Homework 3

The purpose of this homework is to run and test a MapReduce algorithm implemented in Spark on the CloudVeneto cluster available for the course. As a case study, we will consider a **coreset-based MapReduce algorithm** for **diversity maximization**, an important NP-hard combinatorial optimization problem, whose best polynomial-time approximation algorithm is quadratic, hence impractical for large instances.

Diversity Maximization

Given a set P of N points in a metric space and an integer k < N, *diversity maximization* (*remote-clique variant*) requires to find k distinct points of P so to maximize their average distance (i.e., the sum of their k*(k-1)/2 pairwise distances divided by k*(k-1)/2). Diversity maximization is an important primitive for big-data application domains such as aggregator websites, web search, recommendation systems.

- **2-approximate sequential algorithm**: for floor(k/2) times, select the two unselected points with maximum distance. If k is odd, add at the end an arbitrary unselected point. *For datasets of millions/billions points, the algorithm, whose complexity is quadratic in N, becomes impractically slow*.

- **4-approximation coreset-based MapReduce algorithm**: Partition P into L subsets and extract k points from each subset using the Farthest-First Traversal algorithm. Compute the final solution by running the 2-approximate sequential algorithm on the coreset of L*k points extracted from the L subsets.

These informations, with some other details, can be found on these slides on diversity maximization.

We will work with *points in Euclidean space* (real cooordinates) and with the standard *Euclidean L2-distance*.

Using CloudVeneto

A brief description of the cluster available for the course, together with instructions on how to access the cluster and how to run your program on it are given in this User guide for the cluster on CloudVeneto.

Assignment

1. **Download method/function `runSequential(pointSet,k)`**: it receives in input a set of points (`pointSet`) and an integer k, and runs the sequential 2-approximation algorithm for diversity maximization returning k solution points.

   - **For Java users:** the code for the sequential algorithm can be downloaded here (note that it represents input/output as `ArrayList<Vector>`).

   - **For Python users:** the code for the sequential algorithm can be downloaded here (note that it represents input/output as `list` of `tuple`).

2. **Develop the following methods/funcions**

   (a) **`runMapReduce(pointsRDD,k,L)`**: implements the 4-approximation MapReduce algorithm for diversity maximization described above. More specifically, it receives in input an RDD of points (`pointsRDD`), and two integers, k and L, and performs the following activities.

   - Round 1: subdivides `pointsRDD` into L partitions and *extracts k points from each partition using the Farthest-First Traversal algorithm*.

     Hints:

(a) For the partitioning, if you invoke `repartition(L)` when the RDD was created, you can use the Spark Partitions, accessing them through the `mapPartition` method, similarly to what was done in Homework 1. (*For Java users*: in Homework 1 `mapPartitionToPair` was used because you had to create (key,value) pairs, but here, at the end of Round 1, you do not need pairs but just points (i.e., instances of Vector), so use `mapPartition` instead.)

(b) Recycle the implementation of Farthest-First Traversal algorithm developed for Homework 2.

- Round 2: collects the `L*k` points extracted in Round 1 from the partitions into a set called `coreset` and returns, as output, the k points computed by `runSequential(coreset,k)`. Note that `coreset` is not an RDD but an `ArrayList<Vector>` (in Java) or a `list` of `tuple` (in Python).

Instrument your code to separately measure and print the running times of Round 1 and Round 2.

(b) **`measure(pointsSet)`**: receives in input a set of points (`pointSet`) and computes the average distance between all pairs of points. The set `pointSet` must be represented as `ArrayList<Vector>` (in Java) or `list` of `tuple` (in Python).

3. **Write a program GxyHW3.java** (for Java users) or **GxyHW3.py** (for Python users), where xy is your two-digit group number, which receives in input a path to a file containing a set of points in Euclidean space, and the two integers "k" (parameter for diversity maximization) and "L" (number of partitions). *The file must contain one point per line, with coordinates separated by comma.*

   The program incorporates the methods/functions downloaded or developed as specified above, and does the following:

   - Initializes the Spark context as was done in Homework 1. For Python users: when defining the configuration do not use the invocation `setMaster("local[*]")` which you see in the template for Homework 1.

   - Reads the file path and the parameters "k" and "L". Then reads the points from the file into an RDD called "inputPoints" (a `JavaRDD<Vector>` in Java, and a RDD of `tuple` in Python). For reading the points you can execute

     ```
     sc.textFile(inputPath).map(f).repartition(L).cache();
     ```

     where `sc` is the Spark context, `inputPath` is the path to the input file, and `f` is a function that creates a `Vector` in Java (`tuple` in Python) from a string representing its coordinates (a similar transformation was used already in Homework 2, so copy it from there). Note that by invoking `repartition(L)` you create L random partitions of the points which you can use in Round 1 of the runMapReduce, as suggested above. After reading the parameters and creating the RDD, it prints the following lines:

     ```
     Number of points =   number of input points
     k =    value of k
     L =    value of L
     Initialization time =  time (in ms) to read data and create the RDD
     ```

   - Runs `runMapReduce(inputPoints,k,L)` and saves the returned points (which is the solution to the diversity maximization problem) in a variable `solution`. Then, it prints the following lines:

     ```
     Runtime of Round 1 =  time (in ms) of Round 1
     Runtime of Round 2 =  time (in ms) of Round 2
     ```

   - Determines the average distance among the solution points by running `measure(solution)` and prints the following line:

`Average distance =` *`average distance among the solution points`*

Important remark. Time measurements in Spark require some care when using RDDs, due to the *lazy evaluation* mechanism. Please read what is written about this issue in the Spark programming guide.

4. **Test your program in local mode on your PC** to make sure that there are no problems. As datasets for this local test you can download the datasets Uber-small and Zalando that are descibed in this webpage.

5. **Test your program on the cluster** using the large dataset GloVe that is described here. (*Python users: there are memory problems in loading the full GloVe dataset; so, please use the reduced version of the dataset with 200K rather than 2M points*.) Use various configurations of parameters and report your results using the form FormHW3xy.docx.

   When using the cluster, ***you must strictly follow these rules***:

   - To avoid congestion, groups with even (resp., odd) group number must use the clusters in even (resp., odd) days.
   - Do not run several instances of your program at once.
   - Do not use more than 16 executors.
   - Try your program on a smaller dataset first (e.g, Uber-small or Zalando).
   - Remember that if your program is stuck for more than 1 hour, its execution will be automatically stopped by the system.

SUBMISSION INSTRUCTIONS. **Each group must submit only one homework** using the submission link found in the **Homework 3 section** in the Moodle page of the course. Specifically, each group must submit **2 files:**

- The program (**GxyHW3.java** or **GxyHW3.py**)
- The form FormHW3xy.docx

where xy is the group's number.

Make sure that your code be free from compiling and run-time errors, otherwise your score will be penalized.

*If you have questions about the assignment, contact the teaching assistants (TAs) by email to bdc-course@dei.unipd.it . The subject of the email must be "**HW3 - Group xy**", where xy is your group's number. If needed, a zoom/skype meeting between the TAs and the group will be organized.*