COMP 9102 Data Management and Information Retrieval
Assignment 2
Indexing and Similarity Search for Dense Multidimensional Vectors
Due Date: Oct 31st, 2022, 5:00pm

You are going to use synthetically generated data. The file that contains the data is data10K10.txt and you can download it from Moodle. The file contains 10000 lines and each line is a sequence of 10 floats separated by commas. Each line is a 10-dimensional object and the ID of the object is implied by the line number (starting from 0). For example, the 10 numbers at the first line are the values of object with identifier 0 in each of the 10 dimensions.

You are going to implement three methods for evaluating similarity queries on this dataset. The first one is a naïve, linear scan method. The second is the pivot-based method explained in slides 40-42 of class "multidimensional data". The third one is the iDistance method explained in slides 43-44 of class "multidimensional data". You are going to use Euclidean distance (L_2) as the distance measure.

Step 1: data loading and computation of pivots

Read the data from the file into your program and put them in a data structure D (e.g., array, vector, list), such that each object is directly accessible given its ID. Then, write a function that takes as input your data structure and a parameter numpivots and computes and returns a set of object identifiers, which will serve as pivots. Notice that the set of pivots are objects selected from the dataset, so you only need to return an integer array of size numpivots with the identifiers of the pivots.

The procedure of selecting pivots is as follows. You take the first object in D and set it as a seed. Then, the first pivot is the *farthest object from the seed*. The second pivot should be the *farthest object from the first pivot*. The k-th pivot is the object o with the largest sum: $dist(p_0,o)+dist(p_1,o)+\ldots+dist(p_{k-2},o)$, where $\{p_0, p_1, \ldots p_{k-2}\}$ are the k-1 first pivots.

Your function should return not only the pivot identifiers, but a 2D array which keeps the distance from each object to all pivots; i.e., distances[i][j], should be the distance from object i pivot j.

Run your function for numpivots=5 and print the pivots that you have found, e.g.

```
pivots: [1109, 6878, 5514, 5026, 5338]
```

Step 2: iDistance method

Use the results of the function that computes the pivots and their distances to the objects (Step 1) to develop an iDistance index as follows. For each object o, find the nearest pivot npiv(o) to o and mark it. While doing so, for each pivot p keep track of the maximum distance maxd(p) to any o for which p is the nearest pivot. Then, compute the maximum maxd of all maxd(p). Compute an array of (iDist, oid) pairs, where oid is an object-id and iDist is the iDistance value of the object. Let $\{p_0, p_1, \ldots p_{m-1}\}$ be the pivots. Assuming that p_i is the nearest pivot to object o, the iDistance of object o is $i*maxd + dist(o,p_i)$. After adding the entry (iDist, oid) for each object to the array, sort the iDistance array. Write a function, which computes the iDistance array, maxd(p) for each pivot p, and maxd = max{maxd(p)}.

Step 3: range similarity queries

A range similarity query computes, for a given (10-dimensional) query point q and a given distance bound ε, the set of objects o, for which dist(q,o) ≤ ε. Write three functions for the evaluation of range similarity queries:

a) A function which applies a naïve approach: for each object o in the array of objects, compute dist(q,o) and if dist(q,o) ≤ ε add the identifier of the object to the result. At the end, return the query result.
b) A function which applies the pivot-based search approach described in slides 41-43: for each object o in the array of objects and for each pivot p_i, if $|dist(p_i,o) - dist(p_i,q)| > \varepsilon$, prune o; if o is not pruned, then compute dist(q,o) and if dist(q,o) ≤ ε add the identifier of the object to the result. $dist(p_i,o)$ can be accessed from the 2D array of Step 1 and $dist(p_i,q)$ should be computed once for each pivot p_i.
c) A function which applies the iDistance-based search approach described in slide 45. Note that we will not use a B^+-tree to index the iDistances but just a sorted array as discussed in Step 2. You should use binary search to find the first iDistance entry greater than or equal to the lower bound and scan from thereon. Note that, for a given pivot p_i, you should avoid accessing objects which do not have p_i as their nearest pivot.

To test the effectiveness and efficiency of the above methods, use the queries drawn from file queries10.txt. For each method, count (i) the average time required to evaluate each query and (ii) the number of distance computations that it needs to perform. Obviously, (ii) is 10000 in the naïve approach. Example output of your test for numpivots=10 and ε=0.2:

```
average distance comp per query (Naive) = 10000
average distance comp per query (Pivot-based) = 16.71
average distance comp per query (iDistance) = 2074.8
total time Naive = 4.390295028686523
total time Pivot-based = 1.146554946899414
total time iDistance = 1.2163665294647217
```

Your program may take as command-line arguments the number of pivots (numpivots) and the distance bound ε.

Step 4: kNN similarity queries

A kNN similarity query computes, for a given (10-dimensional) query point q and a given positive integer k, the set of k objects o with the smallest dist(q,o). Write three functions for the evaluation of kNN similarity queries:

a) A function which applies a naïve approach: for each object o in the array of objects, compute dist(q,o) and use a priority queue (heap) to keep track of the k objects with the smallest distance. At the end, return the k nearest objects to q and their distances.
b) A function which applies the pivot-based search approach. Initialize an empty priority queue (heap) H. For each object o in the array of objects, if the size of H is less than k, then compute dist(q,o) and add the object to the heap. If the size of H is k, then use the top of the heap (maximum distance among the k smallest ones) as ε; for each pivot p_i if $|dist(p_i,o) - dist(p_i,q)| > \varepsilon$, prune o; if o is not pruned, then compute dist(q,o) and if dist(q,o) ≤ ε replace the top heap element and maintain the heap property. $dist(p_i,o)$ can be accessed from the 2D array of Step 1 and $dist(p_i,q)$ should be computed once for each pivot p_i.
c) A function which applies the iDistance-based search approach described in slide 45. You should find the nearest pivot to the query object q and for the objects that are in the partition of that pivot, apply

method (a) to find the kNN set. For the remaining pivots, use kNN-th distance so far as a bound ε to possibly prune them and their partitions (as in Step 3c). If a pivot is not pruned, scan all objects in the partition to potentially update the kNN. After all pivots are processed the method returns the k nearest objects to q and their distances.

To test the effectiveness and efficiency of the above methods, use the queries drawn from file queries10.txt. For each method, count (i) the average time required to evaluate each query and (ii) the number of distance computations that it needs to perform. Obviously, (ii) is 10000 in the naïve approach. Example output of your test for numpivots=10 and k=5:

```
average distance comp per query (Pivot-based kNN) = 3156.76
average distance comp per query (iDistance kNN) = 6129.025
total time Naive kNN = 4.891640663146973
total time Pivot-based kNN = 4.830014228820801
total time iDistance kNN = 3.8322272300720215
```

Your program may take as command-line arguments the number of pivots (numpivots) and the number of nearest neighbors k.

Step 5: Evaluation

Conduct some experiments that assess the effectiveness of the methods that you developed.

For different values of ε (ε = 0.1, 0.2, 0.4, 0.8), compare the three range query methods (Step 3) and draw two diagrams where the x-axis in both diagrams is ε. In the first diagram, measure and show the total time for each method (as a lines plot). In the second diagram, the y-axis is the average number of distance computations per query.

For different values of k (k = 1, 5, 10, 50, 100), compare the three kNN query methods (Step 4) and draw two diagrams where the x-axis in both diagrams is k. In the first diagram, measure and show the total time for each method (as a lines plot). In the second diagram, the y-axis is the average number of distance computations per query.

Include the diagrams in your report and write your observations about the results.

Submission requirements

You should submit your program(s). You should also write a short report about your implementation, in case it is hard for us to understand your program. Your report should include the output of your program. Your report should also include your comments about the evaluation and any other observations that you have. Submit the report together with your code at the course website. Your code should be compilable without problems and you should include basic instructions on how to compile and use it. Your program can be written in your preferred programming language (e.g., C, C++, Java, Python, etc.). Your program must run within reasonable time.

Please submit your assignment (one **ZIP** file) to Moodle on or before **Oct 31st, 2022, 5:00pm**. Make sure all contents are readable.

Please feel free to post your questions on **Moodle forum** or contact TA Shoufa CHEN (shoufach@connect.hku.hk) if you encounter any difficulty in this assignment. We would be happy to help.