GE Asset Tracking Final Report

Tabitha Sugumar, Jacqueline Araya,
Kun Tao, Fatima Koli

Mentor: Tapan Shah, GE Research Center

Friday, May 15, 2020

Introduction

Purpose

This capstone project is a collaboration between the Columbia Data Science Institute and GE on the subject of asset tracking. The goal of this project is to identify patterns in spatio-temporal trajectory (STT) data. As asset tracking becomes more common, large datasets of trajectories, including everything from taxi movements to hospital inventories, are being created. In the context of asset management, STT pattern recognition and clustering will help optimize movement and storage, decreasing costs and increasing efficiency levels.

A specific example is hospital inventory, which is the focus of one of GE's projects. Using these clustering methods in that context, they could identify

1) Seasonal patterns in the movement of assets. Knowing these patterns could help them optimize use of permanent assets and minimize rental equipment
2) Clusters with large numbers of trajectories that are longer temporally and spatially. These would represent assets with high traffic/demand. Using this info, they could increase the inventory for these assets.
3) Clusters with long and repetitive trajectory structures. For these assets, they could create an intermediate store-room to optimize asset management.

While this is the context for our project, the applications of STT clustering extend beyond asset management. Other applications include identifying optimal delivery patterns and understanding the strategies athletes employ as they are playing.

Our approach involves developing unsupervised clustering algorithms and measurement approaches that together allow us to cluster trajectories

1) That depend on space and time (space and time variant)
2) That depend only on space (time invariant)

3) That depend on neither and are being recognized by their structure (space and time invariant)

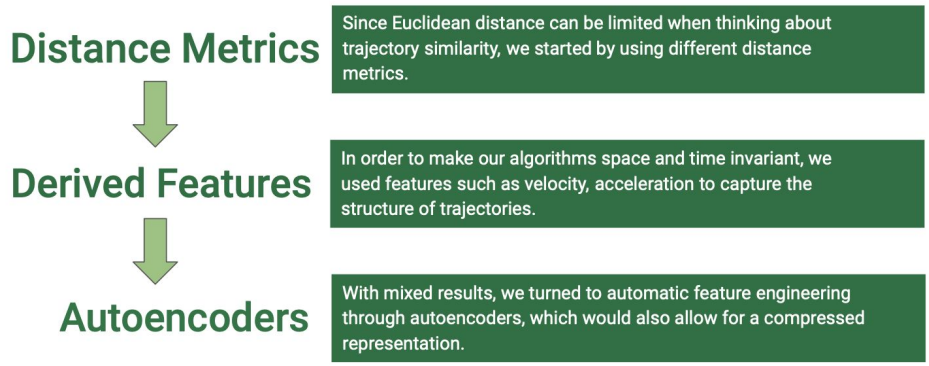
We want to develop a toolbox of such algorithms to apply to spatio-temporal clustering. We are most interested in the third approach, identifying trajectories with similar shapes, but we will work on the other two as well, since the others may work better for a different spatio-temporal clustering use case.

Overview

Unsupervised clustering algorithms group together similar elements, where similarity is defined by a distance metric. The challenge with spatiotemporal data, specifically trajectories, is how does one group together a series of points — how do we quantify the similarity between trajectories?

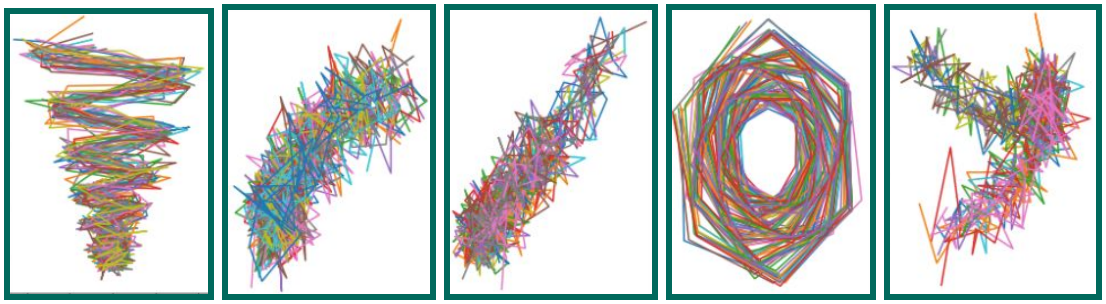
We approached this problem by researching and implementing three unsupervised clustering algorithms, k-means, agglomerative clustering and clustering trajectories, with different measures and metrics. We had three stages to our clustering (as represented in the figure below). We started by exploring four different metrics for measuring the distance between trajectories: longest common subsequence, dynamic time warping, edit distance, and frechet distance. While we had some success in our experiments with combinations of these distance metrics and clustering algorithms, we found that they were limited in consistently creating spatially and temporally invariant clusters.

We then attempted to create derived features (velocity, acceleration, angular velocity, angular acceleration, max magnitude) to capture additional aspects of the trajectories that could enable the creation of better shape-based clusters. We experimented with different combinations of these features and tested their performance in clustering. Since we had mixed results, we turned to vector embeddings generated by sequence to sequence autoencoders. Theoretically, using these embeddings for clustering rather than the original trajectories could incorporate additional information in the clustering process. We tested several variations of sequence to sequence autoencoder models, including experimenting with both LSTM and CNN based models. The following report includes both the concepts behind these experiments and their results – which proved promising.



Datasets

We developed our methods using the simulated datasets given below. These datasets were generated with a groundtruth which allowed us to evaluate whether or not we were creating the kinds of clusters we were looking for. They also had very visually distinct clusters, allowing a visual analysis of the results. The real world datasets also had a ground truth, but varied significantly in trajectory length, dataset size, etc. We evaluated our algorithms on these datasets to test how well they work for different types of STT data. The figure below shows the trajectories that made up the simulated datasets.



Simulated Datasets

Dataset Name	Description
STData_Short	100 trajectories of 50 points each
STData_Short_Noisy	100 trajectories of 50 points each, noisy
STData	10,000 trajectories of 50 points each
STData_Long	1,000 trajectories of 1,000 points each

Real World Datasets

Dataset Name	Description
1	122 trajectories with various lengths 1) Maximum_trajectory_length = 646 2) Minimum_trajectory_length = 11
2	900 trajectories with fixed lengths 1) 3000 points
3	22 trajectories with various lengths 1) Maximum_trajectory_length = 141,156 2) Minimum_trajectory_length = 87,985
4	22 trajectories with various lengths 1) Maximum_trajectory_length = 141,156 2) Minimum_trajectory_length = 87,985

Clustering Algorithms

We utilized three different algorithms during this project: Kmeans, Agglomerative, and DBScan because these algorithms are all well suited for unsupervised spatio-temporal clustering. We were aware of the benefits and limitations of each. KMeans is the fastest of the three, but it requires one to choose the number of clusters beforehand. Agglomerative is easy to interpret and intuitive, but does not scale well for large datasets. Lastly, DBScan deals well with noise and outliers, but has multiple parameters that require tuning.

Distance Metrics

Euclidean distance is limited when it comes to trajectories of different sizes and high dimensionality, therefore the following similarity measures were implemented and used for clustering an attempt to improve on the baseline:
LCSS: Similarity measure based on the number of points from two trajectories that can be considered equivalent. [1.1]
Edit Distance: Calculates the minimum of cost transforming one trajectory into another. Includes a hyperparameter to specify the weight to give the time aspect. [1.2]
Fréchet Distance: Longest distance between corresponding points of two curves. [1.3]
DTW: Similarity measure for time series that align and match in shapes, but are out of phase along the time axis. [1.4]

These similarity measures captured different aspects of trajectory differences. LCSS, edit distance, and DTW all involve creating a kind of matching between trajectories. LCSS is resilient to outliers and noise while edit distance can account for different sized trajectories fairly well and can include the temporal aspect. Interestingly enough, these two measures resulted in similarly shaped clusters when used with the same clustering algorithm (DBscan). DTW's phase matching works well for identifying spatially and temporally invariant patterns. Fréchet distance is somewhat different in that it compares points along a curve (always moving forward), and the distance is the longest distance between point pairs. One of the most significant costs of all these metrics is their time complexity. LCSS, edit distance and DTW are $O(mn)$ time while fréchet distance has a complexity of $O(mn * log(mn))$. This makes using these metrics infeasible for large datasets and long trajectories without significant preprocessing and/or downsampling.

Derived Features

Since we did not see a huge improvement with our distance metrics and since we now wanted to move to temporally and spatially invariant clustering methods, we attempted to create derived features to capture additional aspects of the trajectories. We hoped this would enable the creation of better shape-based clusters.

Velocity, Acceleration, Angular Velocity, Angular Acceleration

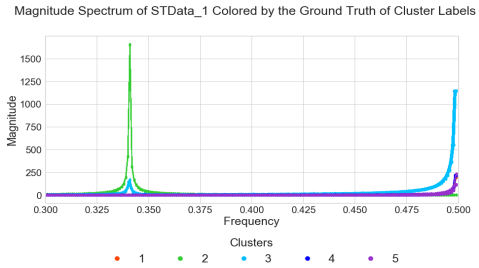
One approach to capture the shape of trajectories is by creating derived features. We calculated the velocity, acceleration, angular velocity and angular acceleration of each point in the trajectory with respect to the previous point. With these new features, we had more columns to characterize each trajectory, and explored the results of applying clustering algorithms over them.

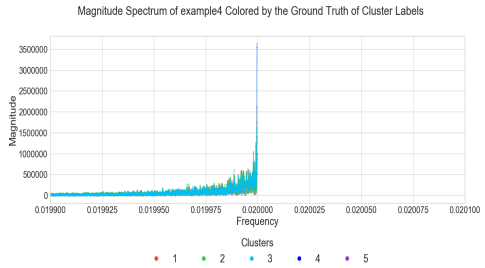
Since velocity and acceleration capture the rate of change, and the rate of change of the rate of change, respectively, the hope was they would function as good proxies of shape. However, the time sensitivity of these features may limit their ability to capture the type of clusters we are looking for. Additionally, to capture the angular nature a trajectory can take, angular derived features were calculated. For this, the 'X' and 'Y' cartesian coordinates were transformed to polar coordinates. Finally, both linear and angular velocities and accelerations are standardized to their means.

Max Magnitude

Identifiable patterns in time-space domain often correspond to recognizable structures in frequency domain. A trajectory defined by a sequence of (timestamp, X, Y) tuples can be

viewed as a two-dimensional time signal in space. With Fast Fourier Transform (FFT), a time signal can be uniquely represented by a pair of magnitude spectrum and phase spectrum in frequency domain. Trajectories shared with similar characteristics in the magnitude or phase spectra perhaps link to alike spatiotemporal properties, which justifies why a derivative feature extracted from frequency domain could be used for trajectory clustering. Potentially, the value of the frequency, magnitude and phase shift of each identifiable sinusoid produced by FFT could be the chosen one, depending on the dataset. For instance, if the magnitude spectrum reveals distinguishable group behavior in terms of the peak values as shown in the figure on the left, we can expect a good clustering performance by using the maximum magnitude of each trajectory as a single derivative feature, otherwise, for a magnitude spectrum as shown on the right, we can tell this feature will not work well, and if there is any patterns in the dataset of interest, it must be deeply hidden and we perhaps have to resort to complicated techniques such as metric DTW or feature representation by deep learning as illustrated in Section V.





Autoencoders

Given that manually created derived features offered limited improvement, we turned to autoencoders for both dimensionality reduction, and capturing additional trajectory structure. Autoencoders are a deep learning technique that can generate feature representations of time series/trajectories. LSTM and CNN based autoencoders are the most popular approaches, and both can hierarchically explore hidden patterns. We also implemented a simpler model, an autoencoder consisting only of dense layers.

LSTM Autoencoders

LSTM Seq2Seq Autoencoder is a powerful weapon in feature extraction of time sequences. It consists of one encoder and one decoder which are usually arranged symmetrically. The feature matrix of input sequences is fed into the encoder, and the output hidden state then goes to the decoder in such a way that the input sequences can be reconstructed with minimum errors. To use for clustering, we can remove the decoder, and just keep the fixed-length representation outputs from the last layer of the encoder for each trajectory. With a long memory capacity and a strength in modeling nonlinear

relationships, the LSTM-learnt deep representation is expected to well capture the complex while significant characteristics hidden in the original feature vector (X, Y) of each time series, and thus suitable for a better clustering performance.

Dense Model Autoencoder

One of the simplest, most common, neural network models one can design is one with dense layers, where all neurons of one layer are fully connected with adjacent layers. Usually these models have a larger amount of weights to learn in each training step, and take more training time.

The input for this model is a one-dimension input vector, so every column in our dataset was stacked as one long vector. This may result in loss of information given that trajectories are naturally sequences where order does matter, in contrast with an LSTM model.

CNN Autoencoder

Usually, convolutional neural network models are used with image data, because of its ability to extract features in with a window-type approach. Nevertheless, CNNs can also model trajectories since, conceptually, any point in a trajectory is influenced by its predecessors and has an influence over the next points, similar to LSTM models.
In the implementation of a CNN autoencoder model, 1 dimensional convolutional layers were used since convolutional operations need to be applied along the temporal dimension of the data points across multiple channels/features. The encoder part of this model, returns embedded vectors of learnt trajectories of 32 and 8 length

Results

Baseline and Summary of our Results

We utilized the fowlkes mallow score to evaluate our cluster results. The table below shows our baseline and a summary of those results with the best scores from each of the methods we utilized. The baseline (Kmeans and euclidean distance) had fairly good results, but it clearly didn't work as well for noisier datasets and datasets with long trajectories.

Utilizing derived features and distance metrics gave us almost the same accuracy on STData_Short as the baseline, but derived features were much faster. With the noisy dataset however, derived features did not perform as well so it appears they cannot

handle noise and outliers. For our larger datasets, our autoencoders show a huge improvement from the baseline, so with enough data, we would recommend using autoencoders. Note that not all of the methods were used for all datasets so there are some empty fields.

In the following sections, we go into detail for the results of different models that we utilized for each method.

Datasets	Baseline	Distance Metrics	Derived Features	Auto-encoders
STData_ Short	0.83	0.82	0.81	0.81
STData_ Short_Noisy	0.62	0.62	0.47	0.52
STData	0.77	-	-	0.90
STData_Long	0.61	-	-	0.87

Distance Metrics and Clustering Algorithms

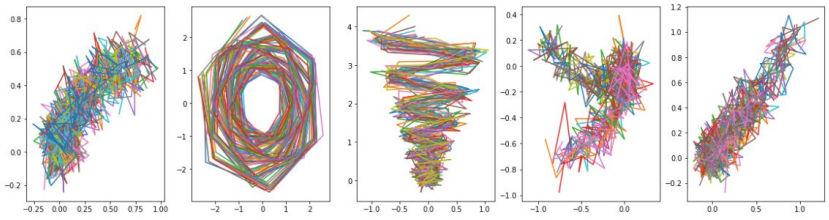
We experimented with several combinations of clustering methods and distance metrics. As is clear from the scores in the table above, none of these methods were successful in improving upon the baseline.

Algorithm	Distance Metric	Silhouette Score		Fowlkes-Mallows Score	
		STData Short	STData, Short, Noisy	STData, Short	STData, Short, Noisy
Agglomerative	DTW	0.76	0.21	0.81	0.52
	Frechet	0.77	0.32	**0.82**	0.57
Density Based Clustering (DBScan)	Edit Distance	0.71	0.16	0.81	**0.62**
	LCSS	0.67	0.03	0.81	0.49
Kmeans	Euclidean	0.62	0.08	0.83	0.62

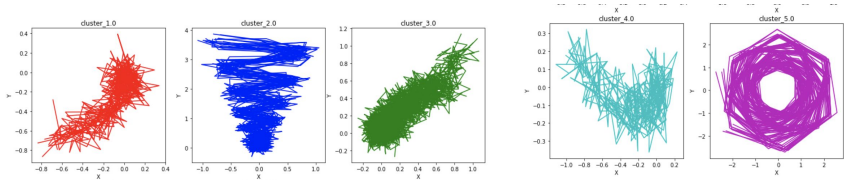
(Baseline)					

The figure below shows the results from the DBscan algorithm with LCSS:

Ground Truth:



Cluster Results:



Derived Features with Kmeans and Agglomerative Clustering

We attempted to capture additional structure of the trajectories through derived features. Unfortunately, they also failed to improve the results.

Algorithm	Features		Fowlkes Mallow Score	
			STData_Short	STData_Short_Noisy
Kmeans	Max Magnitude		**0.81**	0.47
	Velocity, Acceleration		0.55	0.29
Agglomerative	Velocity,Acceleration, Angular Velocity, Acceleration	K = 2, link = average	0.46	0.47
			0.50 (K = 10, link = average)	0.47 (K = 3, link = average)
Kmeans (Baseline)	Euclidean		0.83	0.62

Autoencoders

Autoencoders create a compressed representation of fixed length for each trajectory. These embeddings were fed into a clustering algorithm of choice. The results were a

significant improvement from those of the baseline, indicating that this approach successfully captured additional structure of the trajectories and allowed clustering algorithms to take it into account.

Autoencoder Agglomerative Clustering

Autoencoder	Embedding Length	Trainable Parameters	Fowlkes Score
			STData
Densely Connected	8	40,484	0.4876
	32	32,532	0.4827
LSTM	8	62,918	0.4657
	32	64,134	**0.6071**
CNN	8	1,175	0.4794
	32	1,274	0.4827
Baseline	N/A	N/A	0.7700

As we can observe, for these models, the highest score is achieved by the LSTM autoencoder on the embeddings of length 32. Nevertheless, this result is not as good as our baseline model using Kmeans algorithm. This outcome may be the result of overfitting and too little trajectory data to train the model. Given the high number of parameters the model has to train, having only 9,000 trajectories (out of 10,000) seems too low for the neural network to learn and update that number of parameters. Additionally, the model may be suffering from overtraining because we trained and predicted using the same dataset, given the size of it to avoid training with too few trajectories. Finally, the low Fowlkes value score may be also explained by the type of clustering. Agglomerative clustering is not as flexible as Kmeans to compute its clusters given its hierarchical setting.

KMeans Clustering

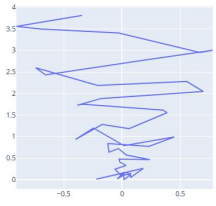
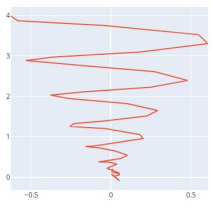
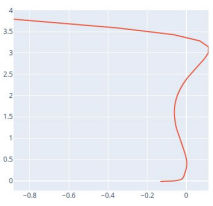
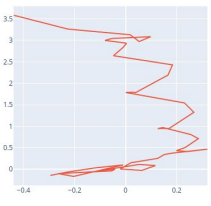
Below are some of the key results from running KMeans on encoded results. KMeans outperforms all of our other models and has a much higher score than our baseline. As with the Agglomerative clustering results, LSTM encoding performs better than CNN. Knowing that LSTMs are better for time dependent, sequential data, this is in line with our expectations.

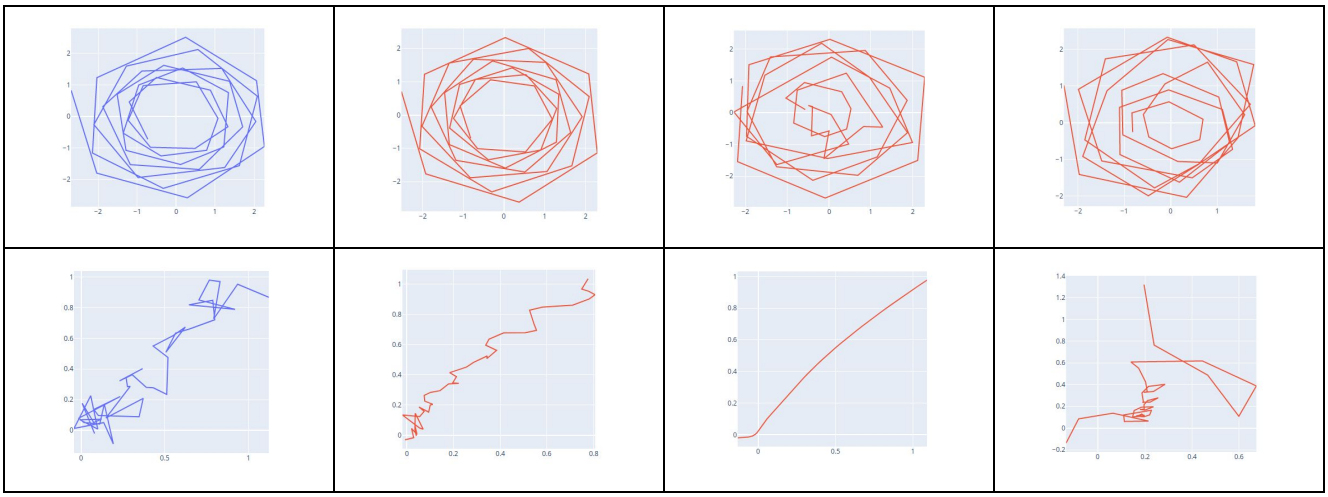
Autoencoder	Fowlkes Score	Fowlkes Score
	STData	STData_Long
LSTM	**0.90**	**0.87**
CNN	0.79	0.73
Baseline	0.77	0.61

Visualizing Autoencoders

Reconstructed trajectories

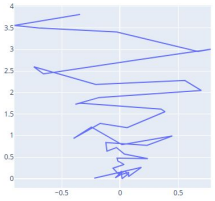
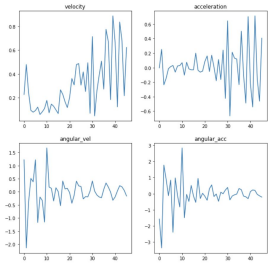
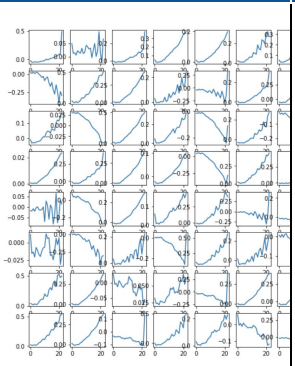
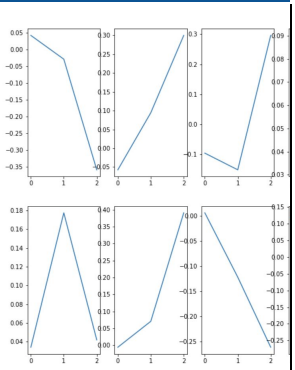
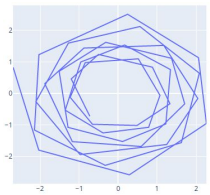
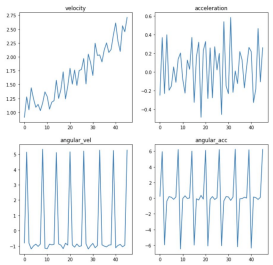
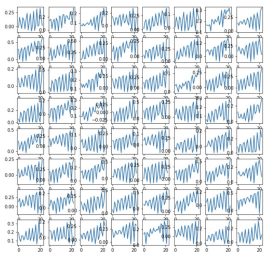
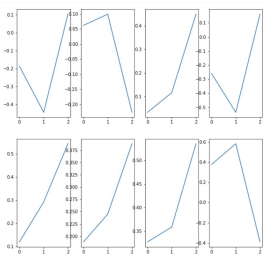
Even though we are interested to learn compressed representations of our trajectories with autoencoders, a less relevant but interesting part of autoencoders is its ability to demonstrate it can reconstruct the input data from the compressed representation of it to its original form; this is the decoder task and can be useful to prove that the obtained compressed representation are capturing the most essential characteristics of each trajectory. Below, we can observe examples of reconstructed trajectories by the trained autoencoders and judge the performance of each decoder:

Original Trajectory	Dense autoencoder	LSTM autoencoder	CNN autoencoder
			



Intermediate layers

We also wanted to explore some of the intermediate layers of the autoencoder in order to see what features it was capturing. We compared the results from our layers to the manual features we had created in order to see if the autoencoder was picking up some of the same features. The image below shows two of our trajectories, the manual features we created for them, the results of one of the intermediate layers, and the final encoded result, which is what will be inputted to our clustering algorithms. As can be seen in both of the trajectories, the autoencoder appears to be picking up some of the manual features in some of the activations in the intermediate layer. This is based on visual inspection, since differing lengths of the intermediate encoding and our manual features doesn't allow for a more quantitative measure of similarity. This is interesting and we had hoped to see such results. The intermediate layers are also picking up on many other aspects of the structure of the trajectories that we obviously would not be able to with just manual feature engineering.

Original Trajectory	Manual Features	Intermediate Layer	Encoded Result
			
			

Real World Datasets

Four different real world datasets are given to evaluate the clustering performance of the aforesaid metrics, derivative features and autoencoders, with each dataset serving a specific group of tests as listed below.

Dataset 1

This dataset consisted of 122 trajectories with various lengths. The longest trajectory contained 646 points and the shortest contained 11. Autoencoders especially do not deal well with different sized trajectories. To account for that, we interpolated between data points so that the structure of the trajectory could remain intact, but we would have similar length trajectories. The results are given below. Although they are not very high, this is not surprising given that these trajectories are not as clean as our original data. Moreover, the dataset was very imbalanced, with many more trajectories for certain labels compared to others. With more pre-processing and parameter tuning, we could hope to see better results.

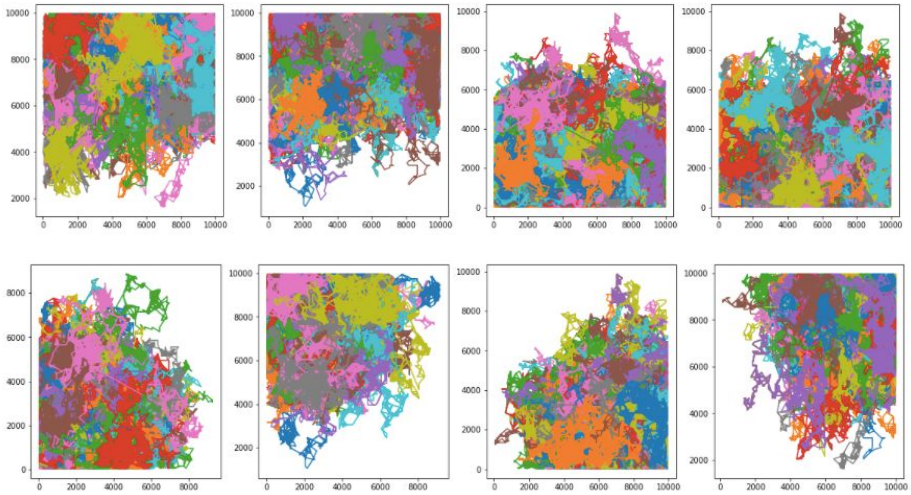
Model	fowlkes_mallows_score
Baseline	0.19
CNN 3 layers - KMeans	0.48
CNN 4 layers - KMeans	0.21

Dataset 2

This dataset consisted of 900 trajectories with 3000 points each. We experimented with training two layer and four layer LSTM models on this dataset, and clustering the generated results with KMeans. The results, in comparison to the ground truth are given in the table below.

Model	fowlkes_mallows_score
Baseline	0.49
LSTM 2 hidden layers	0.47
LSTM 4 hidden layers	0.48

In this case, the baseline actually worked better than the autoencoder based models. Below are the visual results of the ground truth and baseline clustering.



Top: Groundtruth, Bottom: Baseline

While the groundtruth clusters are more up-down shaped, the baseline results are grouped to the side (the visual results were similar for the autoencoder based models [2.1]). Something to consider here is the length of the trajectories; they were 3000 points each. During the development process we found that a four layer LSTM model very much improved the results for STData_long, which had trajectories of 1000 points. It seems possible that a deeper autoencoder with a larger final vector embedding (ei: 64 points instead of 8) could drastically improve on these results. Resource limitations (lack of memory) prevented us from trying this out, but it would be worth experimenting with.

Dataset 3

For dataset 3, the 22 trajectories are divided into smaller trajectories of 7,500 points each. This way, we get 377 trajectories to train our autoencoders models. The following table shows the Fowlkes score of the Densely connected autoencoder using Kmeans and Agglomerative cluster over the resulting embeddings. The score shown is the best one obtained from a range of different lengths of embeddings, 17 for Kmeans and 13 for Agglomerative clustering.

Model	fowlkes_mallows_score
Baseline	0.29
Dense autoencoder - Kmeans	0.28
Dense autoencoder - Agglomerative	0.53

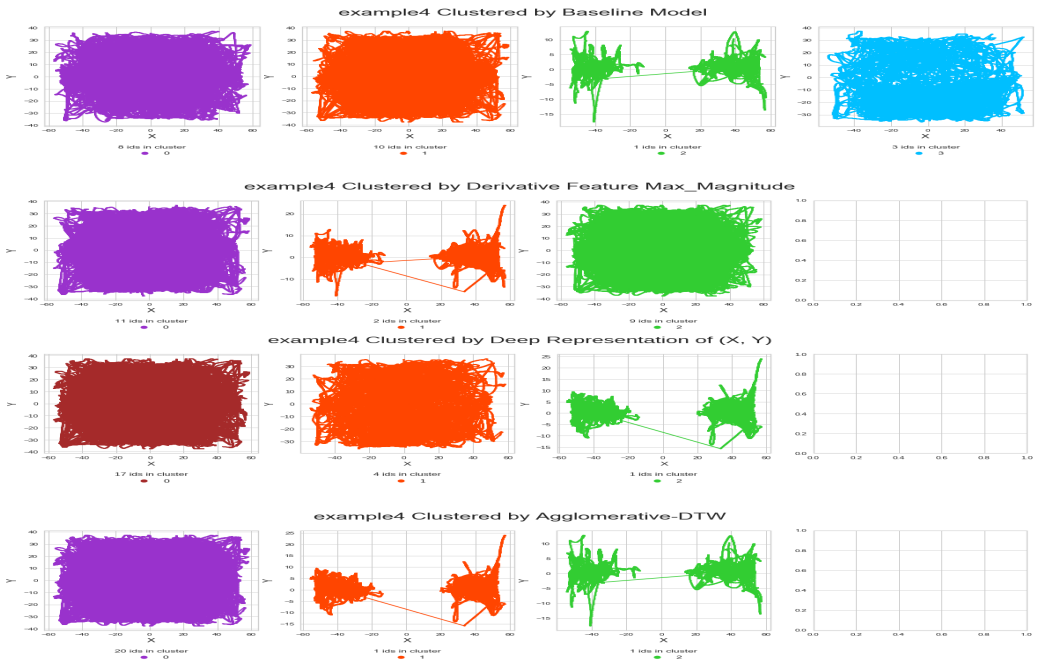
Dataset 4

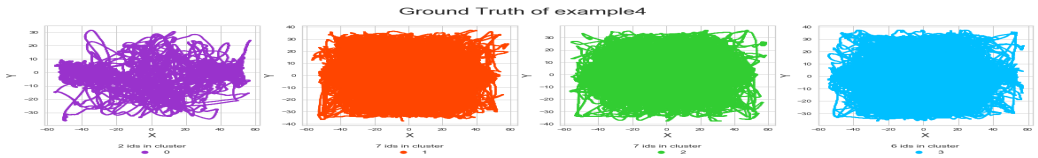
Dataset 4 contains 22 trajectories in total. The longest trajectories consist of 141156 points, whereas the shortest one only has 87985 points. Dataset 4 was used to compare the clustering results obtained by the baseline model, derivative feature Max_Magnitude, Deep Representation of (X, Y) Learnt by LSTM and DTW-distance feature matrix.

Model	fowlkes_mallows_score
K-means on original (X, Y) (Baseline)	0.30
K-means on single derivative feature Max_Magnitude	0.30
K-means on Deep Representation of (X, Y) Learnt by LSTM	0.40
Agglomerative on pairwise DTW-distance feature matrix	0.52

Trajectories in the dataset 4 are too long, which either crashed the memory and made it impossible to calculate the DTW-distance matrix or very time-consuming to train the LSTM model. Therefore, each trajectory was downsampled 100 or 1000 times to run these two models. On the other hand, the clustering process on the Max_Magnitude or baseline is blazing fast and does not need any downsampling.
Overall, clustering performances achieved on the dataset 4 are disappointing, but better than a "random clustering" of which the expectation of FM score is 0.25. The metric DTW shows the highest FM score of 0.52, and the Max_Magnitude has a stable score of 0.3. FM scores obtained from the baseline and LSTM exhibit wild fluctuations for each re-run. Very likely, a major reason for this kind of instability is because there are only 22 trajectories available in the dataset, which is too few and prone to variations introduced by the random initialization when using the feature vector (X, Y) directly. In brief, the LSTM-autoencoder is not suitable for clustering dataset 4 considering its inconsistency.

Visualization of Clustering Results for Dataset 4





Ethical Considerations

For this project, ethical considerations did not critically interfere with the development of any model or analysis.
For confidentiality reasons, our affiliated mentor shared synthetic trajectory data based on his knowledge of real trajectories of GE assets, which we used to perform all the analysis, modeling and development of methods and algorithms. Thus, our datasets do not contain any type of PII (personal identifiable information). GE's real trajectory of assets was provided only to test the most successful methods obtained from our analysis, and this data was not identifiable in any way (assets ID's were *anonymized)*.

However, regarding the scope of our project, the initial objective was to find competitive methods to cluster assets based on their trajectories and additionally to optimize asset management and inventory. We believe this goal imposes a set of other ethical questions to think about, like what happens if clustering results of assets suggests a higher inventory of medical assets in one hospital location and very low one in another, leaving a community under served? Or taxi rerouting based on optimized clusters? Another possible concern is the application of these methods to other types of STT datasets, for example, people's movements collected via smartphones -- depending on the dataset privacy could be a concern. These, and other ethical discussions are not only interesting, but relevant at the time of decision making for a company like GE.

Conclusion

In the course of this project we attempted to develop clustering methods that would work well on a variety of STT datasets. In the development process we found that using an LSTM autoencoder to create trajectory embeddings and then clustering those embeddings with KMeans worked best in creating the spatially and temporally invariant clusters. However, when we applied our methods to evaluation datasets we found that we couldn't quite generalize to say that this was the best approach across all types of STT datasets. For example, DTW with Agglomerative clustering outperformed the autoencoder based models for the fourth evaluation dataset. That dataset had only 22 trajectories -- there was simply not enough data for an autoencoder to work well. Based on the results of the evaluation datasets, when applying the methods we developed to STT datasets, a

reasonable approach would be to try out a few methods to see which approach works best for the given dataset. Something else to consider is dataset specific preprocessing, which could significantly improve outcomes.

Moreover, in many contexts, the goal may simply be to derive patterns from large amounts of data. For example, some researchers explored soccer player's patterns on the field to understand their movement and strategies. These strategies were not known in advance, but by starting with the clustering results, they could characterize different clusters using their background knowledge.

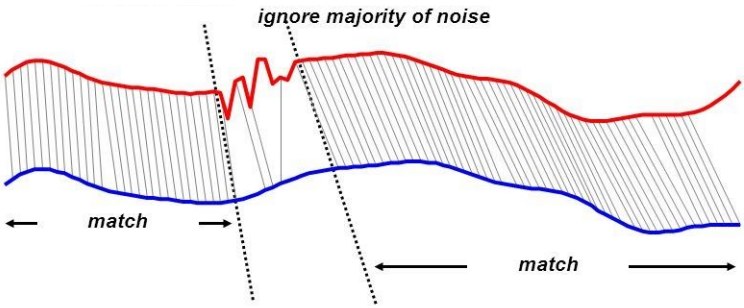
We see these clustering methods to be the most valuable in such contexts since they would reveal patterns that practitioners could explore further. We hope that the methods here provide a good framework of possible approaches to STT dataset clustering, that can be fine tuned for use with specific datasets.

References

- Eiter, T., & Mannila, H. (1994). Computing discrete Fréchet distance (pp. 636-637). Tech. Report CD-TR 94/64, Information Systems Department, Technical University of Vienna.
- Keogh, E. and Ratanamahatana, C.A. (2004), “Exact Indexing of Dynamic Time Warping”, *Knowledge and Information Systems*, DOI 10.1007/s10115-004-0154-9.
- Lee, Jae-Gil & Han, Jiawei & Whang, Kyu-Young. (2007). Trajectory Clustering: A Partition-and-Group Framework. Proceedings of the ACM SIGMOD International Conference on Management of Data. 10.1145/1247480.1247546.
- Toohey, K. and Dukham, M. “Trajectory Similarity Measures”, *SIGSPATIAL Special 8,* Vol. 1, pp. 43-50.
- Sakoe, H. and Chiba, S., "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", *IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING*, Vol. ASSP-26, NO. 1, pp. 43-49, Feb. 1978.
- Yihong Yuan & Martin Raubal (2014) Measuring similarity of mobile phone user trajectories– a Spatio-temporal Edit Distance method, International Journal of Geographical Information Science, 28:3, 496-520, DOI: 10.1080/13658816.2013.854369.

Appendix

1.1 Longest common subsequence (LCSS) is a similarity measure representing the number of points from two trajectories that can be considered equivalent, allowing for both unmatched points and the stretching of trajectories to create a match (Toohey, K and Duckham, M 2015). The figure below shows one example of LCSS matching. It deals well with noise and outliers by ignoring them since it does not require all points to be matched.



In order to account for varying trajectory lengths, the number of matching points is standardized by dividing by the length of the shorter trajectory. LCSS takes two trajectories, T_A and T_B, with lengths m and k, a matching threshold ε, and an integer $\delta \geq 0$, which represents by how much the two trajectory lengths are allowed to differ. The time complexity using dynamic programming is $O((m+k)\delta)$ and it is calculated as below:

$$LCSS_{\delta,\varepsilon}(T_A, T_B) = \begin{cases} 0, & \text{if } T_A \text{ or } T_B \text{ is empty} \\ 1 + LCSS_{\delta,\varepsilon}(Head(T_A), Head(T_B)), & \text{if } |m-k| \leq \delta \text{ and } |a_{m,1} - b_{k,1}| \leq \varepsilon \\ & \text{and } \ldots \text{ and } |a_{m,n} - b_{k,n}| \leq \varepsilon \\ max(LCSS_{\delta,\varepsilon}(Head(T_A), T_B), & \\ \quad LCSS_{\delta,\varepsilon}(T_A, Head(T_B)), & otherwise, \end{cases}$$

Pseudocode for LCSS (Toohey, K and Duckham, M 2015)

1.2 The edit distance algorithm calculates the minimum cost to add, replace, or delete points from the first trajectory, so that it is transformed into the second trajectory.

Given two trajectories, t1 and t2, each cost function measures how much t1 changes from either inserting a point from t2, replacing a point in t1 with a point in t2, or deleting a point in t1. The more the change alters the centroid of t1 the higher the cost (Yuan 2014). The cost functions are as follows:

$$cost[delete(p_{1i})]$$
$$\sqrt{(1-c)[(\frac{\sum_{k=1}^{n} x_{1k}}{n} - \frac{\sum_{k=1,k\neq i}^{n} x_{1k}}{n-1})^2 + (\frac{\sum_{k=1}^{n} y_{1k}}{n} - \frac{\sum_{k=1,k\neq i}^{n} y_{1k}}{n-1})^2] + c[(\frac{\sum_{k=1}^{n} t_{1k}}{n} - \frac{\sum_{k=1,k\neq i}^{n} t_{1k}}{n-1})^2]}$$
$$cost[insert(p_{1i})]$$
$$\sqrt{(1-c)[(\frac{\sum_{k=1}^{n} x_{1k}}{n} - \frac{\sum_{k=1}^{n} x_{1k}+x_{2j}}{n+1})^2 + (\frac{\sum_{k=1}^{n} y_{1k}}{n} - \frac{\sum_{k=1}^{n} y_{1k}+y_{2j}}{n+1})^2] + c[(\frac{\sum_{k=1}^{n} t_{1k}}{n} - \frac{\sum_{k=1}^{n} t_{1k}+t_{2j}}{n+1})^2]}$$
$$cost[replace(p_{1i}, p_{2j})]$$
$$\sqrt{(1-c)[(\frac{\sum_{k=1}^{n} x_{1k}}{n} - \frac{\sum_{k=1,k\neq i}^{n} x_{1k}+x_{2j}}{n})^2 + (\frac{\sum_{k=1}^{n} y_{1k}}{n} - \frac{\sum_{k=1,k\neq i}^{n} y_{1k}+y_{2j}}{n})^2] + c[(\frac{\sum_{k=1}^{n} t_{1k}}{n} - \frac{\sum_{k=1,k\neq i}^{n} t_{1k}+t_{2j}}{n})^2]}$$

Figure 2. Edit Distance Cost Functions Yuan (2014)

This version of edit distance allows the user to weight the value of the temporal aspect of the data with the parameter c, $0 \leq c \leq 1$. When c = 0, time does not factor into the cost, so the metric is temporally invariant and spatially variant. However when c = 1 the spatial aspect of the data is not included, so the metric is temporally variant and spatially invariant. Finally, when $0 \leq c \leq 1$, the metric is both temporally and spatially variant.

This metric requires scaled data, and its runtime is O(nm). This makes it difficult to use with large datasets and/or long trajectories.

1.3 A curve f can be defined as a continuous mapping $f : [a, b] \to V$ where $a, b \in \mathbb{R}$ and $a \leq b$ and (V, d) is a metric space. Notice here, d is the distance function use to measure how far apart two points are in curve f in the plane V.
The frechet distance between two curves P and Q in the same metric space is defined as:

$$\delta_F(P, Q) = \inf_{\alpha,\beta} \max_{t\in[0,1]} d\big(P(\alpha(t)), Q(\beta(t))\big)$$

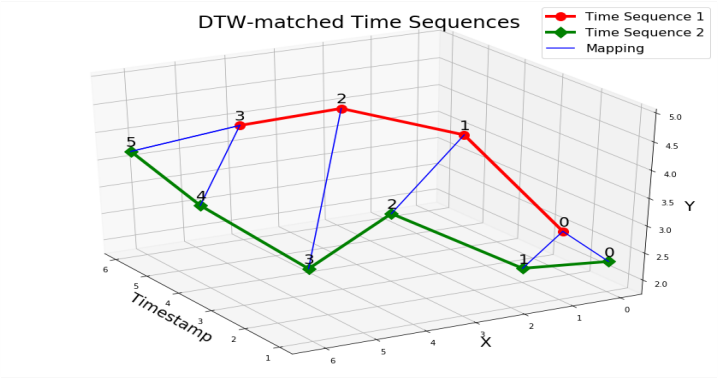
where α and β are continuous non-decreasing functions from $[0, 1] \to [a.b]$. To compute frechet distance a polygonal approximation of the curves is required, getting a discrete form:

$$\delta_F(P, Q) = \inf_{\alpha,\beta} \max_{t\in[0,1]} ||P(\alpha(t)) - Q(\beta(t))||$$

Informally, we can think of frechet distance as follows: Imagine a man walking a dog on a leash, both walk by different paths, defining different trajectories and are only allowed to move forward (i.e. no backward steps). When they finish the walk, what is the minimum amount of leash required to recreate the walk? The minimum amount of leash is determined by the maximum distance between any two ordered points of both trajectories.
This distance takes into account the directionality of the curves, i.e., the course of the points in each trajectory, this is why it is used in applications such as handwriting and protein structure analysis. Eiter (1994) proposes the best known algorithm to approximate the discrete variation of frechet distance, used in this project.

1.4 Dynamic Time Warping (DTW) is a robust similarity measure for time series, which can effectively align, match, and cluster series that have similar shapes but are out of

phase along the time axis (Keogh, E. and Ratanamahatana, C.A. 2004; Sakoe, H. and Chiba, S. 1978), as shown in the figure below.



The major strength of DTW is that it shows a strong capability in detecting time-invariant patterns, and with some preprocessing of the input datasets, it also displays a competitive performance in finding space-invariant structures. The time and space complexity to calculate the DTW-distance for a pair of trajectories are $O(nm)$, thus DTW is not applicable to big datasets due to its time-consuming and memory-crashing nature, if without a proper downsampling of the input datasets.

2.1 Visual results for clustering second evaluation set with 2 layer LSTM and 4 layer LSTM respectively