University of Abdelhamid Mehri-Constantine2
Faculty: NTIC
Department: IFA
Master's second year SDIA 2024/2025

Lab 2 BDPA
K-means in Spark

In this lab, we will implement the K-means clustering algorithm from scratch using Apache Spark's RDD (Resilient Distributed Dataset) API. Unlike traditional implementations, which often rely on high-level machine learning libraries, this exercise will deepen your understanding of Spark's distributed computing capabilities. It's all about how to apply transformations and actions to RDDs to manage data distribution, perform calculations in parallel, and iterate over data in Spark.
Below the Pyspark code of the main algorithm that we want to complete.

```
max_iter = 100
conv = False
iter = 1
k = 2
centroids = centroids_init(dataRdd, k)
old_centroids = []
while iter < max_iter and not conv :
  # Complete This instruction :
  # labeledRdd = …
  centroids = new_centroids(labledRdd, k)
  conv = (old_centroids == centroids)
  old_centroids = centroids
  iter += 1
```

Part 1 : Key functions

Implement the following functions in PySpark:

1. **`centroids_init(data, k)`:**
   - **Purpose:** Initializes k centroids randomly within the range of the data.
   - **Input:**
     - data: An RDD containing the data points.
     - k: The number of clusters.
   - **Output:** A list of k centroids, where each centroid is a tuple of (x, y) coordinates.
2. **`euclidian_distance(point1, point2)`:**
   - **Purpose:** Calculates the Euclidean distance between two points.

- **Input:**
    - point1: The first point (a tuple of coordinates).
    - point2: The second point (a tuple of coordinates).
- **Output:** The Euclidean distance between the two points.

3. `get_label(point, centroids)`:

- **Purpose:** Assigns a cluster label to a data point based on its distance to the centroids.
- **Input:**
    - point: The data point to label.
    - centroids: A list of centroids.
- **Output:** The cluster label (an integer) assigned to the point.
- **Tips:** Use the `euclidian_distance(point1, point2)` to assign the point to his cluster

4. `calculate_mean_point(labeled_rdd, label)`:

- **Purpose:** Calculates the mean point for a given cluster label.
- **Input:**
    - labeled_rdd: An RDD containing data points with their assigned labels.
    - label: The cluster label for which to calculate the mean point.
- **Output:** The mean point (a tuple of coordinates) for the specified cluster label.
- **Tips:** use filter() operation for each label (i.e., label= 0..k) and map(),reduce() operation to compute the required calculation

5. `new_centroids(data, k)`:

- **Purpose:** Calculates new centroids based on the mean points of the current clusters.
- **Input:**
    - data: An RDD containing labeled data points.
    - k: The number of clusters.
- **Output:** A list of new centroids.
- **Tips:** use `calculate_mean_point(labeled_rdd, label)` function to get the mean point of each cluster.

Part 2 Main algorithm

Write the code of the missing instruction in the main algorithm such as :
We want to create a new Rdd called labeldRdd in which, we assign each point of dataRdd to his cluster using `get_label(point, centroids)` function.

Execute the code using the following data and plot the result for each iteration.

```
data = [(1,1) , (2,2), (3,3) , (10,10) , (11,11), (12,12)]
dataRdd = sc.parallelize(data)
```