Report For Advance Task

[A2] Not Finished Take a look at the paper "Continual General Chunking Problem and SyncMap". Download the code mentioned in the arxiv version. Your task is to improve more than 5% in both Overlap1 and Overlap2. Explain the results.

Introduction

I only change the learning rate and remove the DBSCAN from SyncMap using a self made function instead to determine the cluster in this task. These two modifies make the SynMap run better on Overlap2 comparing to the original SyncMap, however it performs equally to the orginial on Overlap1, both my modified version of the SynMap and the orginial version can not have a very good performance on Overlap1, the accuracy cannot exceed $75%$, and my modified version is not stable. So I don't think I complete this task。 Due to the lack of knowledge, I cannot find a better solutation towards this question.

My modification

In this part, I will only introduce the change of learning rate, the replacement of DBSCAN will be introduced in the next task. Although, the author mentioned in the paper that "These parameters were the best performing without unnecessary slowdown after a dozen trials.", I still think that maybe a learning rate which decrease with the number of epoch trained, will get a better performance. So I first double the initial learning rate, because I will decrease the learning rate during the learning, it may cause the SyncMap to take more step to reach the best solution if I don't increase the initial learning rate. In the first half of the learning loops I keep the learning rate the same as the initial, and the other half of the learning loops I lineally decrease the learning rate from initial to $1/2$ initial because when it closer to the result I want the learning rate to be smaller in order to get a more accuracy result.

[A3]{X} Remove DBSCAN from SyncMap. Use edges and weights to determine clusters. Your results is to both decrease the variance from SyncMap and increase accuracy in most of the tasks.

Introduction

In this task I replace the DBSCAN with a method based on the edges and weights to determin clusters. The method is inspired by both the DBSCAN and the Spectral Clustering.

DBSCAN

The DBSCAN algorithm, which stands for "Density-Based Spatial Clustering of Applications with Noise," is a clustering algorithm designed to identify groups of data points in a set based on their density. Developed in 1996 by Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu, it is particularly effective for datasets with arbitrary shapes and densities.

The algorithm requires two parameters:

> Epsilon (eps): This parameter defines the maximum distance between two samples for them to be considered in the same neighborhood. MinPts: The number of samples in a neighborhood for a point to be considered a core point.

It is one of the two key factors that dictate the behavior of the algorithm. In general, larger values of MinPts result in fewer noise points but can lead to data points from different clusters being assigned to the same cluster, while smaller values of MinPts can result in more noise points but might separate clusters better.

The DBSCAN algorithm defines three kinds of points:

> Core Points: For each data point, a sphere of radius eps is defined around it. If there are more than or equal to MinPts points within this sphere (including the point itself), the data point is considered a "core point."
>
> Border Points: For each data point, a sphere of radius eps is defined around it. If there are less than MinPts points within this sphere (including the point itself) and has at least one Core point in this sphere, the data point is considered a "border point."
>
> Noise Points: points are neither border point nor core point

The four relationships defined by DBSCAN:

> Directly Reachable: Any point within the eps distance of a core point is considered "directly reachable" from that core point.
>
> Density-Reachable: If there is a chain of points, each within eps distance of the next one, and all of the points are core points, then every point on the chain is "density-reachable" from every other point.
>
> Connected: If there is a point n that density-reachable both point x and y, then we call the points x and y are "Connected".
>
> Unconnected: If there is not any point n that density-reachable both point x and y, then we call the points x and y are "Disonnected".

The DBSCAN algorithm steps are divided into two phases.

> Phase 1: Finding Core Points to Form Temporary Clusters: Scan all sample points. If the number of points within the radius R of a certain sample point is >= MinPoints, include it in the list of core points, and form a corresponding temporary cluster with the points density-reachable from it.
>
> Phase 2: Merging Temporary Clusters to Form Clusters: For each temporary cluster, check if its points are core points. If they are, merge the temporary cluster corresponding to that point with the current temporary cluster to obtain a new temporary cluster.
>
> Repeat this process until each point in the current temporary cluster is either not in the core points list or all its density-reachable points are already in that temporary cluster, upgrading the temporary cluster to a cluster.
>
> Continue merging the remaining temporary clusters in the same way until all temporary clusters are processed.

Spectral Clustering

Spectral Clustering (SC), is a graph-based clustering method that can identify arbitrarily shaped sample spaces and converge to the global optimum. Its basic idea is to use the eigenvectors obtained by decomposing the similarity matrix of the sample data for clustering. It is independent of sample features and only depends on the number of samples.

The basic idea is to regard the samples as vertices and the similarity between samples as weighted edges, thus transforming the clustering problem into a graph partitioning problem: finding a graph partitioning method that minimizes the weight of edges connecting different groups (meaning that inter-group similarity is as low as possible), while maximizing the weight of edges within a group (meaning that intra-group similarity is as high as possible).

My method

Inspired by the DBSCAN and the Spectral Clustering, I developed a alogrism which doesn't need to determine the number of clusters manully like DBSCAN and works like the Spectral Clustering. My method can be divided into 4 steps:

> Step 1. Calculate the distance between each given points: I use the torch.norm(a[:, None]-a, dim=2, p=2) function to calculate the distance between each points. This function also translate the point list into a distance map. In my function I use the Euclidean distance, which can be replace by other kind of distance function which is not linear. Step 2. Calculate the shortest distance of each pair of points: I used a Floyd Warshall algorithm to calculate this. This step is similar to the phase 2 in DBSCAN, which enrise the size of the original cluster. This step is designed to deal with the situation which doesn't use Euclidean distance. Step 3: Discard the distance which is longer than max bearable distance: I simply set those pairs of points unreachable. In this way we seperate the clusters apart. The points in the same cluster have less distance to each other than the points belongs to other cluster. Step 4: Find all the connected components in the graph:I use the DFS to do this job. this job will generate the label for each point.

Conclusion

My method is simple than the original DBSCAN and can perform better on overlaping test, but it has a less accuracy when dealing with the high dimention test. May be using other distance function other than Euclidean distance will improve the performance.

Code and result

Code:https://github.com/NatsukiKoki/kyusyuUnv.git