Distributed Computing Final project

Youssef Jarraya 1893427

2019 January 18th

1 Problem to solve

1.1 Optimisation problem:

How can we find the best clusters repartition depending on a given number of centroids and a given set of data ?

1.2 Specifications:

- The set of data can have a big set of elements
- The elements in the set of data can have many dimensions
- The number of centroids are not fixed
- Have a graphical and textual result

2 Program and Algorithm

2.1 The structure / workflow of the program - Algorithm:

The chosen algorithm to solve the above problem, is the **K-means algorithm**. This algorithm will alternate between two phases during a fixed number of iterations or until achieving a stable solution: **update** and **assignment**
This loop can stop if it reaches the number maximum of iteration k or if the stabilised_solution return true.
get_closest is the **assignment** and
update_algorithm is the **update**.

```
for x in range(0, k):
    if x != 0 and stabilised_solution(map_centroid_old, map_centroid, nb_centroid, stop_criteria):
        break
    else:
        map_centroid_old = map_centroid
    list_closest = get_closest(sc, map_points, map_centroid)
    map_centroid = update_algorithm(map_points, list_closest, nb_centroid, sc, map_centroid)
```

Figure 1: Main loop

Assignment :
We will look for the closest centroid $c_k^{(t)}$ for every point and put them together. It will form clusters for each centroid at each iteration t.
Formula :

$$C_k^{(t)} = \{x_p : ||x_p - c_k^{(t)}||^2 \geqslant ||x_p - c_j^{(t)}||^2, 1 \geqslant j \geqslant K\}, k = 1, ..., K$$

With $C_k^{(t)}$ the new cluster k at the iteration t, x_p the point to affect to a cluster and K the maximum number of clusters.

```
def get_closest(sc, points, centers):
    point = points.collect()
    center = centers.collect()

    dist_matrix = spatial.distance_matrix(point, center)
    result_id = nump.array([dist_matrix.argmin(axis=1)])
    result = sc.parallelize(result_id[0])
    return result
```

Figure 2: get_closest function

This function will calculate the closest centroid for each point and return a list of centroid index.

Update :
We will do the sum of all the points in a cluster and divide the result by the cardinality of that given cluster.It is applied to all clusters until we have a new set of centroids.
Formula:

$$c_k^{(t+1)} = \frac{1}{n_k^{(t)}} \sum_{x_j \in C_k^{(t)}} x_j, k = 1, ..., K.$$

With $c_k^{(t+1)}$ the new centroid at the iteration t + 1, $n_k^{(t)} = |C_k^{(t)}|$, k the number of the centroid and x_j the points of a given cluster.

```
def update_algorithm(points, list_closest, nb_c, sc, map_centroid):
    point = points.collect()
    closest = list_closest.collect()
    cluster = map_centroid.collect()
    nb_coord = len(point[0])
    nb_points = len(closest)

    for x in range(0, nb_c):
        result_sum = [0.0] * nb_coord
        result_card = 0
        for l in range(0, nb_points):
            if closest[l] == x:
                for coord in range(0, nb_coord):
                    result_sum[coord] += point[l][coord]
                result_card += 1
        if result_card != 0:
            for coord in range(0, nb_coord):
                result_sum[coord] /= result_card
            cluster[x] = result_sum

    map_result = sc.parallelize(cluster)
    return map_result
```

Figure 3: update_algorithm function

This function will update the centroids for each centroid "x" by doing the sum of the points "l" (for each dimension "coord") depending on "list_closest" and

dividing by the cardinality "result_card".It will create a map of centroids in "cluster" and return it.

2.2 Functionality:

- Read csv files

```
with open(log_file) as inputfile:
    next(inputfile)
    reader = csv.reader(inputfile)
    inputm = list(reader)

float_points = list(np.float_(inputm))
```

Figure 4: csv reader code

- Random selection of centroids using numpy

```
for item in range(nb_centroid):
    float_centroids.append(float_points[np.random.randint(len(float_points))])
```

Figure 5: Random centroids code

- Have a graphical display of the result using matplotlib

```
def display(list_points, list_closest, name_file):
    points = list_points.collect()
    closest = list_closest.collect()

    pca = PCA(n_components=2)
    data = pca.fit_transform(points)
    plt.scatter(data[:, 0], data[:, 1], c=closest, alpha=0.5)

    plt.legend(loc='upper left')
    plt.title(label=name_file)
    plt.show()

    return
```

Figure 6: Display function

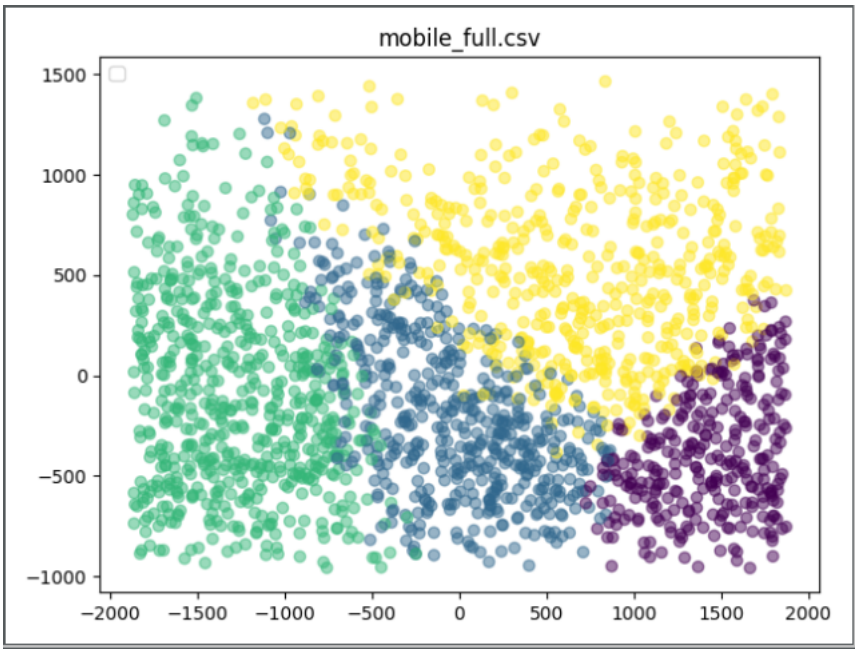

Figure 7: Result

3 Performance evaluation

- The optimisation is solve
- The specifications are respected
- For 2001 points with 20 dimensions, 4 clusters with 20 dimensions and the display on : we have 5.14 seconds of execution time