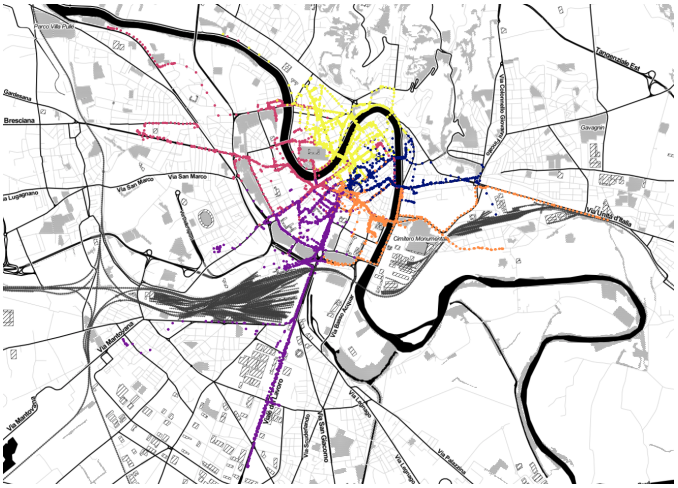
Scooter Trajectories Clustering

University of Verona
Computer Engineering for Robotics and Smart Industry
Machine Learning and Deep Learning
2020/2021

Mirco De Marchi - VR445319

***Abstract*—The aim of this report is to present the main unsupervised learning techniques used for scooter trajectories clustering. The dataset that I used contains a big amount of positions taken in rentals that run through some cities in Italy. The objective is to find recurrent places crossed by the trajectories. This report presents some analysis on the positions and an implementation of heuristics that manage data in a systematic way: *timedelta heuristic*, *spreaddelta heuristic*, *edgedelta heuristic*. I performed the most traditional machine learning clustering techniques on the generated dataset: *K-Means*, *Mean Shift*, *Gaussian Mixture*, *Full Hierarchical Agglomerative*, *Ward Hierarchical Agglomerative*. Then I applied a Deep Clustering technique that revisits the trajectory clustering problem by learning quality low-dimensional representations of the trajectories. I designed three different Auto-Encoder Deep Neural Network architectures, that learn the trajectory features in a latent space used then for a traditional clustering algorithm. The features are extracted with *Principal Component Analysis (PCA)* and *Moving Behavior Feature Extraction*, a sliding window algorithm that extracts a set of moving behavior features to capture space- and time- invariant characteristics of the trajectories.**



I. Motivation

Motion trajectories are really difficult to analyze and handle because of the amount of data. There could be errors in position tracking, due to localization issues, and usually data are not organized as you expect. Consequently trajectories are difficult to represent, filter and manage in relation with themselves or other information. First of all, trajectories clustering is a challenge for its intrinsic difficulty in being treated and today is an ambitious topic in data science research. Moreover, trajectory clustering can help for several applications:

- Monitoring: understand main points of interest and common places visited by tourists;
- Forecasting: prediction of possible destinations starting from the current position and previous ones;
- Viability: traffic monitoring and kind of user's activity extracting semantic concepts from trajectories;
- Smart city: data support to city plan and smart transportation management;
- Security: trajectories which are significantly different from others in terms of some similarity metric may be viewed as outliers;
- Video analysis: movement pattern analysis from video (after extracting trajectories from video data);

II. State of art

Yuan et al. (2012) [1], in their trajectory clustering review, show some techniques widely used for moving object clustering. The latest Machine Learning algorithms are all evolution and improvement of the traditional clustering techniques: K-Means, Mean Shift, Gaussian Mixture, Hierarchy, Agglomerative, etc. For spacial based clustering, *Palma et al. (2008)* [2] considers trajectories geometrical properties to detect pauses and moves in trajectories by using an improved version of *DBSCAN* algorithm. For time depended clustering, *Nanni and Pedreschi (2006)* [3] considers time gaps between trajectory positions and proposes *T-OPTICS* as an adaption of *OPTICS*. For partition based clustering, *Lee et al. (2007)* [4] implement a *partition & group* framework for trajectory clustering based on a formal theory. For uncertain trajectory clustering, Nock and Nielsen 2006 [5] implemented *Fuzzy C-Means (FCM)* that model the uncertainty and reduce the noise of trajectories location between discrete time updates. For semantic trajectory clustering, several works start considering geographical informations as a background from which extracting higher semantic level information (*Palma et al. (2008)* [2] stops and moves).

On the other hand, the main Deep Clustering techniques are based on autoencoders and was first introduced with the *D. Yao et al. (2017)* [9] work, in which they present a technique based on a sliding window that extract behavior feature along the trajectories and further employ a sequence to sequence autoencoder to learn fixed-length deep representation. The autoencoder based clustering, introduced in *D. Yao et al. (2017)*

[9] work, reduces the trajectory representation in a latent space contained in the neural network architecture, but then a traditional clustering algorithm has to be applied in order to locate similar trajectories. *Olive et al. (2020)* [8] employs a deep clustering method named *DCEC (Deep Convolutional Embedded Clustering)*, that uses a convolutional autoencoder and it can identify the clusters directly without the need of using a classical clustering method. In the same work ([8]) they implement *artefact*, a method based on a feed-forward autoencoder and a loss function combining the reconstruction error and a regularisation term based on the projection operator *t-SNE*. *M. Yue et al. (2019)* [6] have currently produced the best deep clustering solution that is named *DETECT (Deep Embedded TrajEctory ClusTering network)*. *DETECT* operates in three parts: first it transforms the trajectories by selecting its own point-of-interest for each position based on its context derived from their geographical locality, then they use autoencoder to learn trajectory representation and finally the a clustering oriented loss is directly built on the latent space features to refine the cluster assignment.

Furthermore, there are works on Deep Clustering that do not adopt the autoencoder architecture. *Mahdi M. K. et al. (2015)* [10] proposed an algorithm composed by four phases in which they break down trajectories into flow vectors that indicate instantaneous movements and then they apply different clustering steps in order to extract similarity with respect to their location, velocity, spacial proximity, motion and reachability. *Manduchi L. (2020)* [7] present *DPSOM (Deep Probabilistic Clustering with Self-Organizing Maps)*, that is based on the *SOM (self-organizing-map)*, which is a clustering technique that creates a nuance between the boundaries of the cluster, thus giving a more flexible interpretation of the clusters. In particular the *T-DPSOM* algorithm outperforms baseline methods in time series clustering and time series forecasting.

In this report I will present some partition and group heuristics, based on time gaps, I will show how the traditional clustering algorithms can be used for trajectory clustering relying on positions spatial information and I will compare the results with Deep Clustering techniques based on the autoencoder architecture explained in [9].

III. Objectives

The objectives of this project is grouping trajectories in order to find common locations crossed by people that rent a scooter to visit a city in Italy. The model built has to be able to distinguish for example the positions related to a trajectory that takes from the train station to the city centre, the ones that move in the city centre, the ones that run through the periphery and so on.

In particular the steps that involves the project are:

1) Dataset filtering and merging in order to build a dataset tidier and semantically correct;
2) Analyse features, group the positions for each rental and sort them through the timestamp;
3) Apply heuristic techniques on features;
4) Apply unsupervised learning techniques on features;
5) Implement the *Behavior Feature Extraction* algorithm;
6) Design and train the autoencoders in order to obtain the latent representation of the trajectories (with and without the Behavior Feature Extraction algorithm);
7) Apply clustering techniques on autoencoders latent representation;
8) Study the results and evaluate the goodness of clustering;

IV. Methodology

A. Dataset

The original dataset is composed by 4 tables in CSV fomat: *pos.csv*, *rental.csv*, *user.csv*, *device.csv*. This dataset is a subset of another dataset with some sensitive data dropped (like user name or email) and positions manumit in order to protect proprietary data. Although this dataset is only a subset of the proprietary data, the positions amount is really huge and it weighs about 2GB.

The *pos.csv* table contains all the positions, characterized by latitude, longitude, speed, timestamp and a rental id used to join with *rental.csv* table. The *device.csv* and *user.csv* tables contain the kilometres travelled respectively by a scooter and by a user. At last, the *rental.csv* table contains the start position and end position in couple of longitude and latitude of the rental trajectories with related start and stop timestamps, and the ids used to join the device and user tables. The dataset was subsequently reorganized, filtered and sorted in such a way to have positions in relation with its rental and the positions that belongs in the range of rental start-end time. This operation was really tricky because the total amount of data is huge and I had to optimize database join algorithms and chunks management to be able to handle these data in acceptable time, but in the end the resulting dataset properties diagram is 21. The figure 1 shows all positions in the two Italy cities contained in *pos.csv* table.

Due to how the dataset is constructed, a trajectory can initially be represented as a set of positions $trajectory = \{p_1, p_2, p_3, ..., p_t\}$ that belong to the same rental sorted by the timestamp. Each position p is a tuple $(t_p, lat_p lon_p, s_p)$ where t_p is the server timestamp or the device timestamp, lat_p and lon_p are the longitude and the latitude of the position and s_p is the speed. Formally a trajectory is $trajectory(rental\_id) = \{p \mid p.rental\_id == rental\_id\}$ and $TRAJ$ is the set of all trajectories 2. The resulting figure of the trajectory divided in rental is 3.

B. Heuristics

After that, I performed some analysis on features. I studied their distribution and I started to think how could be possible to group or divide positions. Two trajectories can be similar in shape and can be divided in time. It means that a sequence of positions, a trajectory, can be similar to another one, evaluating his shape appearance and location in the geographical map, and therefore his sequence of their geographical coordinates. On the other hand, a trajectory can be different to another one if there is a temporal space between each other.

As result I implemented 3 different clustering heuristics performed in a systematic and statistical way on the entire sequence of positions:

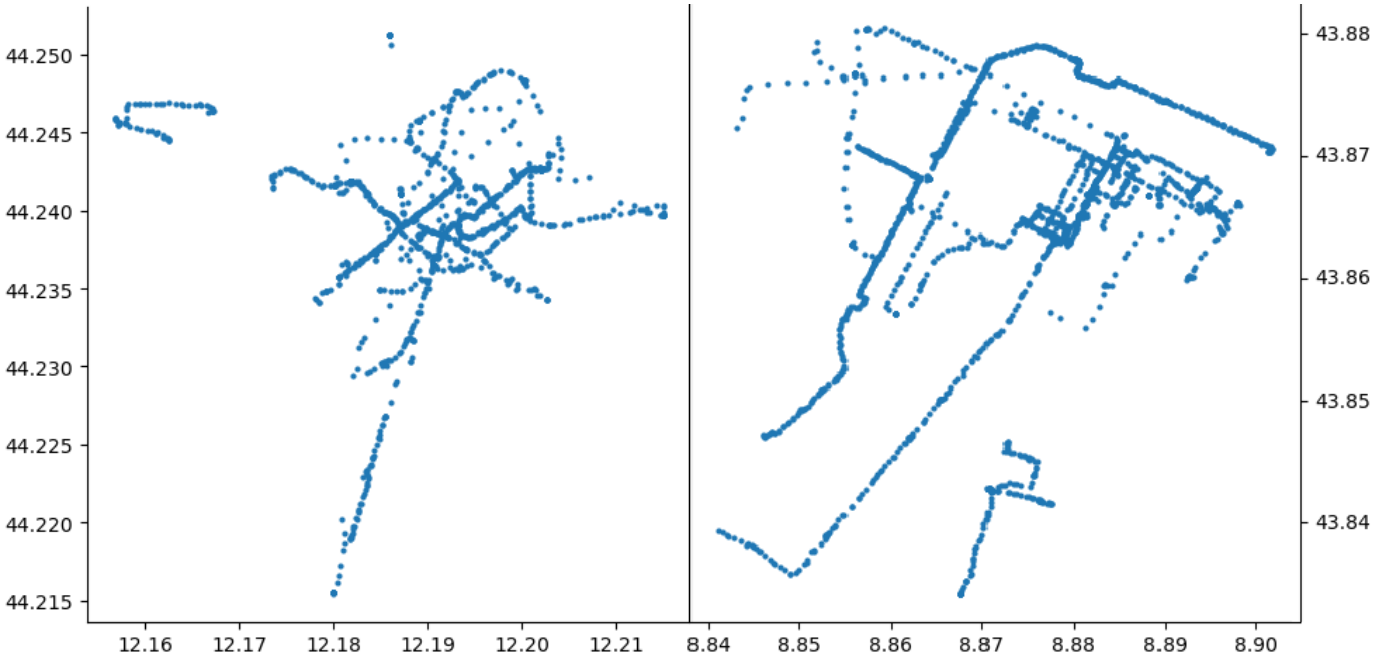


Figure 1. Trajectories without clustering in 2 Italy cities

Figure 2. Trajectories belonging rentals

Dataset	Samples	Features
rental	14826	10
pos	817076	18
merge	817076	18
dataset	14826	13
partition city 1	608251	18
partition city 2	202795	18

Table I: Number of features and samples of generated dataset

- **timedelta heuristic**: considers that a trajectory of a rental can be divided in a sequence of trajectories if the time gap between a position and previous one exceeds a *timedelta* value. First of all I calculated the time gaps for each set of positions grouping in rental:

$$TIMEGAPS = \{t.time - shift(t, -1).time \mid \forall t \in TRAJ\} \quad (1)$$

where `shift(t, -1)` shifts the trajectory backward and the field `.time` gives the timestamp of each position in trajectory. The *timedelta* value has been manually assigned or automatically calculated with the statistical empirical rule (three-sigma rule or 68-95-99.7 rule). The timegaps set is then used to divide the trajectories if the gap exceeds the defined timedelta (figure 4).

- **spreaddelta heuristic**: considers that a rental trajectory can be considered similar to another one if they spread a similar amount of area (figure 5). I calculate the spread area for each rental trajectory in the following way:

$$SPREADS = \{max(t) - min(t) \mid \forall t \in TRAJ\} \quad (2)$$

where `max(t)` and `min(t)` calculate respectively the maximum and the minimum latitude and longitude of a set of positions, and `TRAJ` is the set of trajectories that can be grouped for each rental or for the *timedelta heuristic* (figure **??**). Also in this case, I exploited the empirical rule to compute the spreaddelta value.

- **edgedelta heuristic**: acts as the *spreaddelta heuristic*, but it considers the edges of a trajectory, or rather the first position and the last position of a trajectory 7. The main problem here is that the distribution of edge positions has 2 centres or, in other words, it is bimodal. To resolve this issue I applied for simplicity only 1 iteration of *Mean Shift* starting from a random position in order to get closer to one of 2 centre. The set of edges is calculated in the following way (figure 8):

$$EDGES = \{concat(p[0], p[-1]) \mid \forall t \in TRAJ\} \quad (3)$$

- **coorddelta heuristic**: it is a combination of spread and edge heuristics in order to combine the main advantages of each other.

C. Feature extraction

For feature extraction I decided to perform *Standardization*, *Normalization* and than *Principal Component Analysis (PCA)*.

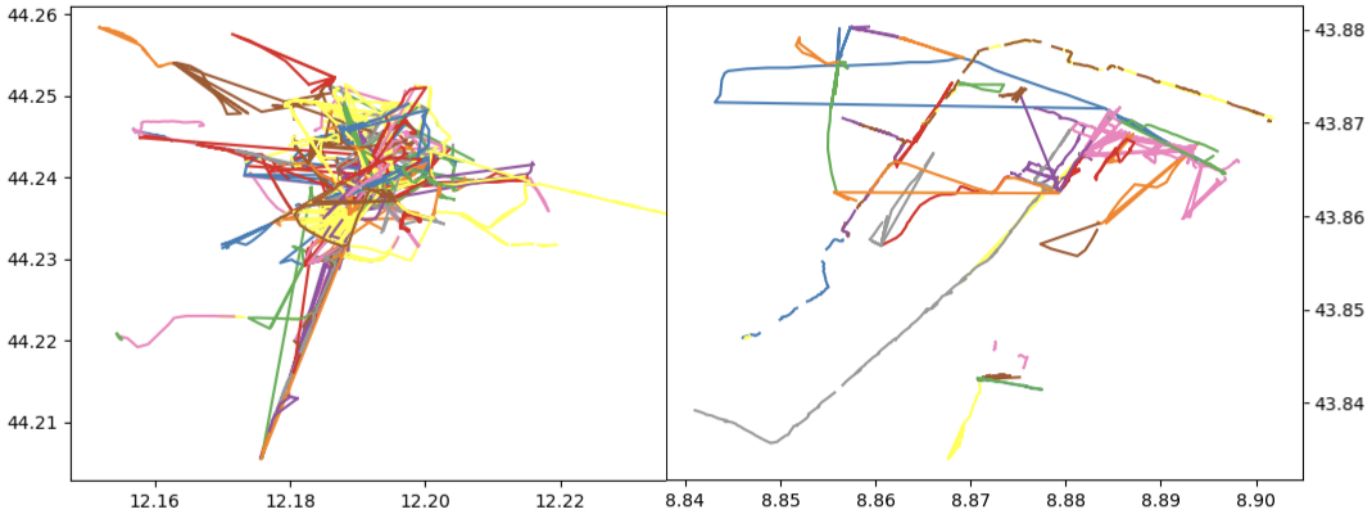


Figure 3. Trajectories belonging rentals

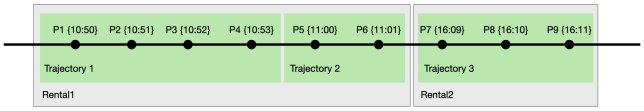


Figure 4. Trajectories belonging timedelta heuristic

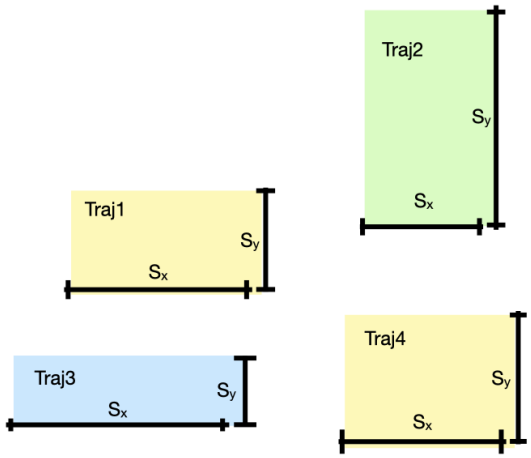


Figure 5. For spreaddelta heuristic $Traj1$ and $Traj4$ are similar because they spread a similar amount of area

As feature extraction, I considered that the work done for the heuristic algorithms could return useful for clustering algorithms. In particular $TIMEGAPS$, $SPREADS$ and $EDGES$ sets can be used as new features for my data. Therefore, I integrate my data features with the heuristic values and I run the pipeline of *Standardization*, *Normalization*, *PCA* and than clustering algorithms. In particular, I tried to perform this pipeline with space and time features together, with only space features, and then with space and heuristic features together. Moreover I tried to perform clustering algorithm on the whole trajectories and even on one city at a time.

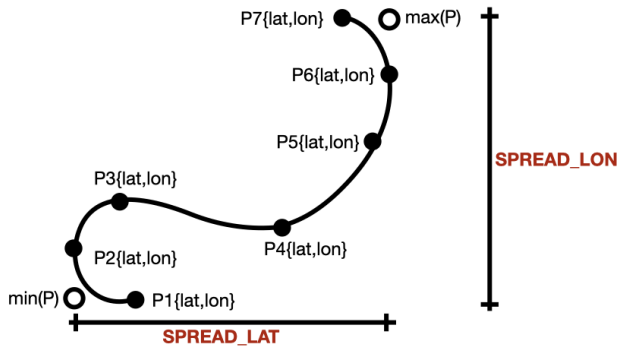


Figure 6. Spreads calculus in trajectory

The feature extraction algorithm for Deep Clustering is the behavior feature extraction. The key idea of the behavior feature extraction is to utilize a sliding window to traverse the records and extract features in each window. As shown in 9, with a sliding window, we aim to obtain space- and time- invariant features to describe the moving behaviors of the object.

Let L and $offset$ denote the width and the offset of the sliding window, respectively. While classic methods often choose $offset = L$, a finer granularity of $offset = 1/2 * L$ can effectively lead to better performance. The moving behavior changes can be reflected by the differences of the attributes between two consecutive records. Let us consider a window with L records. The records in this window are denoted as $W = (p_1, p_2, ..., p_i, ..., p_L)$. Assume the attributes

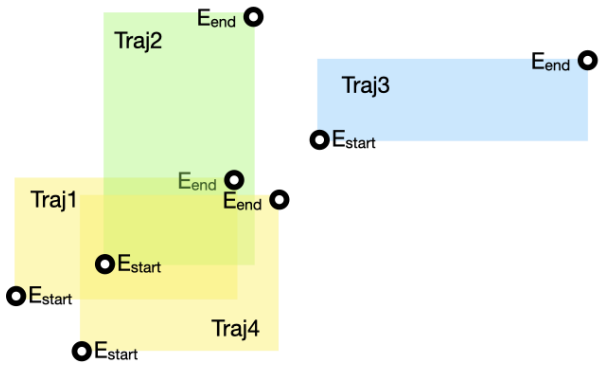


Figure 7. For edgedelta heuristic $Traj1$ and $Traj4$ are similar because they start and finish nearly

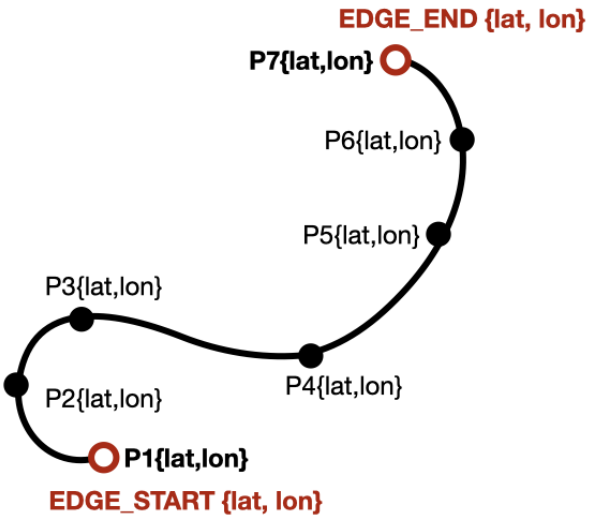


Figure 8. Edges calculus in trajectory

in each record consist of speed and rate of turn (ROT). The extracted attributes for the moving behaviors include: time interval $\Delta t_i = t_{p_i} - t_{p_{i-1}}$, change of position $\Delta lat_i = lat_{p_i} - lat_{p_{i-1}}$ and $\Delta lon_i = lon_{p_i} - lon_{p_{i-1}}$, change of speed $\Delta s_i = s_{p_i} - s_{p_{i-1}}$ and change of ROT $\Delta r_i = r_{p_i} - r_{p_{i-1}}$, where i ranges from 2 to L.

In this way, a window with L records has $L-1$ moving behavior attributes of kind $(\Delta lat, \Delta lon, \Delta s, \Delta r)$. If $L => 1$, for each i from 1 to $L-1$, we compute Δt_i, Δlat_i, Δlon_i, Δs_i and Δr_i. Further compute the change rate of these features $f_i = (f_{\Delta lat_i}, f_{\Delta lon_i}, f_{\Delta s_i}, f_{\Delta r_i})$ in which $f_{\Delta lat_i} = \Delta lat_i / \Delta t_i$, $f_{\Delta lon_i} = \Delta lon_i / \Delta t_i$, $f_{\Delta s_i} = \Delta s_i$, $f_{\Delta r_i} = \Delta r_i$. For two consecutive records, $f_{\Delta lat_i}$ and $f_{\Delta lon_i}$ stand for the average speed, $f_{\Delta s_i}$ stands for the change of speeds and $f_{\Delta r_i}$ stands for the change of ROTs. After computing these features in each pair, we get a

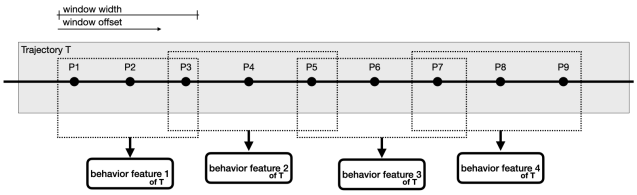


Figure 9. Moving behavior feature extraction algorithm

feature set $f = f1, f2, ..., f_{L-1}$. We use the statistic of f to generate the features in the sliding window. Here, six statistics $mean, max, 75\%quantile, 50\%quantile, 25\%quantile, min$.

In summary, the moving behavior features of each window has $4 * 6 = 24$ dimensions that consist of $f_{\Delta lat}, f_{\Delta lon}, f_{\Delta s}, f_{\Delta r} \times mean, max, 75\%quantile, 50\%quantile, 25\%quantile, min$. If $R = 0$, skip this window. Algorithm 1 shows the generation procedure of moving behavior feature sequence. For each trajectory in T, we generate the moving behavior sequence for it. Then, we put these sequences in a set and denote it as $BS = BTR1, BTR2, ..., BTRN$. Finally, we normalize each feature to prepare for the next sequence to sequence auto-encoder layer.

D. Clustering

The clustering algorithms that I used are the following:

- *K-Means*: choose centroids that minimise the *inertia*, or *within cluster sum of squares (WCSS)* criterion.
- *Mean Shift*: discover blobs in a smooth density of samples with the purpose to find the mean of points within a given region.
- *Gaussian Mixture*: implements the *expectation-maximization (EM)* algorithm for fitting mixture of Gaussian models.
- *Full and Ward Hierarchy Agglomerative*: hierarchical clustering using a bottom up approach and minimizes the maximum distance between observations in pairs of clusters (Full) or minimizes the sum of squared differences between all clusters (Ward).

E. Deep Clustering

An autoencoder is a type of deep neural network used to learn efficient codings of unlabeled data (unsupervised learning). The encoding is validated and refined by attempting to regenerate the input from the encoding. The autoencoder learns a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore insignificant data (Figure 10).

The LSTM autoencoder models used are the following:

- *Simple Autoencoder*: the model is composed by an LSTM that which acts as encoder and another LSTM that acts as decoder. Initially the state of the encoder LSTM is randomly initialized. During training process, the output

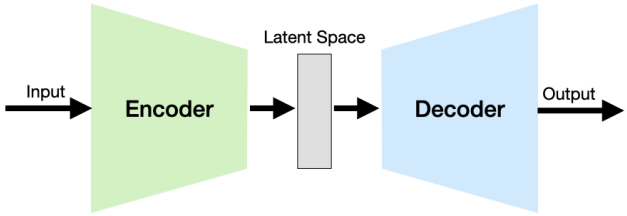


Figure 10. Autoencoder main principle

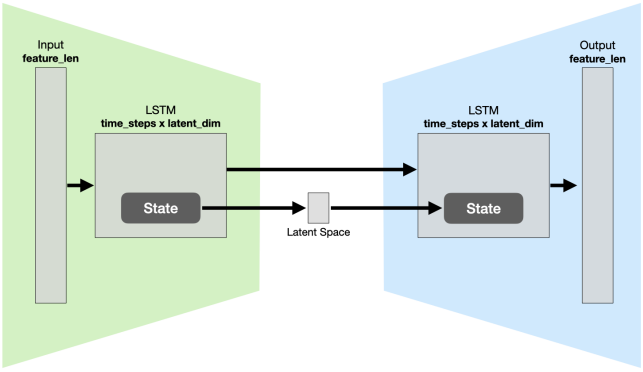


Figure 11. Simple Autoencoder schema

generated from the encoder is passed to the decoder and the LSTM decoder state is initialized with the encoder one (figure 11).

- *Autoregressive Autoencoder*: the encoder LSTM reads the input sequence sequentially and the hidden state $state_i$ is updated accordingly. After the last position of the trajectory is processed, the hidden state $state_t$ is used as the representation for the whole sequence. Then, the decoder first generates the output by taking $state_t$ as the initialized hidden state of the decoder LSTM and the last output of the encoder as the first input of the decoder, and then further generate the other output taking as input the previous output, so as to form the following autoregressive model 12.
- *Addons Autoencoder*: the model is the same of the previous one, but the decoder part is substitute with an already implemented decoder contained in *TensorFlow Addons* library. The decoder is named `BasicDecoder` and it is trained with the `TrainingSampler` that reads output distribution of the current decoding step and pass it to the next decoding step (figure 12).

The target of the decoder is to reconstruct the input sequence. In other words, the encoder LSTM and decoder LSTM are trained together by minimizing the reconstruction error.

The input target of the LSTM autoencoder models could be the moving behavior features but also the raw positions features. In fact, a fundamental characteristic of the neural networks is to learn well the trend of the training inputs,

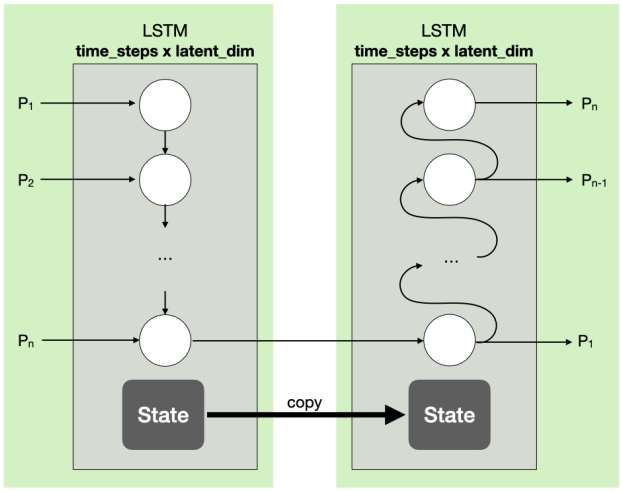


Figure 12. Autoregressive Autoencoder schema

consequently in this case it is not essential to apply a feature transformation algorithm when there are already many.

In order to reproduce the sliding window behavior when raw positions are used, the training is performed creating a dataset of sliding windows over the input timeseries trajectory. Then, the input shape for the autoencoder becomes $batch_size \times number_of_sliding_window \times sliding_window_width$.

V. Results

The best way to show the clustering results is plotting the trajectory with different colours in relation to the cluster which it belongs. In addiction, for clustering algorithms, I used a method of interpretation and validation of consistency within clusters of data, called *Silhouette*.

Silhouette clustering validation technique measures how similar an object is to its own cluster compared to other clusters. The silhouette ranges from -1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighbouring clusters.

The results of heuristic analysis are the following:

- *timedelta heuristic*: in figure 13 you can see some rentals with their positions divided in subgroups of trajectories. That sub trajectories are built considering only the time gaps between the positions, no latitude or longitude has been used to produce this result. This result shows how the time sequences could be really useful to produce a good trajectory partition.
- *spreddelta heuristic*: in figure 14 you can see how the trajectories are clusterized in 3 main groups: wide area trajectories (green), medium area trajectories (blue) and small area trajectories (red).
- *edgedelta heuristic*: in figure 15 you can see how the trajectories are clusterized in relation to the start and end positions of each trajectory generated by *timedelta heuristic*.

For clustering algorithms, in addition to graph analysis, I also use the *Silhouette* validation. To evaluate the number of

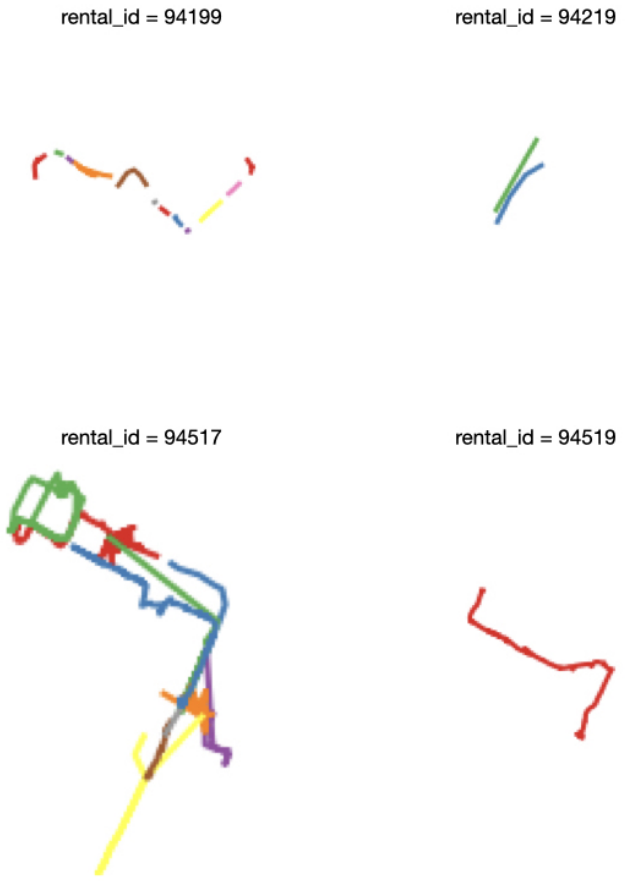


Figure 13. Timedelta heuristic line plot in details

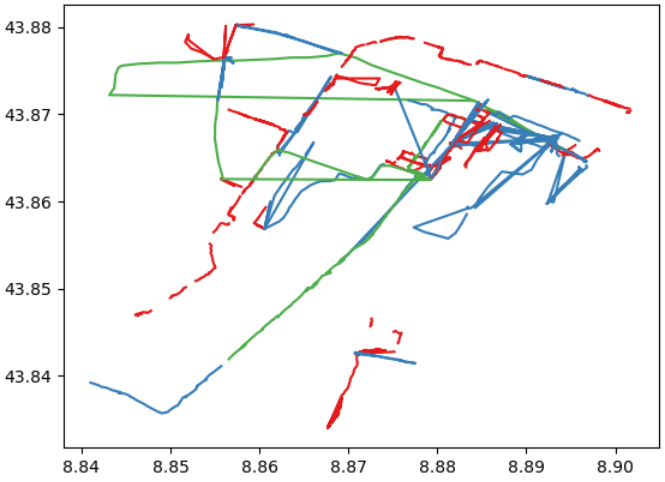


Figure 14. Spreaddelta heuristic line plot

clusters for the algorithms that need it as parameter, I executed *K-Means* for a number of cluster in a range from 1 to 30 and I calculated for each cluster number the *WCSS error*. Then I plot the *WCSS* graph and I used *Elbow method* to choose the best value to use. As result of *WCSS* graph, I chose to set 5 clusters.

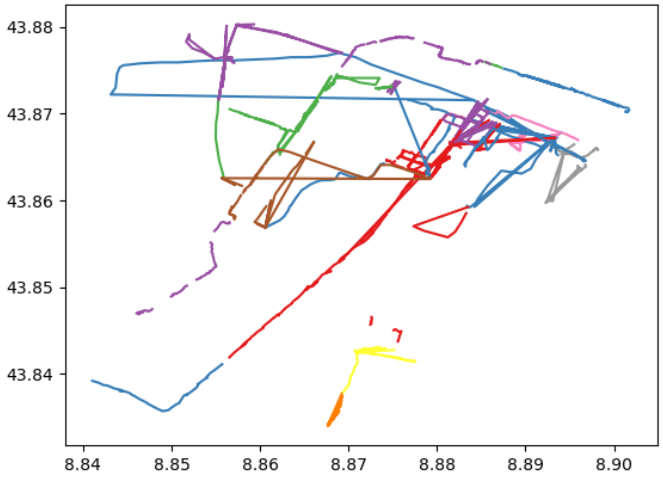


Figure 15. Edgedelta heuristic line plot

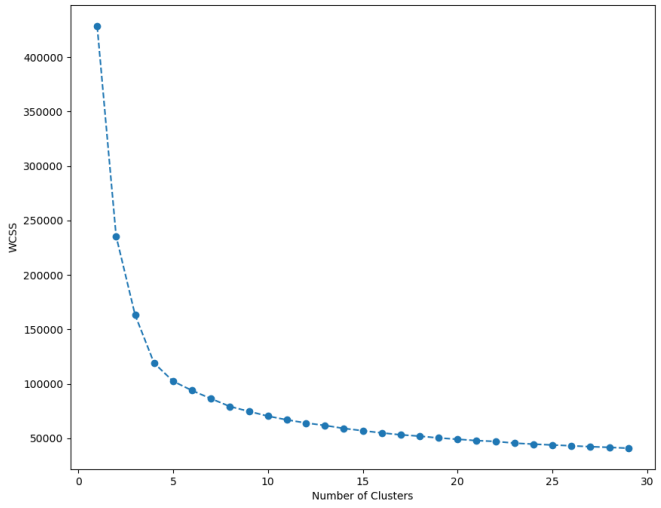


Figure 16. Within Cluster Sum of Squares (WCSS) graph for Elbow method in range 1 to 30 with K-Means

All clustering techniques were performed on each city individually, in order to facilitate the algorithms. The results of clustering techniques are the following:

- *Gaussian Mixture*: performs bad result **??** and the silhouette validation confirms it (silhouette -0.02).
- *Mean Shift*: obtains the best result in terms of silhouette validation (silhouette 0.40). The number of cluster generated is only 3 because *Mean Shift*, at the end of algorithm, performs a pruning operation of similar centres, taking only the more significant. In general, for trajectory clustering, I would like to obtain more clusters, but this is also an alert that the number of clusters can't be really higher.
- *Full Hierarchical Agglomerative*: worst in silhouette validation term (0.16), but the main issue of this technique is a huge memory cost. The dedrogram of this hierarchical algorithm is shown in the this figure 17.
- *Ward Hierarchical Agglomerative*: performs well because

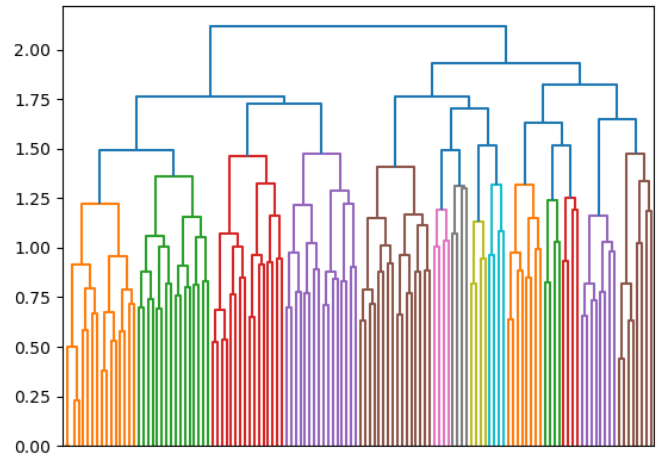


Figure 17. Full Hierarchical Agglomerative dendrogram up to level 5 of merge

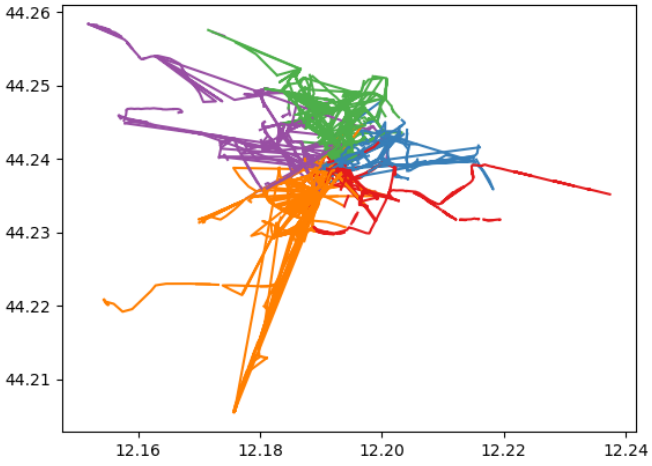


Figure 18. K-Means line plot with silhouette 0.352

it tries to group the centre of the city. It is really similar to *K-Means*, but it is not as accurate as *K-Means* (silhouette 0.28).

- *K-Means*: it is the simplest algorithm but maybe the one that produces the best results. Even if it uses a distance metric to calculate the clusters, it performs very well not only in terms of plot representation, but also in silhouette value. In addition, it is really fast and cheap in memory. The main problem of this clustering result is the mixture of clusters in the city centre, that creates uncertainty.

The deep clustering techniques have been tested with a learning rate of 10^{-3} and from 10 to 30 epochs, but do not produce very interesting results as shown in the figure 19. All 3 autoencoder models have been tested with the use of the moving behavior extration algorithm and produce similar results.

VI. Conclusion

The best result are the one performed by *K-Means*: the feature are extracted with *Standardization*, *Normalization* and

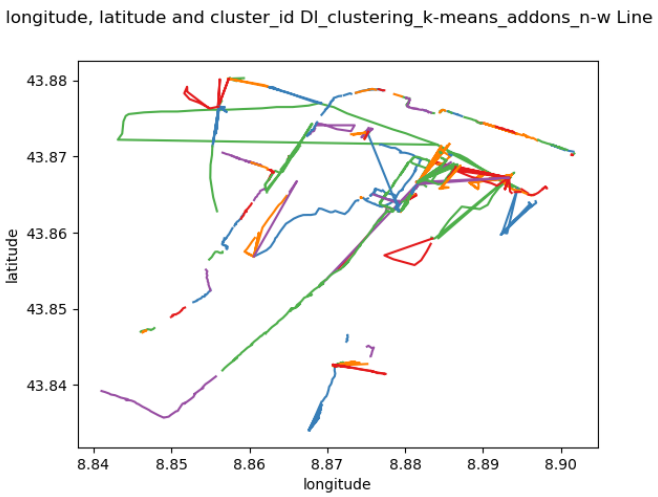


Figure 19. Deep Learning Clustering result with Autoencoder

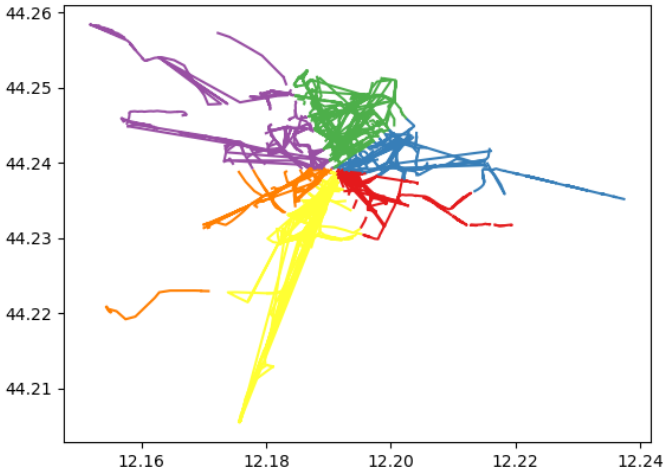


Figure 20. K-Means without PCA with only latitude and longitude features

PCA, the number of cluster given as input is estimated through the *Elbow method* from *WCSS* graph, and the results are given in terms of plot representation and *Silhouette score*.

Moreover, feature extraction without time features produces better clustering result. Time is a feature dimension that brings cluster algorithms to a worse prediction, because the same trajectory could be travelled in different time and instead I want to find the common trajectory independently by time. On the other hand, time is very useful to find subgroups of the same trajectory, as performed by *timedelta heuristic*.

Furthermore, I tried to compare *K-Means* clustering with and without PCA. The figure 20 shows *K-Means* clustering algorithm performed without PCA on latitude and longitude features only. This comparison shows how PCA improve the variance on clustering prediction, and the presence of heuristic features maintains the rental division, on which the heuristic algorithms has been executed.

Unfortunately, the autoencoder model doesn't perform very well. This result is probably due to the fact that in this dataset

every single trajectory is made up of a few positions, therefore, since the model must be trained on every single trajectory independently, the autoencoder is unable to obtain an accurate representation of the trajectory. In fact, in general this deep clustering technique has been tested in big dataset of air flights or of specific road areas such as road junctions.

In order to improve the implementation of the autoencoder models for deep clustering, I suggest the following:

- Test the autoencoder models on well known datasets;
- Test the autoencoder models with more epoch;
- Design more complex autoencoder models (with convolutional layer);
- Trying to augment the number of positions in a trajectory with some techniques, for example adding positions on the line between two positions already existing or following a particular curve given by the inclination the trajectory is taking;
- Improve the edge and coord heuristic applying Mean Shift, in order to avoid the bimodal distribution problem;
- Improve the autoencoder training with specific loss functions focused on clustering;

REFERENCES

[1] G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang, "A review of moving object trajectory clustering algorithms," *Artificial Intelligence Review*, vol. 47, 01 2017.

[2] A. Palma, V. Bogorny, B. Kuijpers, and L. Alvares, "A clustering-based approach for discovering interesting places in trajectories," 03 2008, pp. 863–868.

[3] N. M and P. D, "Time-focused clustering of trajectories of moving objects," *Journal of Intelligent Information Systems*, vol. 27, no. 3, pp. 267–289, Nov 2006. [Online]. Available: https://doi.org/10.1007/s10844-006-9953-7

[4] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: A partition-and-group framework," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 593–604. [Online]. Available: https://doi.org/10.1145/1247480.1247546

[5] R. Nock and F. Nielsen, "On weighting clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 8, pp. 1223–1235, 2006.

[6] M. Yue, Y. Li, H. Yang, R. Ahuja, Y. Chiang, and C. Shahabi, "Detect: Deep trajectory clustering for mobility-behavior analysis," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 988–997.

[7] L. Manduchi, M. Hüser, J. Vogt, G. Rätsch, and V. Fortuin, "Dpsom: Deep probabilistic clustering with self-organizing maps," 2020.

[8] X. Olive, L. Basora, B. Viry, and R. Alligier, "Deep trajectory clustering with autoencoders," 08 2020.

[9] D. Yao, C. Zhang, Z. Zhu, J. Huang, and J. Bi, "Trajectory clustering via deep representation learning," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 3880–3887.

[10] M. M. Kalayeh, S. Mussmann, A. Petrakova, N. da Vitoria Lobo, and M. Shah, "Understanding trajectory behavior: A motion pattern approach," 2015.

Figure 21. Generated dataset ER model