PROJECT-ASSIGNMENT2-README

sdi1800185 Giwrgos Stamatelis
sdi1600222 Violetta Gkika

December 2021

Contents

1 How to Run

1.1 Search

Inside folder TimeSeries/Search run make then as described in the project presentation.
/.search parameters

1.2 Clustering

Inside folder TimeSeries/Clustering run make then as described in the project presentation.
/.cluster parameters

1.3 Unit Testing

To perform unit testing we used CUnit library. So, in order to run test that part of the project one should have downloaded CUnit and know the path to reach its files.
unitTest.cpp contains the path for the Basic.h file of Cunit(line 4). This should be rapalced with the path of your CUnit folder.
After downloading and changing the path one can run the program as follows:
Inside the folder UnitTest you have to run 'make' and then ./UT.
The program will ask you for a path of data for two of the the three testing unit.(will be explained later)
In both cases you have to provide a file path that contains curves in the format of given Datasets. You can give either nasd-input.csv or nasd-queries.csv.

2 Folder Structure

2.1 Common

Here,inside common.cpp/hpp and input.cpp/hpp are some utility functions used for the implementation of all the sections of the project. Like reading inputs, calculating euclidean distance, brute force algorithm etc.

2.2 Data

Stores the given data sets files.

2.3 Fred

Stores the given Fred library.

2.4 NN

Files from Project1.

2.5 TimeSeries

Files for this project.
Clustering
Files that implements Clustering.
Search
Files that implements KNN search.

2.6 Output

some demo output files for nn and clustering (only for curves)

3 AI

3.1 Implementation

This is pretty much the same as assignment one. The curves are treated as vectors. We load the input file in the LSH or the hypercube data structures with the parameters defined by the user. The Data structure uses the L2 distance to find the ANN, we also search the nn. We briefly convert both the approximate and the true nn to curves and then compare their distances to the query, compute the MAF e.t.c.

3.2 Performance

For LSH queries try with L=5 and k=3. As you can see the MAF is very small and the LSH query is much faster than the brute force query.
For Hypercube queries try probes=2 k=3 and M=500. It does not perform as well as LSH in terms of MAF, but it still pretty good and very fast.

4 AII

4.1 implementation

First of all we create the grids and the labels with the parameters defined by the user. It is important to mention that for LSH tables L=1. The L given by the user represents the number of grids. For each of the grids, we snap all the curves to points (using the formula from lectures/eclass discussions) and store the result in a data structure pointsPerGrid. Then we store those points to an LSH table that uses discrete frechet distance.
Each point has a pointer to a discrete2dcurve. getFrechet(point a,point b) uses those pointer to call getFrechet(discrete2dcurve a,b) (operator overloading).

We must not fail to mention that LSH both A2 and A3 can fail and not find a near neighbor (it was discussed in eclass and Dr Chamodrakas said it is possible). In that case, we use the points returned by brute force. The time taken for the query is time taken to check the buckets and find no adjacent neighbors plus the time the brute force algorithm requires.

4.2 performance

Try running the program with L=6, k=2 and delta=0.1 . The MAF is very very small and the queries are faster than brute force.

5 AIII

5.1 Fred

To be able to include the Fred library we cloned the files from github and commented out all the python sections.
Created a function to convert our structures to the one used by Frechet distance function (frechet.cpp - distance(Curve,Curve)). Then used this function to calculate continuous Frechet distance every time we needed it.

5.2 Implementation

To implement the function of this section we read the data from the input file, perform filtering ,snapping and padding and store the new vectors in the lsh structure. Inside the lsh the items in the bucket store both the vectors and the initial curves. The vectors are used for storing and the curves to calculate the distance. Then after reading queries, for each query we perform the same preprocess(filtering, snapping, padding) , calculate the amplification function to find the corresponding bucket and then calculate the distance of the query with each item with same IDs in the same bucket(always using the initial curves). The nearest neighbor is returned and the time needed to find it is calculated. In the output file everything is printed as required.

5.3 Performance

Testing different parameters, we concluded that some good parameters to run this section are L=4 , k=2, delta=0.5. We have set w=40 and e=0.01 The table size is $curves/$.
From different trials we observed that the IDs trick even though makes LSH fail to find a neighbor for some buckets it saves a lot of time and make LSH perform better.

6 B

Apart from the mean curve,there is not much to talk about. It is the same as project one, only when assignment=Mean Frechet, every distance is replaced by the continuous frechet distance and during the update phase of k-means, we use the mean curve function.

6.1 Mean curve

To calculate the mean of N curves we followed the implementation of the presentations from the lecture. We created a tree as an array with the appropriate pointers to the left/right child.
To implement the tree we created struct TreeNode that stores the pointer to the left and right child, a bool isLeaf that indicates if the node is leaf or not, the curve (if the node is leaf will store a given curve , if not will store the mean of two curves of its left and right child curves).

There are three functions to implement the tree. One that initializes the tree and scatter randomly the curves in the leaf, one that calculates the mean of two curves and one that implements the postOrderTraversal function as shown in the lectures.

6.2 performance

We will discuss the performance of kmeans for 2d curves.
We conducted most of our experiments on *nadaq_input.csv*. First we experimented with small k=2. We noticed that while the average silhouette was very high >0.5, the silhouette of the second cluster was very small. Also, the size of the clusters was pretty imbalanced. We experimented with k=4,5,6,10 and 15. We got the best performance with k=5. For k close to 5 silhouette is pretty good as well. For the best performing k(=5) we wanted to see if using LSH and range search can speed up the process without ruining the silhouette. It turns out that with approximately the same average and individuals silhouettes, it required 52 seconds less (on the same laptop).

7 Unit testing

Unit Testing is performed in three different functions of the program.

snap-pad() : the function that takes some curves and performs snapping for a given grid and then padding for some value(730 in our case).
To test this function the program will ask for a file path that contains some "curves". The program will read the file, convert the lines into our data structures that correspond to curves and will perform snapping and padding.
CU-ASSERT function will check if the size of a curve after snapping and padding corresponds to the value of padding(730).

meanOfTwoCurves(): the function takes two curves and calculates and returns the mean of those curves. To test this function the program will ask again for a file path that contains some "curves". Will read the file, convert the lines in the corresponding data structures. The program will call the meanOfTwoCurves for the first two curves of the file. We calculated by hand the mean of those two curves for the first 10 points.
CU-ASSERT function will compare our calculation with the one returned by the function.

getFrechet(): this function takes two 2Dimensional curves and returns their Frechet distance. To test this function, we created two "dummy" curves with three points in the same way as the program creates 2D curves from input and queries files. Calculated by hand the Frechet distance they should return and compared with the one that getFrechet() returned in the CU-ASSERT function.