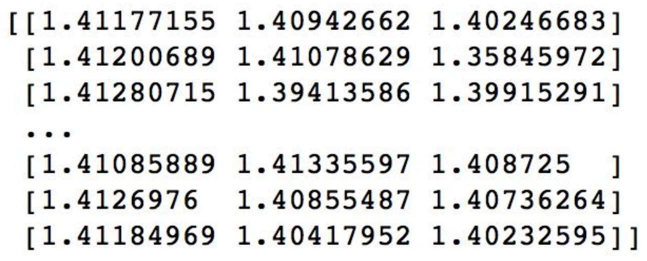
LAB DISTRIBUTED DATA ANALYTICS I
TUTORIAL III

Juan Fernando Espinosa Reinoso
303158
Group 1: Monday Tutorial

Implement K-means 10 Points

The dataset used is the 20NewsGroups. To get an accurate vectorization and avoid errors I made use of Sklearn library. The approach coded for this exercise follows the next sequence.

First, a function called euclidean distance which is in charge of measuring the distance between the centers to all the datapoints of the dataset.

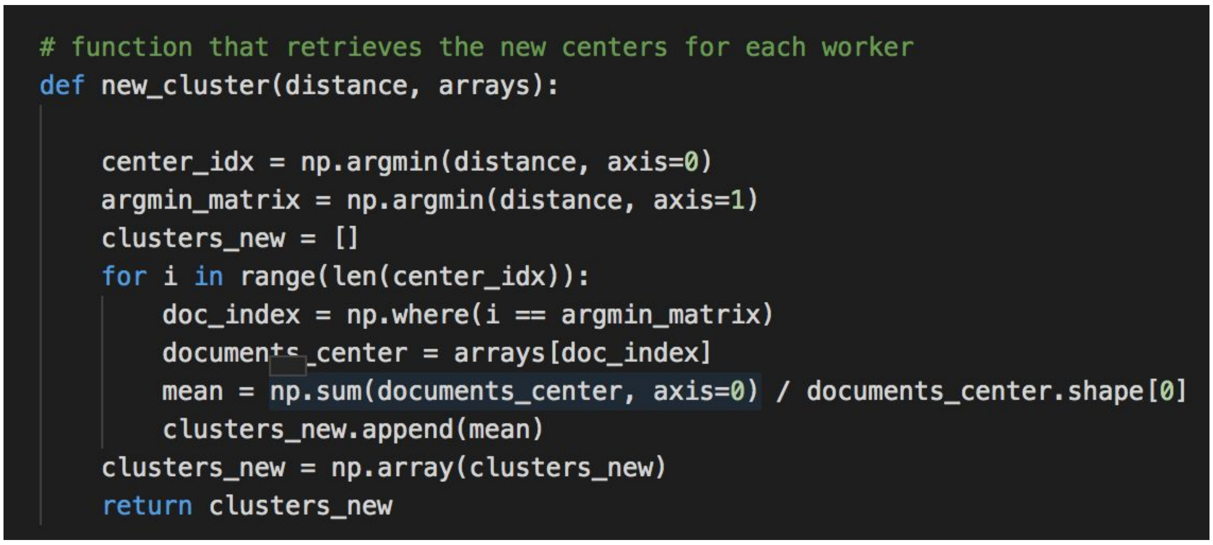


```
[[1.41177155 1.40942662 1.40246683]
 [1.41200689 1.41078629 1.35845972]
 [1.41280715 1.39413586 1.39915291]
 ...
 [1.41085889 1.41335597 1.408725  ]
 [1.4126976  1.40855487 1.40736264]
 [1.41184969 1.40417952 1.40232595]]
```

figure 1. Euclidean distance example - K=3

Second, a function called new_cluster is in charge of measuring the new centers for each worker. Let's split the code down:

- **center_idx** = takes the minimum argument in each column of the distance matrix.
- **argmin_matrix** = takes the minimum value for each datapoint. It will help for assigning the datapoint to the correct center.
- For each center_idx we will check and pick all the argmin_matrix datapoints that **match** with the index of the center_idx.
- Extract the datapoints by using the **indices** of the previous task.
- Take the **min** for the new centers calculation.
- Return the new centers.



```python
# function that retrieves the new centers for each worker
def new_cluster(distance, arrays):

    center_idx = np.argmin(distance, axis=0)
    argmin_matrix = np.argmin(distance, axis=1)
    clusters_new = []
    for i in range(len(center_idx)):
        doc_index = np.where(i == argmin_matrix)
        documents_center = arrays[doc_index]
        mean = np.sum(documents_center, axis=0) / documents_center.shape[0]
        clusters_new.append(mean)
    clusters_new = np.array(clusters_new)
    return clusters_new
```

--- MPI initialization ---

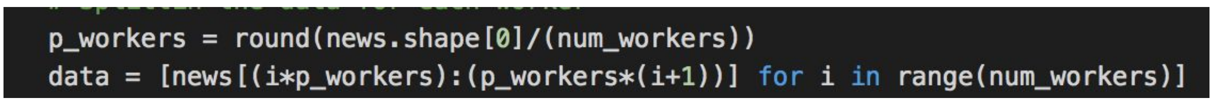
Before diving it, there are two main key factors to bear in mind: the dataset is required each iteration by all the worker and it does not change. On the contrary, the centers are constantly changing until finding the ideal clustering center. Therefore, the following MPI collective communication options has been applied: **bcast, comm.reduce & scatter.**

--- worker 0 ---

- Worker 0 first imports the Tfidf-vectorized data. Take the number of parts the dataset has to be splitted to give each worker an equal partition of the data. Additionally, randoms centers are being picked from the dataset.
- Select the number of centers (K)

--- MPI world ---

- Broadcast the initial centers to all the workers (see introductory paragraph)
- Scattering the dataset to each worker splitting it in equally parts. Each worker considering its rank will receive a bunch of data times the number of p_workers (dataset / num of workers)



```python
p_workers = round(news.shape[0]/(num_workers))
data = [news[(i*p_workers):(p_workers*(i+1))] for i in range(num_workers)]
```

  - max_iterations and iteration initialization.

--- Condition ---

iterate considering the number of iterations:

- each worker -

- Measure the distance between the initial centers to the data
- Get the new centers applying the function above explained
- Use comm.Barrier to wait all the workers to finish their task

- Global -

Apply **comm.Reduce** by summing the partial centers gotten in each worker.

- Worker 0 -

- Collects the new_centers from each worker and completes the mean calculation for the new centers to get the **Global centers.**
- Apply the condition: if the difference between the previous and new centers is less than threshold stop running because convergence has been reached.
- Otherwise, keep running.

--- Worker 0---

Print the final centers.

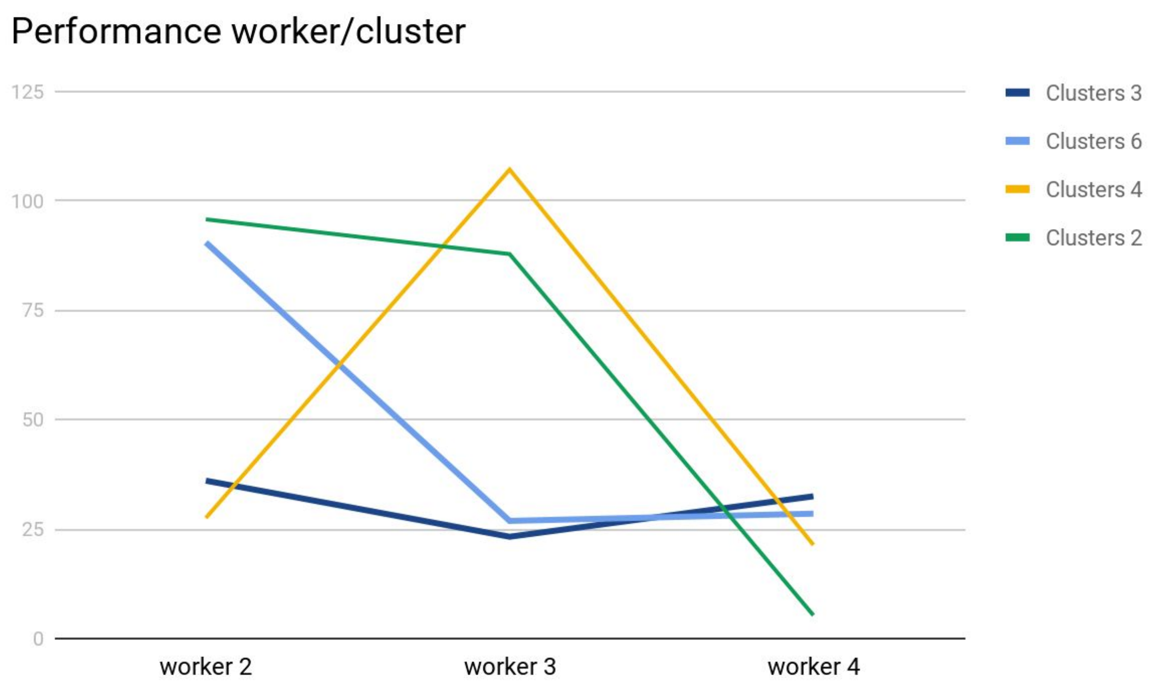
Note: Running all the dataset has caused problems with memory issues and processing capacity of the computer. The dataset output as a Scipy sparse matrix have created several memory issues in the running of the algorithm. Moreover, due to the length of data, transforming to dense or even Numpy array takes more memory. The data has been reduced for testing.

Performance Analysis 10 Points

RESULTS OF EXPERIMENTS

Note: Initial Centroids are picked randomly - the results vary in each running.

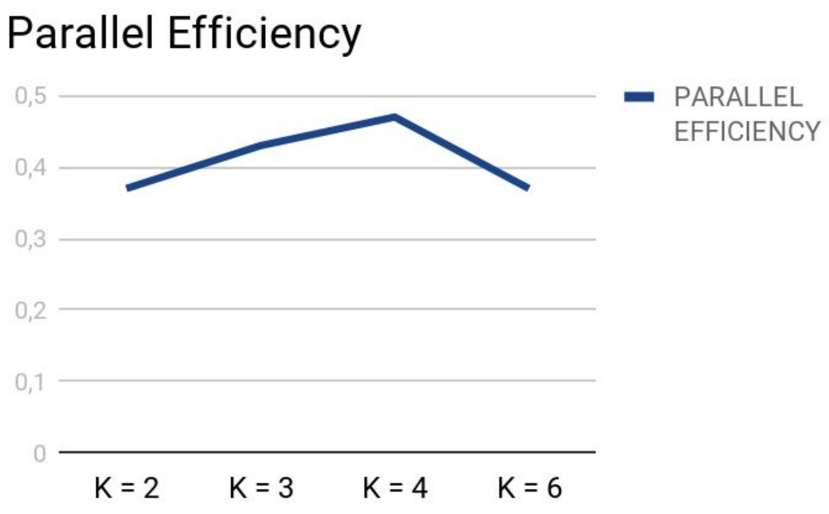
WORKERS | CLUSTERS	**2**	**3**	**4**	**6**
2	95.791324 / 10 iter	36.071977 / 15 iter	27.569627 / 13 iter	90.515571 / 75 iter
3	87.854439 / 75 iter	23.26714 / 14 iter	107.178 / reached max iter	26.8794 17 / iter
4	5.3195 / 2 iter	32.50136 / 22 iter	21.36905 / 12 iter	28.53640 / 16 iter



Since the computer has only two cores the speed up graph will be according to it.

By running several times for k = {2,3,4} an average execution time on Processes was found. Moreover, the best execution time was picked and plot in the following overall execution graph.

	K= 2	K=3	K=4
PARALLEL EFFICIENCY	0.37	0.43	0.47



Conclusion: It tends to be a sublinear speed up with a really poor performance by the time we increase the number of clusters. Moreover, checking the impact with different number of workers (without considering virtual ones) would be helpful to have the full picture of the behavior of the parallelization and its impact in the speed up.

References:

- http://users.ece.northwestern.edu/~wkliao/Kmeans/index.html
- https://kite.com/python/answers/how-to-append-an-element-to-a-key-in-a-dictionary-with-python
- https://stackoverflow.com/questions/35955293/pythonic-way-to-calculate-distance-using-numpy-matrices