**GROUP NUMBER:** 36

**GROUP MEMBERS:** Francesco Visonà, Alessandro Dario, Luca Pellegrini

**AVAILABLE INPUTS:** Input files are available in the hdfs file system: /data/BDC2425/artificial1M7D100K.txt and /data/BDC2425/artificial4M7D100K.txt

**PART 1:** The goal of this test is to assess the scalability of the standard and fair implementations. The test must be performed on file artificial4M7D100K.txt. However, if your implementation is slow (i.e., taking more than 10 minutes for the slowest run), you can use the smaller file artificial1M7D100K.txt. You must use the following parameters: L=16, K=100, M=10.

Fill in the following table.

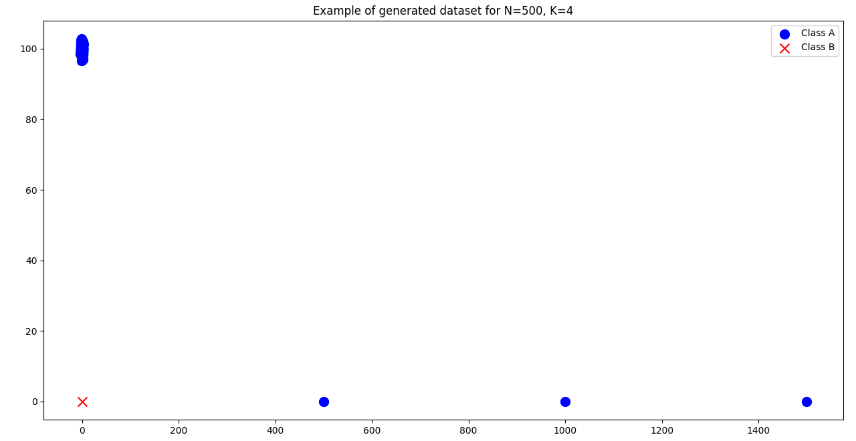
**Name of used file:** /data/BDC2425/artificial4M7D100K.txt

|  |  |  |  |
| --- | --- | --- | --- |
| **SCALABILITY WITH RESPECT TO NUMBER OF EXECUTORS** | | | |
| **Number of executors** | **Spark Lloyd’s implementation** | **MRFairLloyd** | **MRComputeFairObjective** |
| 2 | 12738 | 42797 | 1891 |
| 4 | 8874 | 27114 | 1249 |
| 8 | 7131 | 16925 | 911 |
| 16 | 3054 | 7598 | 293 |

**General hints:**

* Remember that Spark uses the lazy evaluation for constructing an RDD. Therefore, be sure to include an action on the final RDD when you take running times.
* Any used RDD in your program should be cached.
* Do not include the reading of the input in your running times.

**PART 2:** Describe the program GxxGEN.java or GxxGEN.py that you have implemented for point 5 of the specifications. Include a brief high-level description of your program and the constraints, if any, on the input parameters (e.g., the minimum number of points *N*).

****

Our generator can work for every positive integer K and N > K+1, and follows closely what is represented in *Figure 1*.

It creates K-1 points of group A at distance 500 one from the other on the x axis, starting from [500,0] (the blue points that are in [500,0], [1000,0] and [1500,0] in *Figure 1*). Then it puts one single point of group B in [0,0] and the remaining N-K points are given to group A and put near [100,0], in a ball of radius approximately 1 generated randomly.

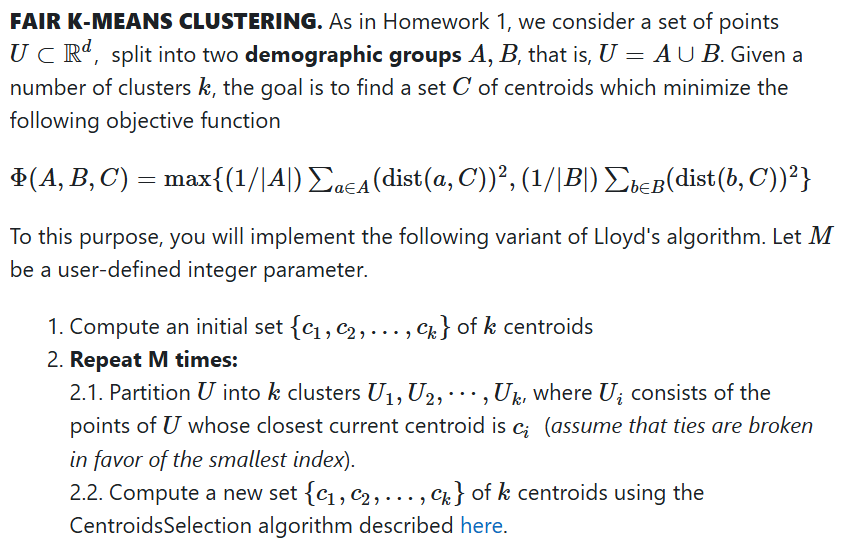
This procedure is done to make it possible to obtain a dataset where the fair objective function of the solution returned by the standard Lloyd’s algorithm is nearly 4 times bigger than the fair objective function of the solution returned by its fair version.

Figure 1. Example of dataset generated by our procedure with N=500 and K=4.

Both the algorithms will with high probability put K-1 centers in the points of group A that are on the x axis, but the real difference is in how they treat the other N-(K-1) points.

The standard K-Means algorithm will put a center closer to [100,0] (the larger N-K is, the closer the center will be to [100,0], but even with N-K = 2 the center will be put in [67,0]), while its fair version will put a center near [50,0] (since the two algorithms weight in a different way the only point of group B).

This means that the objective function:



will be:

1. In the case of the standard Lloyd’s algorithm, considering the distance of every point of group A to its closest center to be really small, while the distance of the only point of group B to its closest center is 100:
2. In the case of its fair version, if one of the centers is put in [50,0]:

This means that the objective function of the standard Lloyd’s algorithm will tend to be 4 times the objective function of its fair version. Experimentally, this also works fine with small N, respecting N > K+1.