



ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών
———ΙΔΡΥΘΕΝ ΤΟ 1837———

SOFTWARE DEVELOPMENT FOR ALGORITHMIC PROBLEMS. 7TH SEMESTER 2020, ASSIGNMENT 1: SEARCH AND CLUSTERING VECTORS

National and Kapodistrian University of Athens, DiT.

30/10/2020

Dimitrios Foteinos | sdi1700181

Evgenia Kalyva | sdi1600256

Contents

1	Abstract	2
2	File Contents	3
3	Compile and Run	3

1 Abstract

This assignment is the 1st part of the set. The subject of this particular homework, belongs purely in the field of Data Science. In this assignment, we were asked to implement some interesting methods for **searching** and **clustering** d -dimension vectors.

For this project, we were confidential about the tools that we may have needed. Ofcourse, this project is implemented in **C++**, because we needed a lot of the structures, utilities and the generic nature of this specific language. Thus, our design was made up in order to have any type of vectors, (such as **int** or **double**). The metrics for the distance that we were asked to choose is the famous **Manhattan Distance** and **Hamming Distance**. To be more detailed, this project was splitted up in two unique parts: A) **Searching** and B) **Clustering**. For each part, we have the following:

A. Locality Sensitive Hashing and Randomized Projections.

The algorithms that we implemented for Searching, was the famous **LSH** and **HyperCube**.

For LSH, we used as distance function the Manhattan Distance. The main idea is that we have some **training data** and some **test data**. First, we choose the number of the Hashtables we want to use. Then, for each Hashtable, we choose from some specific formulas the number of buckets that we want to use. With these being said, we must declare that in this program, there is a hyperparameter: **w**. Stuff can be really tricky about the choosing of **w**, because it can be shown that this parameter can decide crucial things about our program, such as **Accuracy Complexity** and **Time**. As we expected, if we want to increase the accuracy, then the time will be increased too. So, how can we choose the best **w**, in order to achieve the best trade-off between Accuracy and Time?

There is not only one way to answer this question, because some may want to achieve the best accuracy regardless of the time (in this case the computational power may be higher than usual), and some may want to achieve the best possible accuracy with the minimum possible time. Let's assume that we agree with the second case, how can we achieve the best accuracy with the minimum time? In order to accomplish this behaviour, we created a cost function:

$$C(t, d) = t * d$$

Given inputs are: t = the ratio of the average time of LSH and the average time of Bruteforce

$$t = \frac{mean_{LSH}}{mean_{TRUE}}$$

and d = the absolute value of the difference between the average distance of LSH and the average distance of Bruteforce,

$$d = |LSH - TRUE|$$

So , the lower the cost function, the best for us. We did some "grid search" in our data. We used whole the training set (60.000 points) and we took a tiny sample of the test data (100 points). For **LSH** we have the following:

Cost Value/W	52	100	250	500	1000	4.000	20.000	40.000
LSH	61.0441	63.7806	62.7264	65.6498	57.0376	56.8202	19.5274	37.4881

As we can see, the best possible w is 20.000 after our research. We must declare that, if we set w up to 20.000, we may increase the time, but in this value we have the best possible accuracy. So, if someone wants to increase the time of the algorithm but still achieve some satisfying results, it's up to him to play with the hyperparameter and see which one fits perfect for him.

2 File Contents

3 Compile and Run