3116 – Lab Distributed Data Analytics – Group 2

Exercise Sheet 3

Yuvaraj Prem Kumar

303384, premyu@uni-hildsheim.de

Part 1: Distributed K-Means Clustering

Introduction:

We have a dataset "Absenteeism_at_work.csv" containing 21 columns (data features) and 740 rows (training data). This is a non-sparse dataset; every element has a value. In order to implement distributed K-means clustering, the approach is detailed via two parts:

A: K-means algorithm for given dataset

K-Means is an iterative and unsupervised machine-learning algorithm that seeks to cluster homogeneous or similar subgroups in our dataset. A cluster refers to a collection of data points aggregated together because of certain similarities (height, weight, age, marital status, etc.). The goal of clustering formally: to minimize the distance between observations within one cluster, across all clusters.

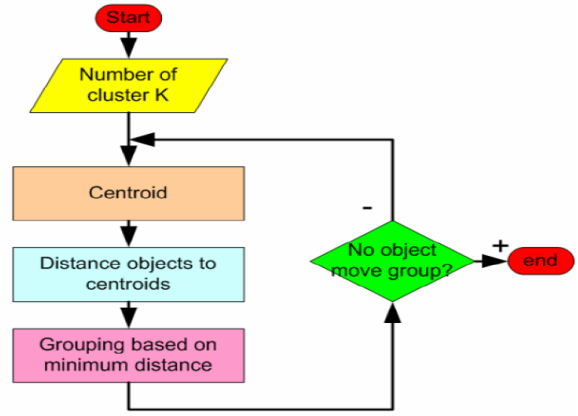


Figure 1: K-Means algorithm flowchart[1]

Step 1: Pre-processing data

Data is imported via pandas 'read_csv' package. Columns "ID" and "Reason for absence" are dropped on import – ID column is just an identifier for the step, and is not an actual feature of the data. "Reason for absence" is an encoded column, based on the International Code of Diseases (ICD). However, this again is just an arbitrary identifier and does not provide insight in the data clustering – we should not compute differences of ICD codes.

Step 2: Pick *K* random points as cluster centres (centroids)

Normally for setting an optimal value of *K*, we can use an elbow plot of the sum of squared errors (SSE). But this is not optimal for this kind of dataset, so I have chosen *K* = 3, by trial and error method. Noted that *K* is a hyperparameter; and using grid-search to set it before the learning process begins. Then we initialize *K* number of centroids.

Step 3: Compute distance between each row of the dataset with the initialized centroids

This is done by calculating *Euclidean distance* between the dataset row and each centroid according to the following formula:

$$\arg\min_{c_i \in C} dist\,(x, c_i)$$

where x is the data and c_i is the current centroid

Step 4: Set new centroid based on calculated average (mean) of all the dataset rows assigned to one particuler cluster:

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

S_i is the set of all dataset rows assigned to the i^{th} cluster

Step 5: Iteration process repeats until none of the cluster assignments change; that is the difference between the previous and current centroid are the zero.

<u>B: Distributed implementation strategy</u>

Step 1: Define the following functions, based on the lecture slides given

- initCenters(): Initialize random centroid array based on k and length of dataset features
- computeDistance(): Calculate *Euclidean distance* as outlined in part 1
- computeCenter(): Update the centroids of clusters based on new assignments. Returns the data rows belonging to each cluster, and the count of each data row for each cluster

Step 2: With rank 0 as the master node, the dataset is imported, and sliced according to the number of process. This follows the same approach as previous two labs. The initial centroid is generated here by calling the initCenters() function.

Step 3: Sliced data rows is ***scattered*** across worker nodes, the array size is equal. The initial centroid array is ***broadcasted*** to all worker nodes.

Step 4: The parallel computation takes place in an infinite while loop (while==true). Firstly the distance matrix is calculated by calling the computeDistance() function. The calculated sums are ***gathered*** back from worker nodes. Within the root node, the current centroid value is copied, and this is later compared across the iterations. The new centroid value is found by calculating the mean of the datarows. The centroid array is then ***broadcasted*** back to all workers for each iterations.

Step 5: The while loop exits once the difference in sum between previous centroid and current centroid is equal to zero, giving the final centroid assignment for the *k* clusters.

A sample program output, for size=4. It takes 13 iterations to reach the final centroid assignment. The program sets the random seed = 1, this is for debugging and validation purpose to ensure a consistent random value.

```
Current clusters arrangement  [[258.]
 [159.]
 [323.]] for iteration 13
Computed centroid changes: 0.0
Number of K-clusters 3
Final centroids after 13 iterations: [[6.76653696e+00 4.05447471e+00 2.60311284e+00 2.39941634e+02
  2.74941634e+01 1.15992218e+01 3.67354086e+01 2.71083335e+02
  9.49105058e+01 7.00389105e-02 1.34630350e+00 1.48638132e+00
  4.16342412e-01 3.50194553e-02 1.19455253e+00 7.56653696e+01
  1.70210117e+02 2.63073930e+01 6.26848249e+00]
 [7.06329114e+00 3.87341772e+00 2.62658228e+00 3.20563291e+02
  3.36582278e+01 1.06518987e+01 3.47848101e+01 2.76684519e+02
  9.44367089e+01 8.86075949e-02 1.10759494e+00 1.37974684e+00
  8.03797468e-01 1.01265823e-01 1.39240506e+00 8.02974684e+01
  1.73759494e+02 2.67974684e+01 8.70253165e+00]
 [5.66770186e+00 3.86024845e+00 2.48136646e+00 1.59844720e+02
  2.96366460e+01 1.43664596e+01 3.73788820e+01 2.71795661e+02
  9.52857143e+01 2.48447205e-02 1.35093168e+00 4.78260870e-01
  5.77639752e-01 9.00621118e-02 7.76397516e-02 8.18416149e+01
  1.74431677e+02 2.71614907e+01 6.63975155e+00]]
Total time taken: 0.17489944874614594
```

Part 2: Performance Analysis

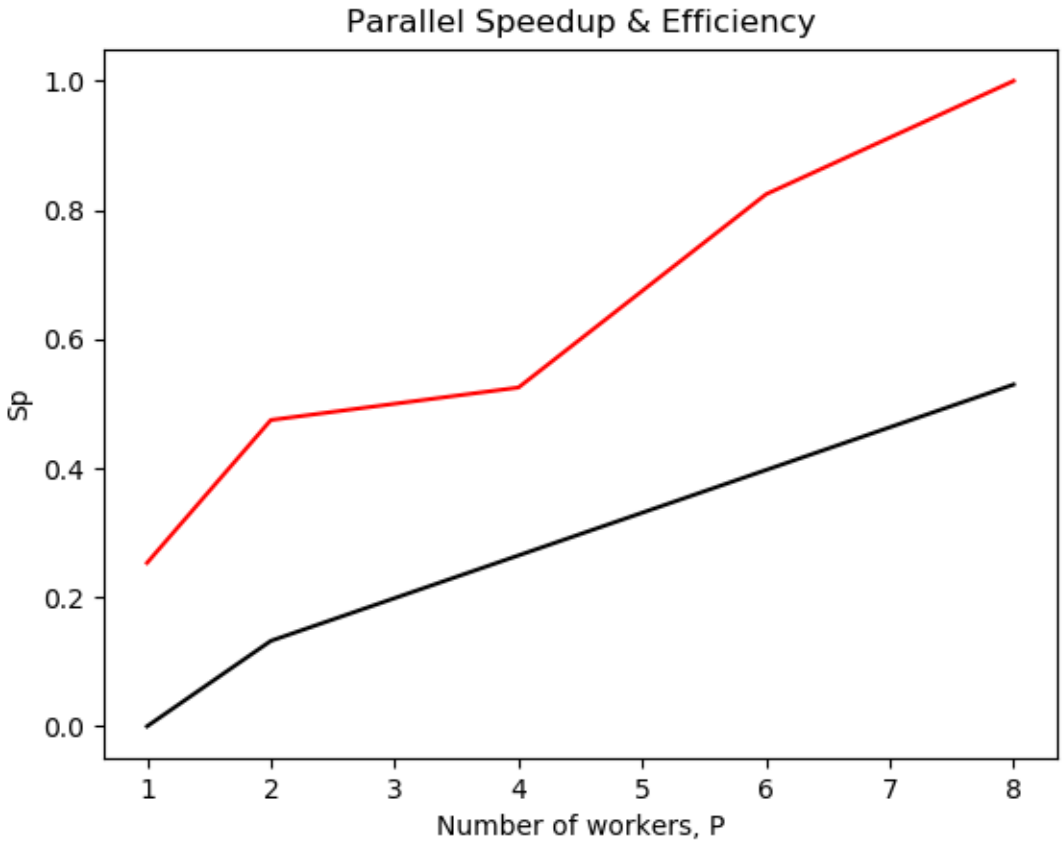
The full experiment is conducted with varying the number of workers, $P = \{1,2,4,6,8\}$. Table 1 shows the result set for program execution times

# Workers, P	Avg. processing time (s)
1	0.52953505
2	0.28256410
4	0.25546333
6	0.16265410
8	0.13409640

Table 1: Avg. processing time

The results are as expected, with a large (around 50%) from 1 to 2 processes. With 4 ,6, and 8 processes, there is a linear speedup observed. With more processes, there likely won't be much achievable improvement as shown in previous labs, as there will be an additional processing cost with all the interleaved processes and communication overhead.

The parallel speedup and efficiency plot shown below details the same results clearly. The read line represents the program's speedup line, whereas the black line is the linear speedup line. It's interesting to note that the speedup line is actually showing a higher efficiency compared to the expected speedup.



References:

[1] Fig. 5: Flow Chart for K Means Clustering. (2017, September 1). Retrieved from https://www.researchgate.net/figure/Flow-Chart-for-K-Means-Clustering_fig5_308811921

[2] K-Means Clustering: All You Need to Know. (2018, July 17). Retrieved from https://byteacademy.co/blog/k-means-clustering/

[3] Ramkishore, M. (2018, November 24). Unsupervised Learning With Python??'K- Means and Hierarchical Clustering. Retrieved from https://medium.com/datadriveninvestor/unsupervised-learning-with-python-k-means-and-hierarchical-clustering-f36ceeec919cPython Tutorial - Image Histogram - 2018. (n.d.). Retrieved from https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_image_histogram_calcHist.php

[4] NK, M. (n.d.). K-Means Clustering in Python. Retrieved from https://mubaris.com/posts/kmeans-clustering/

[5] Dr. Michael J. Garbade. (2018, September 12). Understanding K-means Clustering in Machine Learning. Retrieved from https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

[6] K-means Clustering in Python – Ben Alex Keen. (2017, May 10). Retrieved from http://benalexkeen.com/k-means-clustering-in-python/

[7] andrewxiechina/DataScience. (n.d.). Retrieved from https://github.com/andrewxiechina/DataScience/blob/master/K-Means/K-MEANS.ipynb

[8] K-Means. (n.d.). Retrieved from https://www.saedsayad.com/clustering_kmeans.htm

Appendix A

Code for graph plot

```
import numpy as np
import matplotlib.pyplot as plt

workers = [1, 2, 4, 6, 8]
Tp = [0.52953505, 0.28256410, 0.25546333, 0.16265410, 0.13409640]
Ts = Tp[np.argmin(Tp)]
Sp = [Ts / i for i in Tp]
linear = np.linspace(0, Tp[np.argmax(Tp)], 5, endpoint=True)

plt.plot(workers, Sp, color='red')
plt.plot(workers, linear, color='black')
plt.title('Parallel Speedup & Efficiency')
plt.xlabel('Number of workers, P')
plt.ylabel('Sp')
plt.show()
```

Appendix B

Full sample output for one run

```
C:\PythonProjects>mpiexec -n 4 python Lab3.py
Current clusters arrangement  [[166.]
 [203.]
 [371.]] for iteration 0
Computed centroid changes: 5.396919612271982
Current clusters arrangement  [[226.]
 [182.]
 [332.]] for iteration 1
Computed centroid changes: 13.113535327467932
Current clusters arrangement  [[259.]
 [170.]
 [311.]] for iteration 2
Computed centroid changes: 6.536202258372637
Current clusters arrangement  [[251.]
 [165.]
 [324.]] for iteration 3
Computed centroid changes: 6.487658899396626
Current clusters arrangement  [[262.]
 [155.]
 [323.]] for iteration 4
Computed centroid changes: 4.416393800699466
Current clusters arrangement  [[262.]
 [155.]
 [323.]] for iteration 5
Computed centroid changes: -0.5669684374882569
Current clusters arrangement  [[258.]
 [159.]
 [323.]] for iteration 6
Computed centroid changes: -2.539052749045677
Current clusters arrangement  [[258.]
 [159.]
 [323.]] for iteration 7
Computed centroid changes: -0.015756574850860218
Current clusters arrangement  [[258.]
 [159.]
 [323.]] for iteration 8
Computed centroid changes: -9.827506791686494e-05
Current clusters arrangement  [[258.]
 [159.]
 [323.]] for iteration 9
Computed centroid changes: -6.148928602608961e-07
Current clusters arrangement  [[258.]
 [159.]
 [323.]] for iteration 10
Computed centroid changes: -3.854856114871019e-09
Current clusters arrangement  [[258.]
 [159.]
 [323.]] for iteration 11
Computed centroid changes: -2.417366307128077e-11
Current clusters arrangement  [[258.]
 [159.]
 [323.]] for iteration 12
Computed centroid changes: -1.656869086374968e-13
Current clusters arrangement  [[258.]
 [159.]
 [323.]] for iteration 13
Computed centroid changes: 0.0
Number of K-clusters 3
Final centroids after 13 iterations: [[6.76653696e+00 4.05447471e+00 2.60311284e+00 2.39941634e+02
  2.74941634e+01 1.15992218e+01 3.67354086e+01 2.71083335e+02
  9.49105058e+01 7.00389105e-02 1.34630350e+00 1.48638132e+00
  4.16342412e-01 3.50194553e-02 1.19455253e+00 7.56653696e+01
  1.70210117e+02 2.63073930e+01 6.26848249e+00]
 [7.06329114e+00 3.87341772e+00 2.62658228e+00 3.20563291e+02
  3.36582278e+01 1.06518987e+01 3.47848101e+01 2.76684519e+02
  9.44367089e+01 8.86075949e-02 1.10759494e+00 1.37974684e+00
  8.03797468e-01 1.01265823e-01 1.39240506e+00 8.02974684e+01
  1.73759494e+02 2.67974684e+01 8.70253165e+00]
 [5.66770186e+00 3.86024845e+00 2.48136646e+00 1.59844720e+02
  2.96366460e+01 1.43664596e+01 3.73788820e+01 2.71795661e+02
  9.52857143e+01 2.48447205e-02 1.35093168e+00 4.78260870e-01
  5.77639752e-01 9.00621118e-02 7.76397516e-02 8.18416149e+01
  1.74431677e+02 2.71614907e+01 6.63975155e+00]]
Total time taken: 0.15642816914260038
```