Trajectory Clustering by Statistical Approach

ZHANG Zhexiang

PG Student
Hong Kong University of Science and Technology

November 25, 2024

Table of Contents

Trajectory Clustering

In class, we cluster points. And today, we will focus on clustering trajectories.
What is trajectory? Trajectory consists of a list of points with time stamps, distinguishing it from a single data point. Trajectory data is common in various fields, such as typhoon trajectories analysis, and human movement analysis. Clustering these datasets can be useful. For example, it can reveal distinct walking patterns or insights into weather phenomena[2].

Trajectory Clustering

Intuitively, clustering requires to define a function that measure the distance between targets. Just like clustering points, we should determine the distance between points. For trajectories, so we also need a "similarity" function to evaluate the similarity between trajectories. Intuitively, the defined similarity function should return large value between similar trajectories.

Trajectory Clustering

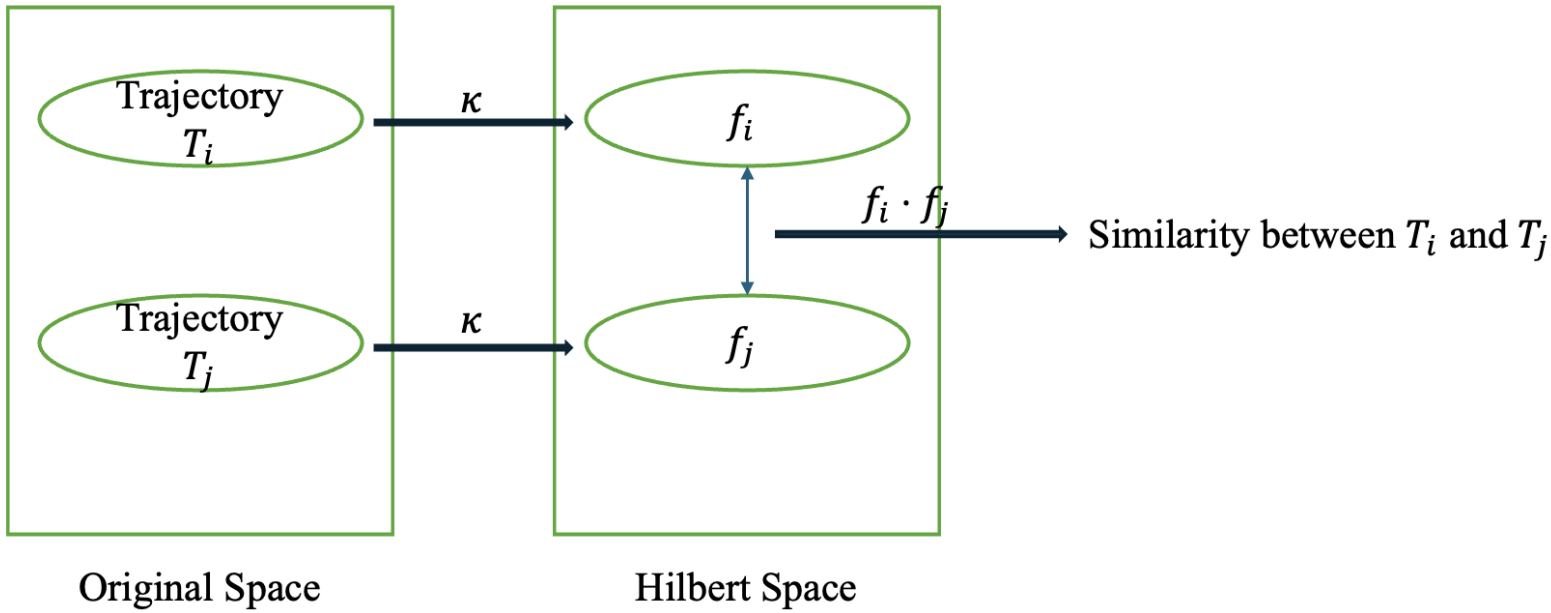
In the paper **Distribution-Based Trajectory Clustering** [4], it provides the metric we want. It views trajectories as probability distributions. The trajectory, i.e., a list of points $T = \{p_1, \ldots, p_n\}$ can be considered independent and identically distributed from an unknown distribution P_T. We can use of metric that provides the similarity between distributions to find the similarity between the trajectories.

Question

How to find the distance between distributions?

KME

A kernel function $\kappa(\cdot, \cdot)$ will be employed. It can map the **Trajectory** T_i(to be viewed as points sampled from some distribution) in the original space to a **point** f_i in Hilbert space H. A measure of the similarity between the two trajectories(distributions) can be obtained by calculating the dot product of two points $f_i \cdot f_j$ in Hilbert space.



KME

Today we do not need to explore the details of the kernel function; we simply regrad κ as the TOOL to map the data points to Hilbert space.

KME

The dot product (i.e. A measure of similarity between trajectories) can be calculated as following formula using kernel mean embedding[3]:

$$f_i \cdot f_j = \frac{1}{|P_{T_i}||P_{T_j}|} \sum_{p_i \in T_i} \sum_{p_j \in T_j} \kappa(p_i, p_j)$$

Basically, by averaging the kernel values over all pairs of points from the two trajectories, we obtain a measure that reflects how similar or related the two trajectories are to each other. And different kernel functions κ will have different performance.
For example, we can use Gaussian kernel $\kappa(x, y) = \exp(\lambda \|x - y\|_2)$. But the performance is bad[3], i.e., cannot measure the "similarity" properly.

Table of Contents

Isolation Dependency Kernel

The paper[3] proposed a kernel called Isolation Dependency Kernel (IDK). The core idea of the isolated kernel function is to divide the data space into several cells. Then the kernel value $\kappa(x, y)$ of data points x and y is defined as the probability that these two points are in the same cell:

$$\kappa(x, y) = \mathbb{E}[I(x, y \mid h \in H)] \approx \frac{1}{t} \sum_{i=1}^{t} I(x, y|h_i)$$

Here, H contains some methods that partition the space. I is the indicator function. If x and y are in the same cell under the partition method h, the function returns 1; otherwise, it returns 0. Since κ does not have a closed form, we can employ sampling techniques to estimate it.

IDK

For example, if we use Voronoi diagrams for partitioning (i.e. $H = \{\text{All partition methods using Voronoi diagrams}\}$). Suppose we have 20 points, and each time we divide the space into 5 cells. We attempt this 3 times (i.e. $t = 3$):

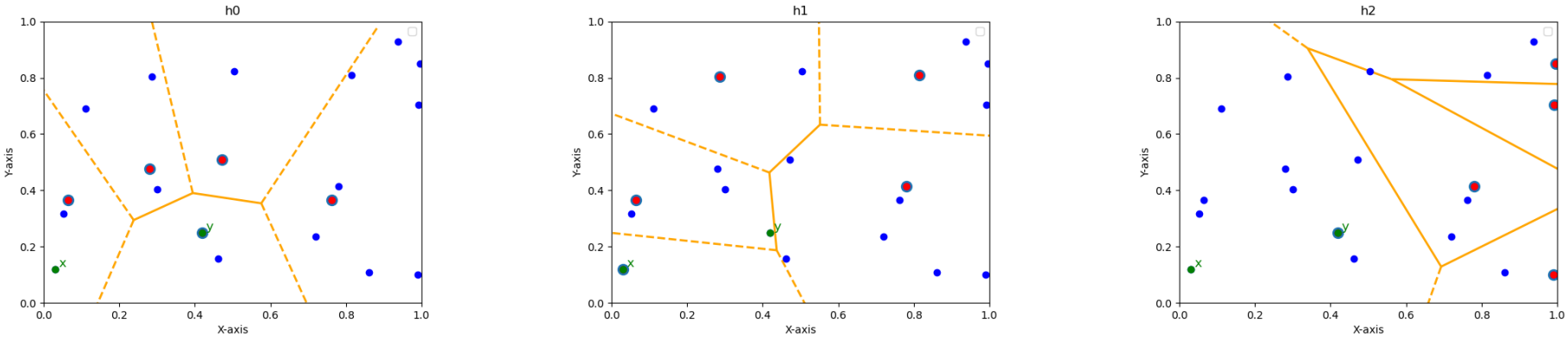


Figure: Voronoi diagrams h_0, h_1, h_2

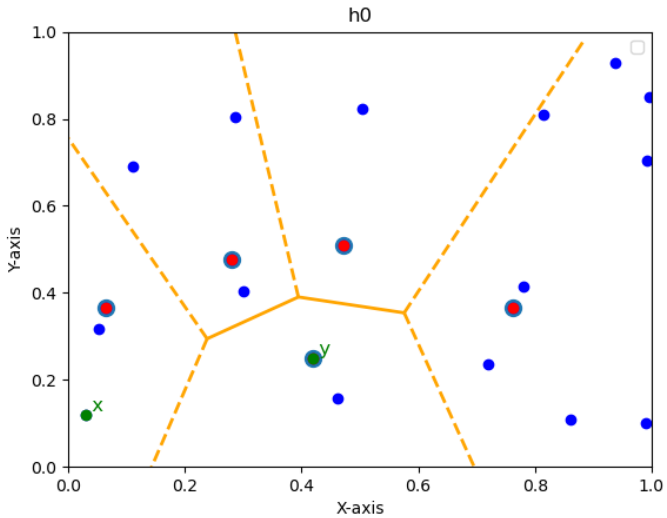
In this context, let us consider the function $\kappa(x, y)$:

$$\kappa(x,y) = \mathbb{E}[I(x,y \mid h \in H)] \approx \frac{I(x,y|h_0) + I(x,y|h_1) + I(x,y|h_2)}{3} = \frac{1}{3}$$

If we sample it t times, where t is large, we will obtain a reliable result.

IDK

To make the calculation easier, we can utilize a vector in place of the indicator function I during the partition h_i. For instance, in the figure below, we define a vector to represent the position of a point: if the point is located in cell i, then the i-th element of the vector is set to 1; otherwise, it is set to 0. In this case, I can be represented as the dot product of two vectors.



$$v_{x,h_0} = (1,0,0,0,0), v_{y,h_0} = (0,0,0,0,1), I(x,y|h_0) = v_{x,h_0} \cdot v_{y,h_0}$$

IDK

$$v_{x,h_0} = (1,0,0,0,0), v_{y,h_0} = (0,0,0,0,1), I(x,y|h_0) = v_{x,h_0} \cdot v_{y,h_0}$$

$$v_{x,h_1} = (1,0,0,0,0), v_{y,h_1} = (0,1,0,0,0), I(x,y|h_1) = v_{x,h_1} \cdot v_{y,h_1}$$

$$v_{x,h_2} = (1,0,0,0,0), v_{y,h_2} = (1,0,0,0,0), I(x,y|h_2) = v_{x,h_2} \cdot v_{y,h_2}$$

Similar operations can be applied. We combine these vectors respectively to obtain v_x and v_y.

$$v_x = (v_{x,h_0}, v_{x,h_1}, v_{x,h_2}) = (1,0,0,0,0,1,0,0,0,0,1,0,0,0,0)$$

$$v_y = (v_{y,h_0}, v_{y,h_1}, v_{y,h_2}) = (0,0,0,0,1,0,1,0,0,0,1,0,0,0,0)$$

$$\kappa(x,y) \approx \frac{v_x \cdot v_y}{3}$$

IDK

$$\kappa(x, y) \approx \frac{v_x \cdot v_y}{3} = \Phi(x) \cdot \Phi(y)$$

IDK can be expressed as the product of a function of x (i.e. $\Phi(x)$)and a function of y (i.e. $\Phi(y)$), which is a nontrivial property.
Not all kernels have this property. For example, there do not exist the function Φ such that:

$$\kappa(x, y) = (x - y)^2 \neq \Phi(x)\Phi(y).$$

For IDK, Φ is called feature map[1].It can be considered to be the function that maps points to higher dimensional representations.

IDK

Similar operation can be performed for any points. Suppose we have N points. We can sample t times, each time we partition the space into ϕ cells. Then we can obtain the feature map $\Phi(x)$ for each point x.
Then the kernel can be calculated using:

$$\kappa(x, y) \approx \frac{1}{t} \sum_{i=1}^{t} I(x, y | h_i) = \Phi(x) \cdot \Phi(y)$$

Distance Between Trajectories

Recall the dot product we want to calculate[3]:

$$f_i \cdot f_j = \frac{1}{|S||T|} \sum_{s \in S} \sum_{t \in T} \kappa(s, t)$$

For IDK, we can write $\kappa(s, t) = \Phi(s)\Phi(t)$, then we obtain:

$$f_i \cdot f_j = \left(\frac{\sum_{s \in S} \Phi(s)}{|S|}\right) \cdot \left(\frac{\sum_{t \in T} \Phi(t)}{|T|}\right)$$

Basically, the dot product is computed as the product of the average values of the function Φ over S and T.

Distance Between Trajectories

This means that if we have some trajectories, $T_1, \ldots, T_m$, with a total of N points. We can calculate the average value of the Φ function within each trajectory first. That is, we can define the feature map for the trajectory T_i:

$$\Phi(T_i) = \frac{1}{|T_i|} \sum_{x \in T_i} \Phi(x)$$

It sums the values of $\Phi(x)$ for each element x in T_i and divides by the number of elements. It can be considered to be the function that maps a trajectorys to higher dimensional representation. Based on this, the dot product $f_i \cdot f_j$ can be directly calculated:

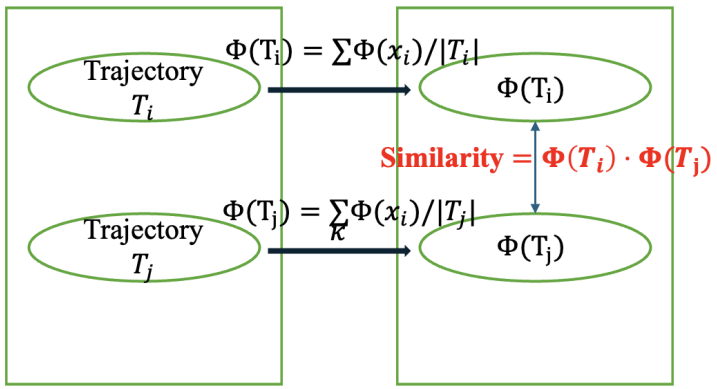


Table of Contents

IDK2

We are able to calculate the dot product between any two trajectories, it seems that clustering can be implemented immediately.
However, the paper[3] points out that using a two-level IDK would result in higher performance. This may be confusing: what is a two-level IDK?

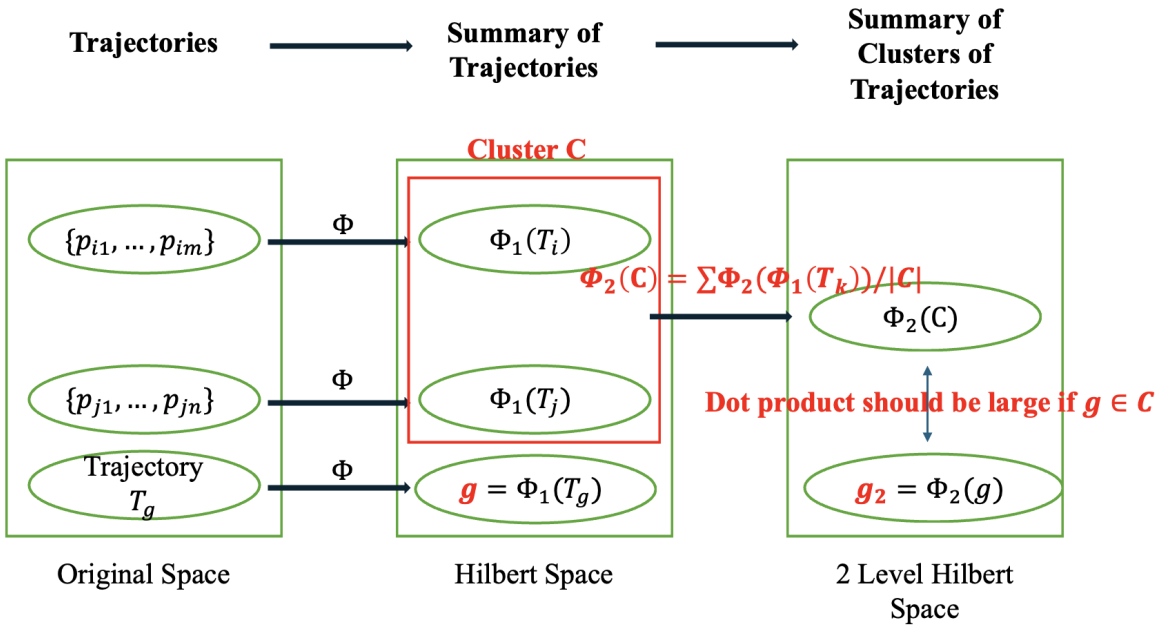
IDK2

Starting from the point anomaly detector, the paper[3] points out that if we have a set C composed of data points, and a small portion of them are anomalous, to find these anomalous points, we need to first calculate $\Phi(C)$ of these data points.
The anomalous data point g is considered to be point with smaller dot product $\Phi(C) \cdot \Phi(g)$[3], i.e.

$$\Phi(C) \cdot \Phi(g) = \frac{1}{|C|} \sum_{c \in C} \Phi(c) \cdot \Phi(g)$$

IDK2

The idea in point anomaly detector is also useful for clustering. Given a cluster C, if a point g should belong to that cluster, then the dot product $\Phi(C) \cdot \Phi(g)$ should also be relatively large.
Note that if the points in the cluster C are mapped trajectories, and calculating the dot product is actually calculating the dot product in the second level IDK. That's why a new clustering algorithm[4] called TIDKC based on 2-level IDK is proposed.



Algorithm 1 TIDKC[4]

$D = \{\Phi_1(T_1), \cdots, \Phi_1(T_N)\}$ /*map trajectories*/
$E = \{C_1, \cdots, C_k\}$ /*k initialized clusters*/
$N = D - \cup_i C_i$ /* remaining trajectories to be assigned */
$\tau = \max_{g \in N, C_i \in E} \Phi_2(g) \cdot \Phi_2(P_{C_i})$ /* Choose the maximum possible dot product as the threshold*/
repeat
    $\tau \leftarrow \rho \times \tau$
    Expand cluster C_j to include unassigned point $g \in N$ for $j = \arg\max_{C_i \in E} \Phi_2(g) \cdot \Phi_2(P_{C_i})$ and $\Phi_2(g) \cdot \Phi_2(P_{C_i}) > \tau$
    $N \leftarrow G - \cup_i C_i$
until $|N| = 0$ **or** $\tau < 0.00001$
Assign each unassigned point g to nearest cluster C by $\Phi_2(g) \cdot \Phi_2(P_{C_i})$

produce as the threshold τ

repeat

  $\tau \leftarrow \rho \times \tau$

  Expand cluster C_j to include unassigned point $g \in N$ for $j = \arg\max_{C_i \in E} \Phi_2(g) \cdot \Phi_2(P_{C_i})$ and $\Phi_2(g) \cdot \Phi_2(P_{C_i}) > \tau$

  $N \leftarrow G - \cup_i C_i$

until $|N| = 0$ **or** $\tau < 0.00001$

Assign each unassigned point g to nearest cluster C via $\Phi_2(g) \cdot \Phi_2(P_{C_i})$

After initializing the clusters, we define a threshold τ to represent the threshold of the dot product. Each time, we attempt to classify an unclassified point g into the cluster with the largest dot product. If the dot product is smaller than the threshold, i.e., the quality of the cluster is poor regardless of where it is assigned, we abandon this classification. The process continues until all point classifications are completed or the threshold becomes too small. At that point, we will quit the loop and force the remaining points to be classified (if any).

Table of Contents

Implementation

The algorithm has been implemented. We utilize the `Geolife Trajectory 1.3` database for our experiments. This trajectory dataset was collected in Microsoft Research Asia Geolife project in a period of over five years. There are 182 individuals, and each individual has hundreds to thousands of trajectory files, and each trajectory file contains thousands of data points. Multiple experiments were conducted, each fetching trajectories from different number of individuals.

Implementation

Each trajectory is mapped to a 256-dimensional vector in Hilbert space. And each cluster within this Hilbert space is mapped to a 256-dimensional vector in a two-level Hilbert space.
These 256-dimensional vectors are generated from 16 samplings($t = 16$), wherein each sampling involves the random selection of 16 points. Voronoi diagrams are then utilized for partitioning the space based on these 16 points.
We apply k-means (in 1 level IDK) and TIDKC (in 2 level IDK) to cluster these vectors separately (#of clusters is given). We calculate the accuracy for comparison.

Implementation

The following table presents the experimental results.

individuals	# of data points	k-means-accuracy	TIDKC-accuracy
0,1	0.28M	.78	.81
0,1	0.28M	.79	.81
0,1	0.28M	.78	.80
0,1	0.28M	.78	.81
0,1,2,3	1.02M	.76	.82
0,1,2,3	1.02M	.73	.81

Comparing the two algorithms, we found that the TIDKC algorithm outperforms k-means and has a higher accuracy. The accuracy of the TIDKC method is consistently high, ranging from 0.80 to 0.82 for all the tested individuals. In contrast, the accuracy of k-means clustering is consistently around 0.78. Due to the randomness in the algorithm, we performed multiple executions on certain individual sets and found that TIDKC has high robustness. In the last line, the accuracy of k-means dropped to 0.73, while the accuracy of TIDKC remained stable.

Table of Contents

Conclusion

We provide a comprehensive demonstration of how to utilize the results presented in the papers[3, 4] to calculate the IDK and classify trajectories based on this kernel using TIDKC algorithm.
Experiments have shown that the TIDKC outperforms traditional k-means clustering approaches in accuracy and robustness. This improvement suggests that TIDKC can better capture the complex relationships within trajectory data, offering a more reliable framework for trajectory analysis and clustering.