in SageMaker using the clustering algorithms and MPI. Then, using MPI, the clustering algorithms are applied parallel to each subset to process the data across numerous nodes.

During the parallel processing, each computed node calculates its own local centroids and assigns data points to the nearest centroid. Then, the intermediate results are merged to obtain the final clustering solution.

The results are typically stored in an S3 bucket, which can be accessed by other AWS services or downloaded for further analysis. Using SageMaker for parallel clustering with MPI allows for the efficient processing of large datasets and significantly reduces the time required for clustering tasks.

Overall, parallel clustering using MPI in AWS SageMaker enables clustering in
the context of distributed computing of big datasets and provides a scalable and efficient way to process data using multiple compute nodes.

A managed service machine learning (ML) Amazon’s Elastic Compute Cloud (Amazon EC2) notebook example, which continues to run Jupyter Software, is an Amazon Sage Maker notebook example. This same notebook example is used to build and manage Jupyter notebooks for information extraction, training, and deploying machine learning models.

To begin the analysis, an Exploratory Data Analysis (EDA) of the chosen datasets will be conducted, followed by the training and creation of a model using AWS Sagemaker’s built-in version of clustering. The resulting recommendation engine can then be deployed using an AWS Sagemaker inference endpoint and integrated into an application via an API. These steps were performed within an AWS Sagemaker-hosted Jupyter Notebook instance running on an ml.t2.xlarge instance.
4.5.2 Steps to generate a Sage Maker notebook example

- Go to https://console.aws.amazon.com/sagemaker/ to access an Amazon Sage Maker console.

- Select Notebook cases, then start creating notebook examples.

- I will provide additional data just on the Generate Notebook example site (when the ground is not noted, start leaving the default settings in Sage Maker).

- Inside the Notebook example input box, enter an identity for one’s notebook example. Select item ml.t2.medium as the Notebook Test case. It is the cheapest example form supported by notebook cases and is adequate for this workout. If the ml.t2.medium applies differently and is not accessible in one existing AWS Environment, select ml.t3.medium.

- Select a console category to generate the notebook example in Console Identification. This console form chooses the operating system and the Jupyter edition in which one’s notebook instance is formed—View Amazon Linux 2 vs. Amazon Linux notebook cases for more data on application framework identification kinds. View Jupyter version control for more data.

- Add a new position for IAM, then start creating a role. The above IAM position is granted access to every S3 bucket with the word Sage maker. Such authorizations are obtained via the Amazon Sage Maker Full Access strategy, which Sage Maker associates with the position.

- Sage Maker starts an ML compute example in the above scenario, a notebooks example in a matter of seconds, and connects a 5 GB Amazon EBS storage capacity to it. This same netbook instance includes a Jupyter notebook domain.
controller that has been preselected, Sage Maker, AWS SDK library services, and a set of Anaconda library services.

- Generate a Notebook Example for further data on generating a Sage Maker netbooks example.

### 4.6 Implementation

![Implementation Process Diagram]

Figure 4.5: Implementation Process.

#### 4.6.1 Data Collection

We utilized Python programming language and machine learning techniques to implement algorithms on the KAGGLE country dataset [51], which consists of nine key factors. From those nine key factors, we use only four key factors to predict what organizations can use to determine financial assistance to nations. These nine attributes include Child Mortality, Exports, Health, Imports, Income, GDP, Inflation,
Life Expectancy, and Total Fertility.

To perform the analysis, we used an Apple M1 processor with a memory of 16GB. Our primary objective was to use these algorithms to identify countries that may require financial aid based on their performance in these four factors.

4.6.2 Data Pre-Processing

First, the code sorts the DataFrame data by the values in the gdpp column in ascending order using the sort values function and then resets the index of the sorted DataFrame using the reset index function with the argument drop=True. This creates a new DataFrame with the same data as data but with the rows sorted by GDP per capital in ascending order and a new index.

Then, the code creates a scatter plot using the plot scatter function with the x values as the index of the sorted DataFrame and the y values as the gdpp column of the sorted DataFrame. The plot xlabel, plot ylabel, and plot title functions are used to add axis labels and a title to the plot, and plot show is used to display the plot.

Next, the code computes the thirds of the 'gdpp' column using the quantile() function with the argument [0, 0.33, 0.67, 1] to get the lower quartile (0-33), middle quartile (33-67), and upper quartile (67-100) values. After that, the code defines the category labels as ['Low', 'Medium', 'High'].

Then, the code adds a new column to the DataFrame called GDP Category using the pd cut function with the arguments bins=quantiles and labels=categories. This function creates categories for the gdpp values based on the quantiles computed earlier and assigns the category label to each gdpp value in the new GDP Category column.
Finally, the code drops the columns health, life expec, total fer, and 'child mort from the DataFrame using the drop function with the arguments columns, inplace=True, and axis=1. This removes the specified columns from the DataFrame, without creating a new DataFrame.

In summary, this code performs data cleaning and visualization tasks on a DataFrame that contains information about various countries. It sorts the DataFrame by GDP per capital, creates a scatter plot of the GDP per capital values, categorizes the GDP per capital values into low, medium, and high categories based on quartiles, and drops some columns from the DataFrame that are not needed for further analysis.

4.6.3 Data Transformation

Since data transformation entails transforming raw data into a similar and standardized style, it is essential to clustering. Data transformation’s primary goal is to reduce the influence of different scales, ranges, and skewness in the variables because these factors can significantly affect the results of clustering. Normalization, standardization, PCA transformation, and log transformation are a few examples of the data transformation methods frequently used in clustering. The best data conversion technique must be chosen depending on the issue domain and the data’s nature. If the appropriate data transformation methods are applied, clustering may yield precise and insightful results.

4.6.4 Implementation of Clustering Algorithms

Sequential K-Means

The code provided reads a processed data file containing information about different countries. It initializes an empty list called sse to store the sum of squared
errors (SSE) for different values of k. Then it iterates through the range of k values from 2 to 6. A K-Means object is initialized with the current value of k, and the algorithm is fitted to the dataset. The SSE for the fitted model is then appended to the sse list.

The next block of code selects the input column for the K-Means technique and plots the SSE for different values of k. The plot shows that the optimal number of clusters is likely 4, as the SSE begins to level off at that point.

The code initializes a K-Means object with k=4 and fits it to the dataset. The appropriate prediction method is then used to predict the cluster assignments for each data point. These predicted cluster assignments are then added as a new column to the DataFrame.

Finally, the code prints out the centroid coordinates for each cluster.

In summary, the K-Means technique clusters similar data points into k clusters. The code provided reads a file of processed data, calculates the SSE for different values of k, and selects the optimal number of clusters based on the plot. It then predicts cluster assignments for each data point and adds them to the DataFrame. Finally, it prints the centroid coordinates for each cluster.

**Sequential Minibatch K-Means**

This code performs clustering on a dataset using the Mini Batch K-Means algorithm. The dataset is loaded from a CSV file, and the columns for clustering are specified. The range of k values to try is defined, and an empty list is initialized to store the sum of squared errors (sse).

A loop is then executed over each k value in the defined range. A Mini Batch
K-Means model is initialized with the current k value and fitted to the dataset in each iteration. The inertia attribute of the model is then appended to the sse list. This indicates how well the data is being clustered, with a lower sum of squared errors indicating a better fit.

Once the loop completes, the results are plotted as a line graph with k on the x-axis and the sum of squared errors on the y-axis. This plot can be used to identify the optimal number of clusters. This code sets the number of clusters to 4, and the model is fitted to the dataset using the prediction method.

The predicted clusters and the original dataset are then plotted as a scatter plot, with each point colored according to its cluster assignment. The cluster centers are also plotted as red circles. The resulting plot shows how the data has been partitioned into distinct clusters based on the specified features.

**Sequential Fuzzy C-Means**

This code performs fuzzy c-means clustering on a dataset of country data. First, the data is loaded from a CSV file, and non-numeric columns are removed. The remaining numeric columns are then normalized using StandardScaler.

The number of clusters is set to 4, and the fuzzy exponent parameter (m) and the maximum number of iterations are also defined. The fuzzy c-means clustering algorithm is then initialized with the normalized data, number of clusters, m, and maxiter values.

The algorithm is run, and the predicted cluster membership for each data point is obtained by taking the argmax of the u matrix. This assigns each data point to a cluster based on the degree of membership in each cluster.
The predicted clusters are added to the original dataframe, and the number of countries in each cluster is printed to the console. This indicates how well the data has been clustered and can help with further analysis or decision-making. Fuzzy c-means clustering is functional when data points can belong to multiple clusters simultaneously, allowing for more nuanced clustering than traditional k-means clustering.

Parallel K-Means Minibatch K-Means and Fuzzy C-Means Using MPI

This code is an implementation of the K-means clustering algorithm using MPI (Message Passing Interface) for parallelization. The code is divided into two parts: the master part (rank 0) and the worker part (other ranks).

In the master part, the code reads a CSV file containing data and initializes the number of clusters (num clusters) and other necessary variables. The data is divided into chunks, and each chunk is sent to a worker using MPI’s scatter operation. The initial centroids are also broadcasted to all workers.

In the worker part, each worker receives its chunk of data and calculates the distance between each data point and the centroids. It assigns each data point to the cluster with the closest centroid. It then counts the number of samples in each cluster and sends the count to the master using MPI’s gather operation. The worker also receives the updated centroids from the master using MPI’s allreduce operation.

This process of calculating distances, updating clusters, and centroids is repeated until the centroids no longer change. The final cluster assignments from all workers are gathered at the master, and the adjusted Rand score is calculated to compare the results with the labels obtained from running KMeans from scikit-learn (kmeans).
Finally, the code visualizes the clusters and centroids using matplotlib. It also performs some analysis, such as printing the frequency of clusters for different GDP categories, replacing cluster labels to match the original labels, computing a confusion matrix, and printing classification metrics.

Overall, this code demonstrates parallelizing the K-means algorithm using MPI and compares the results with scikit-learn’s KMeans implementation. It also provides visualizations and analysis of the clustering results. We will replace the KMeans function with MiniBatchKMeans and FuzzyCMeans and run the same code.

**Parallel K-Means, Minibatch K-Means, and Fuzzy C-Means using MPI in AWS Sage Maker**

As discussed above, we use the same code to run in AWS SageMaker, which employs parallel processing with MPI and all algorithms to cluster data.

### 4.6.5 Pattern Information

In Clustering, data points are grouped according to how alike they are using pattern information. The method finds patterns in the data by repeatedly assigning each data point to the closest centroid and adjusting the centroid by averaging all the data points given to it. Up until the centroids stop moving or the maximum number of repetitions has been achieved, this procedure is continued. The ensuing clusters show collections of data points with comparable patterns or traits. By comparing new data points to existing clusters, these clusters may be utilized to predict future data points and understand the underlying patterns within the data.
4.6.6 Comparison of various Techniques

We will compare all three methods of execution times, including in terms of execution speed, accuracy, precision, recall, and F1 slope and support.

4.7 Setup of Exploration

We tested and implemented all algorithms using Jupyter Notebook on my computer, and we also utilized Amazon SageMaker Jupyter Notebook for cloud services. We used a computer with an Apple M1 processor and 16 GB of RAM to run our algorithms during testing. To supplement our methodology, we used a KAGGLE dataset that contained 167 rows and ten columns, totaling 13.2 kb.

We utilized machine learning techniques and various Python libraries to implement our methodology. Our approach involved thorough data analysis and preprocessing, then implementing and testing various algorithms on the cleansed data. We then analyzed the results to determine the best approach for accurately predicting financial aid requirements for different nations.
5 Evaluation Matrices

Evaluation measures are used in machine learning to gauge a model’s effectiveness. These measures include accuracy, precision, recall, F1 score, and area under the curve (AUC). The sort of challenge getting resolved and the model’s goals influence the choice of evaluation metrics. Metrics for evaluation are crucial for determining a model’s efficiency and pinpointing areas that need development.

5.1 Confusion Matrix

A confusion matrix is a table that compares the actual and predicted values of a collection of data to assess the effectiveness of a classifying model. It is a matrix that shows how many categorization problems have true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) for each class.

Table 5.1: Confusion matrix

<table>
<thead>
<tr>
<th>Actual Positive</th>
<th>Predicted Positive</th>
<th>Predicted Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive (TP)</td>
<td>False Negative (FN)</td>
<td></td>
</tr>
<tr>
<td>False Positive (FP)</td>
<td>True Negative (TN)</td>
<td></td>
</tr>
</tbody>
</table>
The Table 5.1 the columns in the matrix show the classes’ predicted values, while the rows show the classes’ actual values. The model was successful when the model correctly anticipated a positive class, as shown by the number of true positives (TP), as opposed to the number of false positives (FP). When the model correctly forecasts a negative class, it is indicated by the number of true negatives (TN). At the same time, incorrect predictions are shown by the number of false negatives (FN).

The confusion matrix might determine a categorization model’s precision, recall, F1 score, support, accuracy, and other efficiency indicators.

5.2 Precision

Precision is the ratio of true positives to all algorithmically generated positive forecasts. It is calculated as true positives split by the sum of true positives and false positives.

\[
\text{Precision} = \frac{TP}{TP + FP}
\]

5.3 Recall

Recall measures the percentage of actual positive examples in the dataset that are genuine positives. One can calculate it by dividing the sum of true positives and false negatives by the quantity of true positives.

\[
\text{Recall} = \frac{TP}{TP + FN}
\]

5.4 F1 score

The harmonic mean of recall and precision is the F1 number. It is a means of balancing the compromise among recall and accuracy. It is determined by dividing
the sum of precision and memory by two times the product of precision and recall.

\[ F1 \text{ Score} = \frac{2 \times (P \times R)}{(P + R)} \]

Where \( P = \text{Precision} \)
\[ R = \text{Recall} \]

### 5.5 Support

Support shows how many examples there are in the dataset for each class. It is helpful to comprehend how the dataset’s classes are distributed and to assess how well the model performs for each class.

\[ \text{Support} = TP + FN \]

### 5.6 Accuracy

One of the evaluation measures that can be determined from a confusion matrix is accuracy. It displays the percentage of instances in the dataset that were properly classified.

\[ \text{Accuracy} = \frac{(TP + TN)}{(TP + FP + FN + TN)} \]
6 RESULTS AND CONCLUSION

6.1 Elbow method

The Elbow method is a practical approach for determining the optimal no of clusters in a model. Applying this method makes it possible to identify the most appropriate number of distinct clusters for a given dataset Figure 6.1.

![Elbow Curve for Clustering](image)

Figure 6.1: Distortion Score Elbow Curve for Clustering.

We can create a new model with the optimal number of clusters identified, which is 4, and then examine the cluster assignments for each observation. Essentially, we would assign each data point to one of the 4 clusters established on their characteristics or similarities. This process can provide valuable insights into patterns and relationships within the data, potentially aiding in making informed decisions or predictions and the graphs below.
1. Sequential K-Means

Figure 6.2: Clusters formation for Sequential K-Means.

2. Parallel K-Means using MPI

Figure 6.3: Clusters formation for Parallel K-Means using MPI.
3. Parallel K-Means using MPI in Amazon Sage maker

Figure 6.4: Clusters formation for Parallel K-Means using MPI in Amazon Sage maker.

4. Sequential Minibatch K-Means

Figure 6.5: Clusters formation for Sequential Minibatch K-Means.
5. Parallel Minibatch K-Means using MPI

Figure 6.6: Clusters formation for Parallel Minibatch K-Means using MPI.

6. Parallel Minibatch K-Means using MPI in Amazon Sage Maker

Figure 6.7: Clusters formation for Parallel Minibatch K-Means using MPI in Amazon Sage Maker.
7. Sequential Fuzzy C-Means

Figure 6.8: Clusters formation for Sequential Fuzzy C-Means.

8. Parallel Fuzzy C-Means using MPI

Figure 6.9: Clusters formation for Parallel Fuzzy C-Means using MPI.
9. Parallel Fuzzy C-Means using MPI in Amazon Sage Maker

Figure 6.10: Clusters formation for Parallel Fuzzy C-Means using MPI in Amazon Sage Maker.

The confusion matrix is a tool that helps to summarize the performance of a model by displaying the number of correct and incorrect predictions made for each class in the dataset. It consists of three columns and three rows, corresponding to the three classes in our data. The true values are represented on the y-axis while the predicted values are represented on the x-axis. Using the values obtained from the confusion matrix, precision, recall and F1 scores can be calculated for the model. The confusion matrices for three different clustering algorithms; Sequential K-Means Figure 6.11, Sequential Minibatch K-Means Figure 6.12 and Sequential Fuzzy C-Means Figure 6.13.
Figure 6.11: Confusion matrix of Sequential K-Means.

Figure 6.12: Confusion matrix of Sequential Minibatch K-Means.
Table 6.1: Evaluation Matrix of Sequential Clustering

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
<th>precision</th>
<th>recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-Means</td>
<td>0.45</td>
<td>0.68</td>
<td>0.66</td>
<td>0.67</td>
</tr>
<tr>
<td>Minibatch K-Means</td>
<td>0.48</td>
<td>0.75</td>
<td>0.72</td>
<td>0.71</td>
</tr>
<tr>
<td>Fuzzy C-Means</td>
<td>0.46</td>
<td>0.69</td>
<td>0.65</td>
<td>0.63</td>
</tr>
</tbody>
</table>

The table shows that Minibatch K-Means performs the best out of the three algorithms in all four metrics. It has higher sequential accuracy, precision, recall, and F1-score than K-Means and Fuzzy C-Means. This indicates that Minibatch K-Means is the most effective algorithm in accurately grouping the data points into clusters.
Figure 6.14: Comparison of Sequential Clustering techniques.

Table 6.2: Evaluation Matrix of Parallel Clustering using MPI

<table>
<thead>
<tr>
<th>Parallel using MPI</th>
<th>Accuracy</th>
<th>precision</th>
<th>recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-Means</td>
<td>0.47</td>
<td>0.75</td>
<td>0.72</td>
<td>0.71</td>
</tr>
<tr>
<td>Minibatch K-Means</td>
<td>0.46</td>
<td>0.69</td>
<td>0.65</td>
<td>0.66</td>
</tr>
<tr>
<td>Fuzzy C-Means</td>
<td>0.45</td>
<td>0.67</td>
<td>0.66</td>
<td>0.67</td>
</tr>
</tbody>
</table>

The table shows that when implementing these clustering algorithms using MPI parallelism, K-Means performs slightly better than Mini-Batch K-Means and Fuzzy C-Means in terms of accuracy, precision, recall, and F1-score. It achieves higher values in most of the metrics, indicating better overall performance. However, as always, it’s important to consider the specific dataset and problem context when choosing the appropriate algorithm and parallelization strategy. Table 6.2. Also, we presented a graphical comparison representation of Figure 6.15.
Table 6.3: Evaluation Matrix of Parallel Clustering using MPI in Amazon Sage Maker

<table>
<thead>
<tr>
<th>Amazon Sage Maker</th>
<th>Accuracy</th>
<th>precision</th>
<th>recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-Means</td>
<td>0.48</td>
<td>0.74</td>
<td>0.71</td>
<td>0.70</td>
</tr>
<tr>
<td>Minibatch K-Means</td>
<td>0.45</td>
<td>0.68</td>
<td>0.66</td>
<td>0.67</td>
</tr>
<tr>
<td>Fuzzy C-Means</td>
<td>0.46</td>
<td>0.70</td>
<td>0.69</td>
<td>0.68</td>
</tr>
</tbody>
</table>

The table shows that all three algorithms when implemented on Amazon Sage Maker. K-Means has the highest accuracy, followed by Fuzzy C-Means and Minibatch K-Means. It achieves higher values in most of the metrics, indicating better overall performance Table 6.3. Also we presented graphical comparison representation Figure 6.16.
Finally, by comparing the performances of three clustering algorithms K-Means, Fuzzy C-Means, and Minibatch K-Means we can see that the sequential implementation using these algorithms outperforms the parallel implementation using MPI or Amazon Sage Maker. Among the three algorithms, Minibatch K-Means has shown better results compared to K-Means and Fuzzy C-Means, having the highest accuracy, recall, precision, and F1-score values in the sequential implementation.

The table displays the execution times of three clustering algorithms: K-Means, Mini-Batch K-Means, and Fuzzy C-Means, for both sequential and parallel techniques using MPI and AWS Sage Maker, measured in milliseconds. Significantly, the parallel implementations using AWS Sage Maker exhibited greater efficiency when compared to the parallel implementations using MPI and the sequential implementations for all three algorithms. Notably, Fuzzy C-Means demonstrated the shortest execution time for both parallel implementations, highlighting its potential suitability for clustered...
data analysis, and the implementation of Fuzzy C-Means in parallel using AWS Sage Maker may therefore provide the most effective means of optimizing algorithm performance and scalability Table 6.4.

Table 6.4: Comparing execution time of all techniques

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Execution Times (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential K-Means</td>
<td>2.853</td>
</tr>
<tr>
<td>Parallel K-Means using MPI</td>
<td>1.050</td>
</tr>
<tr>
<td>Parallel K-Means using AWS Sage Maker</td>
<td>0.828</td>
</tr>
<tr>
<td>Sequential Minibatch K-Means</td>
<td>2.310</td>
</tr>
<tr>
<td>Parallel Minibatch K-Means using MPI</td>
<td>1.215</td>
</tr>
<tr>
<td>Parallel Minibatch K-Means using AWS Sage maker</td>
<td>0.921</td>
</tr>
<tr>
<td>Sequential Fuzzy C-Means</td>
<td>1.285</td>
</tr>
<tr>
<td>Parallel Fuzzy C-Means using MPI</td>
<td>0.932</td>
</tr>
<tr>
<td>Parallel Fuzzy C-Means using AWS Sage maker</td>
<td>0.623</td>
</tr>
</tbody>
</table>

Overall Considering these findings, cloud computing platforms like AWS Sage Maker have the potential to offer substantial advantages for data processing in parallelized clusters. Also, you can see the graphical representation of Figure 6.17

![Comparison of Three Different Execution Times](image)

Figure 6.17: Comparison of Execution times in all techniques.
6.2 Discussion

The independent features are imports, exports, health, and inflation. Based on these four features, clusters are formed. We determine the model accuracy using the GDP feature from the dataset and the clusters. The GDP values are categorized as low, medium, and high. Based on the GDP category, we determine whether a country requires financial assistance or not. If the GDP is categorized as low, it signifies that the country needs help. For medium GDP, the country may or may not require assistance, and for high GDP, the country does not require help.

![Needed Help Per Country (World)](image)

Figure 6.18: Help needed per country (World).

From Figure 6.18, the green color indicates help is not required, the red color indicates help need, and the yellow might need help and shows the countries in need of assistance and those that may require assistance. As a result, an NGO or organizations can quickly analyze data and assist countries.

The utilization of parallel clustering techniques results in numerous advantages over sequential clustering approaches, including enhanced efficiency, flexibility, scalability, fault tolerance, resource utilization, and real-time capabilities. These
benefits make parallel clustering techniques the preferred option for dealing with large-scale datasets and time-sensitive clustering tasks.

The managed and user-friendly environment provided by Amazon SageMaker is ideal for developing and deploying machine learning models that include clustering algorithms. It offers scalability, managed infrastructure, integration with AWS services, flexibility, monitoring capabilities, and cost-effectiveness advantages that make it a leading choice over manual setup and management of parallel clustering techniques that utilize MPI.

6.3 Conclusion

Machine learning (ML) is a field devoted to understanding and building methods that let machines leverage data to enhance the performance and accuracy of different applications. A subset of machine learning is closely related to data clustering, which focuses on grouping data samples into various groups for data prediction, detection, identification, etc.

Unsupervised ML algorithms are the most effective methods for data clustering. This work investigates various approaches and identifies factors influencing the algorithm’s efficiency. Among them are the effects of the starting cluster on accuracy and the way the data is parallelized on efficiency. Finding the best approach based on the available data takes time and effort. We concentrated on country data and analyzed it using various parallel approaches and cloud computing. The analysis is based on four different country factors, and this analysis will assist various organizations that are prepared to assist the country.

The results demonstrate that implementing these models on a cloud platform significantly improves their performance in terms of execution time. By evaluating
accuracy scores and confusion matrix values, we were able to compare the effectiveness of the three models. The findings indicated that when utilizing the sequential process, Minibatch K-Means achieves better performance compared to K-Means and Fuzzy C-Means. However, in cases where the parallel process was used, K-Means produced better results than the other two models.

6.4 Future Work

Our future work will concentrate on analyzing data using MapReduce and Multiprocessing, as well as developing an application that receives data as input and recommends the most efficient method for speedy execution. We aim to enhance the accuracy, precision, recall, and F1-score of our results. Additionally, we will experiment with different clustering algorithms for cluster initialization and construct an algorithm to assess the dataset and determine the most appropriate parallelization technique.
7 Applications

7.1 Customer Segmentation

Grouping assists marketing teams in increasing their customer base by focusing on critical regions and segmenting customers based on purchasing history, preferences, or continuous monitoring. Businesses can use segmentation to select specific cluster nodes of clients for marketing tactics.

7.2 Document Classification

Group records into multiple classes based on labels, topics, and file content; this is a prevalent classification problem, but K-Means is an appropriate algorithm. This exact initial computation of the records is required to reflect each file as a vector and uses word frequency to recognize frequently used phrases that aid in text categorization. The manuscript vectors are therefore clustered to aid in identifying commonalities between file organizations.

7.3 Identifying crime hotspots

As for crime statistics accessible places within such a city, the classification of crime, this neighborhood of violent acts, and the relationship between them can provide performance understanding in and out of crime-prone regions inside a city or a geographical region.
7.4 Delivery store optimization

Use a mixture of k-means to determine the best amount of takeoff places and an evolutionary algorithm to fix the lorry pathway as a traveling salesman issue to maximize the procedure of perfect delivery utilizing lorry drone technology.

7.5 Ridesharing data analysis

The freely released cab coaster knowledge dataset contains valuable data about traffic, transit times, peak delivery locations, and more. Evaluating the above information can provide an understanding of other urban traffic conditions and aid in designing future metropolitan areas.

7.6 Analysis of call record details

The data collected by telecommunications companies throughout a user’s calls, texts, and online activity is called a cell phone location file. Once combined with segment customers, the above knowledge gives a better understanding of the client’s needs. The K-means hierarchical clustering can be utilized to analyze customer segments based on one’s utilization by minutes.

7.7 Detecting insurance fraud

Machine learning is essential in detecting fraud and has various uses in vehicles, universal health care, and health coverage fraud prevention. It is possible to separate initial allegations predicated on their closeness to groupings that suggest fraudulent trends using past historical information on false allegations. Because cybercrime may have a mega-dollar influence on a company, this same ability to spot theft is critical.
7.8 Criminals who use cyber-profiling

Cyber profiling collects information from both people and groups to discover basic and core information. Virtual lawbreaker profiles provide data to categorize criminal activities.

7.9 Fully automated clustering of IT notifications

Even though alert notifications may indicate operational problems, they must always be individually scanned to prioritize a big company’s process operations. IT confirms framework elements such as system, storage, or data system. The clustering algorithm might provide insight into the notify classifications, indicate repair times, and aid defect prediction.

7.10 Prediction for the stock market

Because of the most recent advancement in ML algorithms well over the last 20 years, these have attracted many academics’ and scientists’ networking attention. It has also resulted in identifying special software in several fields, including financial. Our suggested model uses ML algorithms and parallel computing to provide a new purchase and sale strategy for non-desk connected and multi-dimensional economic statistics of Corporation Limited that was obtained from a disintegrated acct using the upstox (API) Application, which extracts data at a standard computer(C) language extraction speed of 10 minutes. This suggested variant employs a K means ML strategy to model the accumulated inventory information and accurately anticipate the impending stock returns in conjunction with concurrent processing technology.

The evaluation of valuation data from the previous year’s stock market is presented in this research paper, along with an interpretation of the findings on
extensive training and the establishment of CUDA learning guidelines that consider the pressure of real-time trading. With the help of system learning methodologies, the computer’s overall performance has significantly improved, so this study uses parallel computing to speed up the process of creating the effects. When CUDA parallel computing generation is used instead of more traditional methods that use a single CPU, the overall output time is significantly reduced (CPU). This ultimate goal of acquiring and selling by accurately anticipating the stock prices in advance assisted in reducing computing time through a massive margin and, as a result, benefitted e-book revenues.

Based on three clusters as the predicted good strategy set of regulations, purchasers can decide whether to hold onto that stock, sell, buy some additional shares, or make an unbiased option. According to the neutrality option principle, if a person already has stock, he should keep it with him; if he doesn’t, he should not buy more. The stock price is collected every 10 minutes with the intention that now the current technique can decide what should be done, making it suitable for intraday trading. We just considered one stock, yet in the future, it will be able to assess behavioral patterns across many stocks or, indeed, forecast it for every stock traded just on the stock exchange. It can also advise a stock portfolio that relies on clustering-based unsupervised learning that results primarily in profit and minimizes trade danger.

Combining chart patterns’ findings with Twitter sentiment classification may improve the accuracy of the suggested model. Twitter sentiment analysis can be included to reduce errors. Web apps can be created for practical uses to improve the user experience. Changing the group count could create future studies on volatile stock behavior.
7.11 High dimensional text data

Throughout this study, a parallel K-means approach to handling thin, high-dimensional text data was presented (PKHT). The suggested technique provides an up to 11x shorter runtime by utilizing MPI (Message-passing Interface) with GPU (Graphical Processing Unit). The focus of the clustering algorithm is large-scale data clustering. There are generally two different groups for textual clustering algorithms. The second is focused on division clustering, while the first is focused on hierarchical clustering. The well-liked k-means grouping technique is utilized in this study. This information is separated into various classes based on the various traits of the text data.

Because Makowski Length cannot be utilized to create a cluster for high-dimensional data, TF-IDF and cosine are used to achieve this. Utilizing GPU, you may perform parallel matrix computation. A single GPU cannot match the demands of data growth.

Hence, multi-GPU processing is made possible with the use of MPI technology. The GPU implementation in just this study is made 11 possible by CUDA universal parallel computing framework. According to experiment results, the suggested has a lower time overhead on large amounts of data and greater precision. Actual test findings demonstrate that the technique suggested in this research also performs well for grouping high-dimensional sparse text data but also dramatically increases operating speed by utilizing the CUDA architecture and MPI.
REFERENCES


Data Preprocessing Python Code

Data Set: https://www.kaggle.com/datasets/rohan0301/unsupervised-learning-on-country-data

```python
# !pip3 install mpi4py
import os
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
# import seaborn as sns
# from mpi4py import MPI
from sklearn.cluster import KMeans
# import plotly.express as px
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import statistics
from sklearn.cluster import Birch
```
import time

from collections import Counter
from sklearn.cluster import DBSCAN

# import plotly.express as px
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from random import sample

# import plotly.io as pio
pip install -U kaleido

# import kaleido
import math
import numpy as np
import pandas as pd
import csv
import time
import numpy as np
import collections

# from mpi4py import MPI
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score

# pio.renderers
# import warnings
# from yellowbrick.cluster import KElbowVisualizer
# warnings.filterwarnings('ignore')
# pio.renderers.default = "svg"
# from plotly.offline import plot, iplot, init_notebook_mode
# import plotly.graph_objs as go
#init_notebook_mode(connected=True)

df=pd.read_csv('Country-data.csv')

#Display the data of country data
df

df = df.sort_values(by='gdpp').reset_index(drop=True)

# Display the sorted DataFrame
print(df)

# Create a scatter plot of the "gdpp" column
plt.scatter(df.index, df['gdpp'])

# Add axis labels and a title
plt.xlabel('Country Index (sorted by GDP per capita)')
plt.ylabel('GDP per capita')
plt.title('GDP per Capita by Country')

# Show the plot
plt.show()

# Compute the quantiles of the "gdpp" column
quantiles = df['gdpp'].quantile([0, 0.33, 0.67, 1])

categories = ['Low', 'Medium', 'High']

# Add a new column to the DataFrame with the category labels
df['GDP_Category'] = pd.cut(df['gdpp'], bins=quantiles, labels=categories)

# Display the updated DataFrame
print(df)
```python
# Display the data after sorting of gdp

df

df['GDP_Category'][0] = df['GDP_Category'][1]

# Data we are dropping from the Data

columns = ['health', 'life_expec', 'total_fer', 'child_mort']

df.drop(columns, inplace=True, axis=1)

# Display the data after drop

# Final data set after sorting gdp and dropping data

df.to_csv('processed-country-data.csv', index=False)
```
B Appendix

Sequential K-Means and Minibatch K-Means Python Code

Note: - For Sequential Minibatch K-Means, also we use the same code. We use the MiniBatchKMeans function instead of the KMeans function.

```python
# !pip3 install mpi4py
import os
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
# import seaborn as sns
# from mpi4py import MPI
from sklearn.cluster import KMeans
# import plotly.express as px
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import statistics
from sklearn.cluster import Birch
import time
```
from collections import Counter
from sklearn.cluster import DBSCAN
# import plotly.express as px
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from random import sample
# import plotly.io as pio
pip install -U kaleido
# import kaleido
import math
import numpy as np
import pandas as pd
import csv
import time
import numpy as np
import collections
# from mpi4py import MPI
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score
# pio.renderers
# import warnings
# from yellowbrick.cluster import KElbowVisualizer
# warnings.filterwarnings('ignore')
#pio.renderers.default = "svg"
#from plotly.offline import plot, iplot, init_notebook_mode
#import plotly.graph_objs as go
#init_notebook_mode(connected=True)
df=pd.read_csv('processed-country-data.csv')

#Display the data of country data

df

x_col = ['exports', 'imports', 'income', 'inflation']

#this is the K-Means Model
sse = []
k_rng = range(2,7)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[x_col])
    sse.append(km.inertia_)
plt.xlabel('K')
plt.ylabel('Sum of squared error')
#Will get graph for elbow method
plt.plot(k_rng,sse)

#based on the elbow method clusters
km = KMeans(n_clusters=4)
#we predict the data
y_predicted = km.fit_predict(df[x_col])
y_predicted

df['cluster']=y_predicted
df.head()

#clusters centers
km.cluster_centers_
#graphical representation of clusters

fig, frame1 = plt.subplots()
clusters = km.labels_

centers = km.cluster_centers_

plt.scatter(df.iloc[:, 0], df.iloc[:, 1], c=clusters, s=50, cmap="viridis")

plt.scatter(cen[:, 0], cen[:, 1], c="red", s=200, alpha=0.8)

frame1.axes.get_xaxis().set_ticks([])
frame1.axes.get_yaxis().set_ticks([])

plt.xlabel("Clusters")
plt.ylabel("Centers")

df['label'] = df['GDP_Category']

df10 = pd.DataFrame(df.loc[df['label'] == "Low"])

# print(df10)

frequency = df10.cluster.value_counts()

print(frequency)

df11 = pd.DataFrame(df.loc[df['label'] == "Medium"])

# print(df11)
frequency = df11.cluster.value_counts()
print(frequency)
df12 = pd.DataFrame(df.loc[df['label'] == "High"])
# print(df11)
frequency = df12.cluster.value_counts()
print(frequency)
df['predicted'] = df['cluster']

import os
os.makedirs('csv_cluster', exist_ok=True)
df.to_csv('csv_cluster/out.csv')
df0 = pd.DataFrame(df.loc[df['label'] == "Low"])
df0.to_csv('csv_cluster/low.csv')
df1 = pd.DataFrame(df.loc[df['label'] == "Medium"])
df1.to_csv('csv_cluster/medium.csv')
df2 = pd.DataFrame(df.loc[df['label'] == "High"])
df2.to_csv('csv_cluster/high.csv')

# we will replace the clusters to adjust
df['predicted'] = df['predicted'].replace([0], 2)
df['predicted'] = df['predicted'].replace([1], 5)
df['predicted'] = df['predicted'].replace([3], 1)
df['predicted'] = df['predicted'].replace([5], 0)

df['label'] = df['label'].map({'Low': 0, 'Medium': 1, 'High': 2})

# Display data
df

y_test = df.label
y_pred_class = df.predicted
dfz = df['label'].value_counts()
print(dfz)

#confusion MatrixDisplay
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred_class)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Low: 0', 'Medium: 1', 'High: 2'])
cmd.plot()

#Display the precision,recall,f1slope,support
from sklearn.metrics import classification_report
# Compute classification metrics based on the confusion matrix
print(classification_report(y_test, y_pred_class,
    target_names=['Low: 0', 'Medium: 1', 'High: 2']))

#labels for countries help needed or not
df['label'].loc[df['label'] == 2] = 'Help not needed'
df['label'].loc[df['label'] == 1] = 'Might need help'
df['label'].loc[df['label'] == 0] = 'Help needed'

import kaleido
```python
import plotly.express as px

fig = px.choropleth(df[['country', 'label']],

    locationmode='country names',
    locations='country',
    title='Needed Help Per Country (World)',
    color_discrete_sequence=['orange', 'red', 'green', 'black'],
    color=df['label'],
    #color_discrete_sequence=px.colors.sequential.Plasma
    #color_discrete_sequence=px.colors.diverging.Earth
    color_discrete_map={'Help needed': 'Red',
                        'Might need help': 'Yellow',
                        'Help not needed': 'Green'}

    fig.update_geos(fitbounds="locations", visible=True)

    fig.update_layout(coloraxis_colorbar=

    dict(title=df_labeled2['abel'].name))

    fig.update_layout(legend_title_text='Labels',
    legend_title_side='top', title_pad_l=260, title_y=0.86)

    fig.write_html("NeededHelpPerCountry(World)kmeans.html")

    fig.write_image("NeededHelpPerCountry(World)kmeans.png",
    scale=3)

    fig.show(engine='kaleido')

    #Display Execution time

    stop = time.time()

    print("Execution Time ", (stop-start))
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import skfuzzy as fuzz

import time
start = time.time()

# Load the country dataset
df = pd.read_csv('/content/sample_data/processed-country-data.csv')

# Remove non-numeric columns
numeric_cols = ['imports', 'exports', 'income', 'inflation']
data = df[numeric_cols]

# Normalize the data using StandardScaler
scaler = StandardScaler()
data_norm = scaler.fit_transform(data)

# Set the number of clusters
n_clusters = 4

# Set the fuzzy exponent parameter
```python
m = 2

# Set the maximum number of iterations
max_iter = 100

# Initialize the fuzzy c-means clustering algorithm
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(data_norm.T, n_clusters, m, error=0.005, maxiter=max_iter, init=None)

# Get the predicted cluster membership for each data point
cluster_membership = u.argmax(axis=0)

# Add the predicted clusters to the original dataframe
df['Cluster'] = cluster_membership

# Print the number of countries in each cluster
print(df['Cluster'].value_counts())

fig, frame1 = plt.subplots()
clusters = fuzz.cluster.cmeans.labels_
centers = fuzz.cluster.cmeans.cluster_centers_
plt.scatter(df.iloc[:, 0],
            df.iloc[:, 1],
c=clusters,
s=50,
cmap="viridis")
plt.scatter(centers[:, 0],
            centers[:, 1],
c="red",
s=200,
alpha=0.8)
frame1.axes.get_xaxis().set_ticks([])
```
frame1.axes.get_yaxis().set_ticks([])
plt.xlabel("Clusters")
plt.ylabel("Centers")

df['label'] = df['GDP_Category']
df['predicted'] = df['Cluster']
df

f['predicted'] = df['predicted'].replace([0], 2)
df['predicted'] = df['predicted'].replace([1], 5)
df['predicted'] = df['predicted'].replace([3], 1)
df['predicted'] = df['predicted'].replace([5], 0)

df['label'] = df['label'].map({'Low': 0, 'Medium': 1, 'High': 2})

y_test = df.label
y_pred_class = df.predicted

dfz = df['label'].value_counts()
print(dfz)

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred_class)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Low: 0', 'Medium: 1', 'High: 1'])
cmd.plot()
from sklearn.metrics import classification_report

# Compute classification metrics based on the confusion matrix
print(classification_report(y_test, y_pred_class, target_names=['Low: 0', 'Medium: 1', 'High: 2']))

stop = time.time()
print("Execution Time ", (stop-start))

Note: - For Parallel Minibatch K-Means and Fuzzy C-Means, also we use the same code. We use the MiniBatchKMeans and Fuzzy C-means function instead of the KMeans function.

When Using MPI in AWS Sage Maker, we use the same code for parallel K-means, Minibatch K-Means and Fuzzy C-Means.

apt-get install openmpi-bin openmpi-common openssh-client openssh-server libopenmpi-dev

pip install mpi4py

import math
import csv
import time
import numpy as np
import collections
from mpi4py import MPI
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

#global dimensions, num_clusters, num_points,dimensions,data,flag
num_clusters=4

#Divide data set to further scattering

def chunkIt(seq, num):
    avg = len(seq) / float(num)
    out = []
    last = 0.0
    while last < len(seq):
        out.append(seq[int(last):int(last + avg)])
        last += avg
    return out

#Count labels for recalculating means of centroids

def addCounter(counter1, counter2, datatype):
    for item in counter2:
        counter1[item] += counter2[item]
    return counter1

import pandas as pd

if rank==0:
    start_time = time.time()
    df = pd.read_csv('/content/sample_data/processed-country-data.csv')
    data = df[['exports', 'imports', 'income', 'inflation']].values
    kmeans = KMeans(n_clusters=num_clusters, random_state=0).fit(data)
fit(data).labels_

#Initialize centroids matrix
initial=[]
for i in range(num_clusters):
    initial.append(data[i])
initial=np.vstack(initial)
num_points = len(data)
#number of rows
dimensions = len(data[0])
#number of columns

#chunks = [ [] for _ in range(size) ]
#for i, chunk in enumerate(data):
#    chunks[i % size].append(chunk)
chunks=chunkIt(data,size)
#deviding data set on parts for further scattering
else:
    chunks = None
    initial = None
    data = None
    dimensions = None
    num_points = None
    cluster= None
    Q_clust= None
    num_clusters= None
    centroid=None
    kmeans= None
    start_time=None
start_time=comm.bcast(start_time,root=0)
data=comm.scatter(chunks, root=0)
num_clusters=comm.bcast(num_clusters,root=0)
initial=comm.bcast(initial, root = 0)
flag= True
while flag==True:  
     clusters=[]
     cluster=[]
     dist =np.zeros((len(data),len(initial)))
     for j in range(len(initial)):
       for i in range(len(data)):
         dist[i][j]=np.linalg.norm(initial[j]-data[i])
     for i in range (len(dist)):
       clusters.append(np.argmin(dist[i])+1)
     Q_clusts=collections.Counter(clusters)
     counterSumOp = MPI.Op.Create(addCounter, commute=True)
     totcounter = comm.allreduce(Q_clusts, op=counterSumOp)
     comm.Barrier()

     cluster=comm.gather(clusters, root=0)
     comm.Barrier()
     if rank==0:
       cluster=[item for sublist in cluster for item in sublist]
centroids = np.zeros((len(initial), len(initial[0])))

for k in range(1, num_clusters+1):
    indices = [i for i, j in enumerate(clusters) if j == k]
    centroids[k-1] = np.divide(np.sum([data[i] for i in indices], axis=0).astype(np.float), totcounter[k])

centroid = comm.allreduce(centroids, MPI.SUM)

comm.Barrier()

if np.all(centroid == initial):
    flag = False
    print("Execution time %s seconds" % (time.time() - start_time))
else:
    comm.Barrier()

if rank==0:
    print('adjusted_rand_score', adjusted_rand_score(kmeans, cluster))

kmeans
df['predicted'] = kmeans
df
centroid

import matplotlib.pyplot as plt
fig, frame1 = plt.subplots()
frame1.axes.get_xaxis().set_ticks([])
frame1.axes.get_yaxis().set_ticks([])
clusters = kmeans

centers = centroid

plt.scatter(df.iloc[:, 0],
            df.iloc[:, 1],
            c=clusters,
            s=50,
            cmap="viridis")

plt.scatter(centers[:, 0],
            centers[:, 1],
            c="red",
            s=200,
            alpha=0.8)

plt.xlabel("Data Index")
plt.ylabel("Import Values")

data10 = pd.DataFrame(df.loc[df['GDP_Category'] == "Low"])
# print(data10)
frequency = data10.predicted.value_counts()
print(frequency)
data11 = pd.DataFrame(df.loc[df['GDP_Category'] == "Medium"])
# print(df11)
frequency = data11.predicted.value_counts()
print(frequency)
data12 = pd.DataFrame(df.loc[df['GDP_Category'] == "High"])
# print(df11)
frequency = data12.predicted.value_counts()

print(frequency)

df['predicted'] = df['predicted'].replace([1], 4)
df['predicted'] = df['predicted'].replace([2], 1)
df['predicted'] = df['predicted'].replace([4,3], 2)

# data['predicted'] = data['predicted'].replace([5], 0)

df

df['label'] = df['GDP_Category'].map({'Low': 0, 'Medium': 1, 'High': 2})

df

y_test = df.label
y_pred_class = df.predicted

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred_class)
cmd = ConfusionMatrixDisplay(cm, display_labels=['Low: 0', 'Medium: 1', 'High: 2'])
cmd.plot()

from sklearn.metrics import classification_report

# Compute classification metrics based on the confusion matrix
print(classification_report(y_test, y_pred_class, target_names=['Low: 0', 'Medium: 1', 'High: 2']))