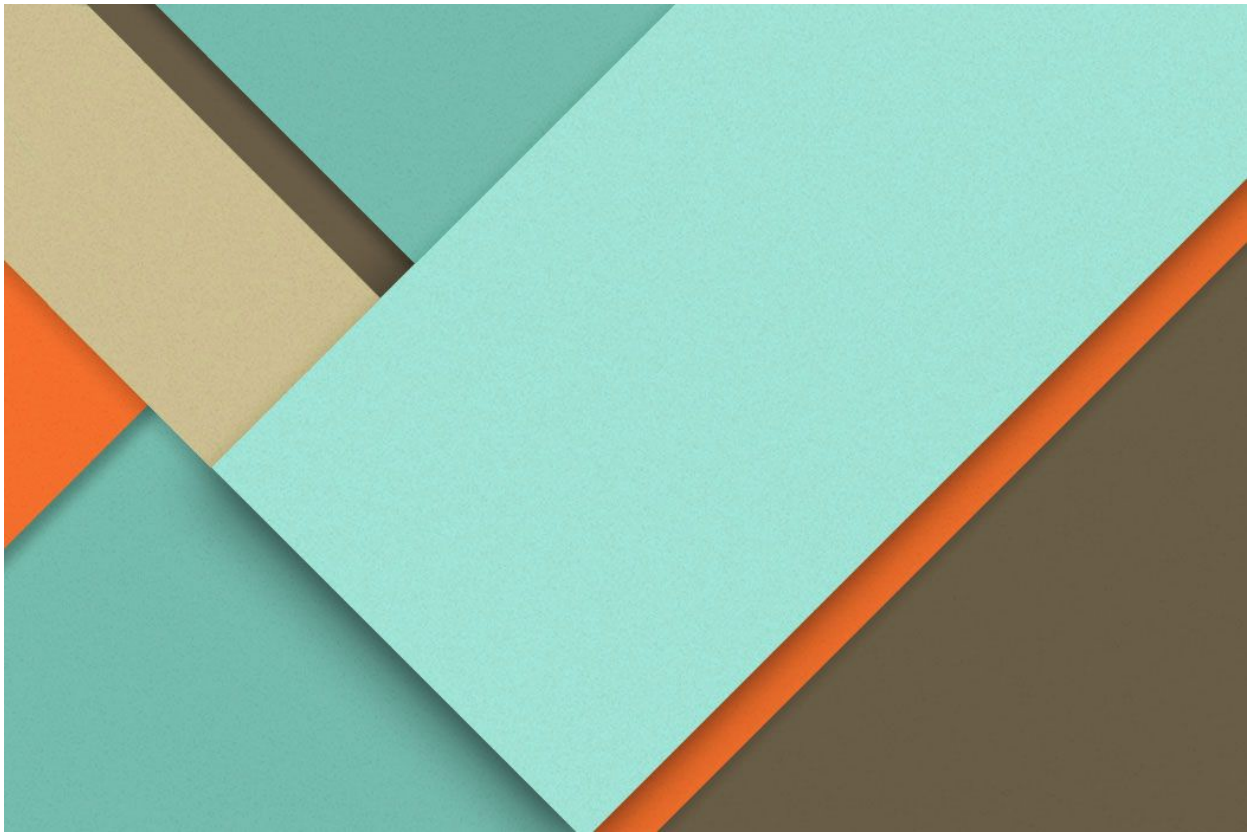

Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα Εργασία 2

Ονοματεπώνυμο: Μπαρμπάρ Ηλίας-Ελιάς
ΑΜ: 1115201200118

Summary

This program implements the Kmeans and Kmedoid clustering methods, with two distinct initialization modes (random , kmeans++), three assignment modes (Lloyd's, Range Search with LSH, Range Search with Hypercube) and two Update modes (Kmeans, PAM).
It is given the vectors in a .cvs file and the parameters in a .conf file. We also save the clusters, the time for each run and the Silhouette as a .txt file.

Files

mainKmeans.cpp:

We read the cmd arguments, input and configuration file, then we run our clustering algorithm based on our cmd arguments and finally we make our outputfile.

readCmd.cpp/.hpp:

-Reads the command line argument using the getopt function of GNU.

readFileConf.cpp/.hpp:

-Reads the configure file and sets parameters.

readFileCsv.cpp/.hpp:

-Reads the training set from the csv file we were given.

readFileCsvV2.cpp/.hpp:

-Reads the training set from the csv file we were given and only keeps distinct vectors.

kMeans.cpp/.hpp:

-We have our one class myKmeans which implemets our two initializations, our three assignment methods and our two updates.

-myKmeans consists of:

-vector<myVector> vectors: a pointer to our train set

-vector<myVector *> centroids: the centroids that will be made for the update functions

-vector<int> clustAssigned: is simply a static array that keeps the assignment per vector

-double clustSi: keeps the Silhouette's S(i) mean per cluster

-int * vCountPerCluster: as the name suggests keeps the vector count per cluster

-int kClusters: is the sum of clusters

-int vDimensions: is the deminsion of each vector of doubles our vectors have.

-string metric: is the metric being used {euclidean, cosine} (for the time being)

- double finalSi: keeps the value of Silhouette

- double duration: keeps the duration of the algorithm running in seconds.

- bool comments: dubs in comments (only used on update_PAM() function as a test.

Functions:

- runKmeans: Gets three inputs that relate to which methods of initializing, assigning and updating we are going to use the proceeds to first initialize once, assign and update for a (static int) given amount of times which is defined in our kmeans.hpp as runKmeansXtimes. If we choose to assign using lsh then runLSH will be called and if we choose to assign using HC runHC will be called.
- runLSH/runHC: The chosen conditions to terminate are 1) We've done at least 5 runs and 95% of our points have been assigned. We call the assignment_LSH function until one of our conditions is met.
- runUpdate: Simply runs one of our two available update options depending on the input.
- initialize_RandomClusters: Initializes centroids by making sure not two same vectors are chosen.
- initialize_Kmeancpp: Uses the kmeans++ algorithm to initialize the centroids.
- findDistanceOfClosestCentroid: Simply finds the distance to the closest centroid in an exhaustive way.
- assignment_Lloyds: Assigns vectors to centroids in an exhaustive way.
- assignment_LSH: For each centroid we get a list of vectors from range search with which we assign them to that centroid
- assignment_HC: Same as above.
- hclsh_FindRadius: We parse the centroids in a factorial fashion to find the minimum distance,

- complementaryLloydsAssignment: Uses the flagAssigns bool array to assign points that were not reached by range search to their centroid using Lloyd's algorithm.
- findLocationOfClosestCentroid: Same as finding distance but we keep the location as opposed to keep the distance.
- update_kmeans: We set the centroids to 0 then we find the sum of vectors assigned to each cluster while we add those vectors to that centroid and at the end divide with the sum of vectors in each cluster.
- setupKmeans: We make sure that we have allocated memory for the new vectors that are going to be made by update_kmeans.
- update_PAM: We find the medoid for each cluster and then se the new centroids
- Silhouette: We firstly find the two closest centroids for each vector and then we proceed with the calculations of a(i) and b(i) to find the s(i) per vector then we find the clustS(i) for each cluster and then we add them and divide by the number of clusters to get the finalS(i).
- getDistance: Calls what distance function we are going to use depending on the metric.
- makeOutputFile: Makes the output file with the instructions we were given.

Instructions

We use make to compile our program. Then the main parameters are our static int runKmeansXtimes which speaks for itself and is defined in the file: "kMeans.hpp".

A call for our program should look like this:

./cluster -i <input file> -c <configuration file> -o <output file> -d <metric> -n <init opt> -a <assign opt> -u <update opt>
We keep our tests and comparison in the folder named "tests".

Compilation

We compile using a makefile so all the user has to do is type 'make' which has the following form:

KM:

g++ mainKmeans.cpp kmeans.cpp LSH.cpp HyperCube.cpp cosineH.cpp euclideanH.cpp gfunction.cpp readCmd.cpp readFileConf.cpp readFileCsv.cpp readFileCsvV2.cpp -o KM

.PHONY: KM