ACM Summer School 2023

Course: Spatial and multi-dimensional indexing and data analytics

Assignment

Goal: Develop and test a fundamental similarity search technique for dense multidimensional vectors

You are going to use synthetically generated data. The data file is data10K10.txt. 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 identifier of the object is implied by the line number (starting from 0). For example, the 10 numbers at the first line are the coordinates of the 10-dimensional object with identifier 0.

You are going to implement two 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 class. You are going to use **Euclidean distance** (L_2) as the distance measure dist().

$$L_2(p,q) \equiv \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Write a program and save it in the same folder as the data of this assignment. Your program should take as command-line arguments:

a) The number of pivots (numpivots), int
b) A distance bound ε (epsilon), float
c) The desired number of nearest neighbors (k), int

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. For example, you may use a 2D array D, such that D[0] is the array that stores the vector with identifier 0. 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)+… +dist(p_{k-2},o), where {p_0, p_1, … p_{k-2}} are the k-1 first pivots.

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

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

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

Step 2: 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*: 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.

After running the function that computes the pivots, test the effectiveness and efficiency of the above methods, using 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
total time Naive = 4.390295028686523
total time Pivot-based = 1.146554946899414
```

Recall that your program should take as command-line arguments the number of pivots (numpivots) and the distance bound ε (epsilon).

Step 3: 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 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. For the first k objects in the data array D, compute their distances to q, put them in a data structure H, and use the distance of the furthest object to q as a bound ε. For each subsequent object o, use 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 update the current set of kNN objects and ε. $dist(p_i,o)$ can be accessed from the 2D array of Step 1 and $dist(p_i,q)$ should be computed once for each pivot p_i.

After running the function that computes the pivots, test the effectiveness and efficiency of the above methods, using 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 (Naive) = 10000
average distance comp per query (Pivot-based kNN) = 3156.76
total time Naive kNN = 4.891640663146973
total time Pivot-based kNN = 4.830014228820801
```

Recall that your program should take as command-line arguments the number of pivots (numpivots) and the number of nearest neighbors k.