BIG DATA COMPUTING (Classes A and B, proff. Pietracaprina and Silvestri) 2021-2022

Assignment of Homework 2: DEADLINE May 17, 23.59pm

INTRODUCTION. Homeworks 2 and 3 will focus on the **k-center with z outliers problem**, that is, the robust version of the k-center problem which is useful in the analysis of noisy data (a quite common scenario in big data computing). Given a set P of points and two integers k and z, the problem requires to determine a set $S \subset P$ of k centers which minimize the maximum distance of a point of P-Z from S, where Z are the z farthest points of P from S. In other words, with respect to the standard k-center problem, in the k-center with z outliers problem, we are allowed to disregard the z farthest points from S in the objective function. Unfortunately, the solution of this problem turns out much harder than the one of the standard k-center problem. The 3-approximation sequential algorithm by Charikar et al. for k-center with z outliers, which we call **kcenterOUT**, is simple to implement but has superlinear complexity (more than quadratic, unless sophisticated data structures are used). The rigorous specification of the problem and the description of kcenterOUT with pseudocode, can be found in this set of slides: PresentationHW2.pdf.

The two homeworks will demonstrate that in this case a coreset-based approach can be successfully employed. In Homework 2 you will implement the 3-approximation sequential algorithm and will get a first-hand experience of its inefficiency. In Homework 3, you will implement a 2-round MapReduce coreset-based algorithm for the problem, where the use of the inefficient 3-approximation is confined to a small coreset computed in parallel through the efficient Farthest-First Traversal.

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

FOR JAVA USERS. In Spark, points can be represented as instances of the class *org.apache.spark.mllib.linalg.Vector* and can be manipulated through static methods offered by the class *org.apache.spark.mllib.linalg.Vectors*. For example, method Vectors.dense(x) transforms an array x of double into an instance of class Vector. Given two points x and y, instances of Vector, their Euclidean L2-distance can be computed by invoking: Math.sqrt(Vectors.sqdist(x, y)). Details on the classes can be found in the Spark Java API. **Warning.** *Make sure to use the classes from the org.apache.spark.mllib package. There are classes with the same name in org.apache.spark.ml package which are functionally equivalent, but incompatible with those of the org.apache.spark.mllib package.*

FOR PYTHON USERS. We suggest to represent points as the standard *tuple of float* (i.e., point = (x1, x2, ...)). Although Spark provides the class Vector also for Python (see pyspark.mllib package), its performance is very poor and its more convenient to use tuples, especially for points from low-dimensional spaces.

TASK for HOMEWORK 2. you must

1. **Develop a method SeqWeightedOutliers(P,W,k,z,alpha) which implements the weighted variant of kcenterOUT** (the 3-approximation algorithm for k-center with z-outliers). The method takes as input the set of points P, the set of weights W, the number of centers k, the number of outliers z, and the coefficient alpha used by the algorithm, and returns the set of centers S computed as specified by the algorithm. It is understood that the i-th integer in W is the weight of the i-th point in P. **Java users**: represent P and S as *ArrayList<Vector>*, and W as *ArrayList<Long>*. **Python users**: represent P and S as *list of tuple* and W as *list of integers*. Considering the algorithm's high complexity, try to make the implementation as efficient as possible.
2. **Develop a method ComputeObjective(P,S,z)** which computes the value of the objective function for the set of points P, the set of centers S, and z outliers (the number of centers, which is the size of S, is not needed as a parameter). Hint: you may compute all distances d(x,S), for every x in P, sort them, exclude the z largest distances, and return the largest among the remaining ones. ***Note that in this case we are not using weights!***
3. **Write a program GxxxHW2.java** (for Java users) or **GxxxHW2.py** (for Python users), where xxx is your 3-digit group number (e.g., 004 or 045), which receives in input the following command-line (CLI) arguments:

    - **A path to a text file containing point set in Euclidean space**. Each line of the file contains, separated by commas, the coordinates of a point. Your program should make no assumptions on the number of dimensions!
    - **An integer k** (the number of centers).
    - **An integer z** (the number of allowed outliers).

The program must do the following:

- Read the points in the input file into an ArrayList<Vector> (list of tuple in Python) called **inputPoints**. To this purpose, you can use the code provided in the file InputCode.java, for Java users, and InputCode.py, for Python users.
- Create an ArrayList<Long> (list of integer in Python) called **weights** of the same cardinality of inputPoints, initialized with all 1's. (In this homework we will use unit weights, but in Homework 3 we will need the generality of arbitrary integer weights!).

- Run **SeqWeightedOutliers(inputPoints,weights,k,z,0)** to compute a set of (at most) k centers. The output of the method must be saved into an ArrayList<Vector> (list of tuple in Python) called **solution**.
- Run **ComputeObjective(inputPoints,solution,z)** and save the output in a variable called **objective.**
- Return as output the following quantities: n =|P|, k, z, the initial value of the guess r, the final value of the guess r, and the number of guesses made by SeqWeightedOutliers(inputPoints,weights,k,z,0), the value of the objective function (variable objective), and the time (in milliseconds) required by the execution of SeqWeightedOutliers(inputPoints,weights,k,z,0). IT IS IMPORTANT THAT ALL PROGRAMS USE THE SAME OUTPUT FORMAT AS IN THE FOLLOWING EXAMPLE: **OutputFormat.txt**

Test your program using the datasets available here (the outputs of some runs of the program on those datasets will be made available at the same link)

ADD DATASETS and SUGGESTED VALUES OF k and z

SUBMISSION INSTRUCTIONS. Each group must submit a single file (**GxxxHW2.java** or **GxxxHW2.py** depending on whether you are Java or Python users, where xxx is your group ID). Only one student per group must submit the files in Moodle Exam using the link provided in the Homework2 section. Make sure that your code is free from compiling/run-time errors and that you use the file/variable names in the homework description, 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 "**HW2 - Group xxx**", where xxx is your ID group. If needed, a zoom meeting between the TAs and the group will be organized.*

Last modified: Thursday, 28 April 2022, 3:39 PM

◄ Submission Homework 1

Jump to...

Datasets for Homework 2 ►