20T1-COMP9318-PROJECT-REPORT

Hao ZHONG (z5195835)
Chang Wang (z5196324)

Task 1

First of all, we need to split the input data into P size. We calculated the length for each size, due to P is always divides M in this project, we considered a while loop to keep obtaining a fragment data with length M/P and append this new fragment data in a new list. Thus the new data shape is (P, N, M/P) now. For each fragment of data and centroids, through the package 'scipy.spatial' we could calculate L1 distance for each fragment.

Considered one of them, through the argmin() function of numpy package we could find the index of the minimum value for each point of data, where the value is the smallest distance to all centroids. Then we updated the cluster of each center point. The next step is to update the center point if the cluster changed. Repeating these steps for max_iter times. We also considered if the center points has no changes comparing with the last iteration, we could end the loop before it has iterated for max_iter times, and the time cost decreased. Repeating this for all of data, where it has split into P size, we could obtain the result for codebooks with shape(P, K, M/P).

Similar with the method we just mentioned, to calculate the L1 distance between data and codebooks, we obtained a (N, K) array. We also used argmin() function to obtain the index of center point, and find the index of this center point in codebooks, it is the output for each point in the codes. Therefore we could obtain the index in the codebooks for all points, and it is the result of codes with shape(N, P).

Task 2

Input: queries, codebooks, codes, T
Queries: shape (Q, N)
Q: dimensionality of one query, N: the amount of query

Codebooks: shape (P, K, M/P)
P: the number of partitions the vector will be split into
K: the amount of centroids
M/P: the dimensionality in each partition

Codes: shape (N, P)
N: the amount of initial data point
P: the number of partitions the vector will be split into

Step 1: spliting queries

In this part, each query will be split into P partitions. For example,
Input queries is [[1,1,1,1,1,1], [2,2,2,2,2,2]], and P = 2
then after running the part,
the queries is [[[1,1,1], [1,1,1]], [[2,2,2], [2,2,2]]].

Step 2: calculating the distance between queries and codebooks

In this part, we will use L1 distance(cityblock) to calculate the distance. At first, we will get one query from queries, then each partition of this query will be calculated with codebooks. We can use distance function in the scipy.

scipy.spatial.distance.cdist([one_query[i]], codebooks[i], metric='cityblock')

After finishing this part, we will get a distance set.

Step 3: sorting distance

For getting sequence number order by distance, we use np.argsort(). In this part, we will get a list that contains the index of distance set and sort by distance.

Step 4: codes dictionary

We use a dictionary to match each code and its index in codes. For example, in toy_example, we get a codes [[0 0], [0 0], [0 0],[1 1] ,[1 1] ,[1 1]…]. Then in codes dictionary, it will be change to codes_dict[(0 0)]:[0,1,2], the key is a tuple and the value is its index in codes. The reason why we use dictionary in python is that, comparing with list, the key in dictionary can be hashed, so it would be helpful to improve our search time.

Step5 : multi-sequence algorithm

In this part, we implement a loop for all the query. In the loop we use priority queue to save the vector that we already find. Moreover, a dictionary would be used to save the traversed vector which is helpful to check whether the vector is traversed. If a vector is traversed, its tuple will be insert into this dictionary. We use a list to store the output (the codes index). At first, we will push the vector that have the minimal distance sum. Then we will go into a while loop, which will run until the output is large than T. at the start of this loop, we will pop a tem_index which have the minimal distance among traversed vector from priority queue, the each dimension of tem_index must less than the amount of initial data point. after then, we will do a judgement that whether the children of this tem_index is traversed, if not the children will be push into priority queue. And also insert it into the traversed dictionary.

**********END**********