ML Project,
Destination prediction for taxis in Porto

Ziyad BENOMAR et William TIOULONG

Supervised by Yannig GOUDE

February 25, 2022

1 Introduction

In this project, we are interested in a problem of predicting taxi arrival points in Porto. This problem was given in a Kaggle challenge in 2015. The winning teams of the latter proposed remarkable methods to solve it, and some published papers explaining the details of their approaches. Unfortunately, the winning approaches (proposed by teams from large companies) require in their majority a very strong computational power (dedicated machines) and thus cannot be easily reproduced. We have tried to implement some of these solutions and to propose some adaptations to have a reasonable computation time. We have also proposed some methods based on ideas seen in the course and our own understanding of the problem and the data involved.

For our implementations, we used Python and some of its classical libraries (numpy, scipy, pandas, seaboen, sklearn, pytorch, ...). In particular for the visualizations of the map of Porto, we used Folium in some figures and matplotlib + Osmnx in others.

1.1 The Challenge

A full description of the challenge, the data and the rules can be found in [1].
The goal of this challenge is to build a supervised learning model to predict the arrival point of a taxi trip given certain information about the latter. The training data describes a full year (01/07/2013 to 30/06/2014) of the trajectories of the 442 taxis circulating in Porto (Portugal). Each row of our training set gives features about a specific trip:

- TRIP_ID: (integer) unique identifier of the route
- CALL_TYPE: char, indicate the nature of the cab request:
  1. 'A' if the cab is requested from a central cab office,
  2. 'B' if the cab is requested from a specific cab stand,
  3. 'C' otherwise.

- ORIGIN_CALL: (integer) unique identifier of cab callers (recognized from their phone number)
- ORIGIN_STAND: (integer) indicates the stand where the cab is taken if CALL_TYPE == 'B' and NULL otherwise,
- TAXI_ID: (integer) unique identifier of the cab driver performing the trip,
- TIMESTAMP: (integer) Unix timestamp indicating the departure time of the trip,
- POLYLINE: (String) can be interpreted as a list containing the GPS coordinates -Longitude, Latitude) of the cab along its route, taken every 15 seconds
- MISSING_DATA: (boolean) indicating if a location is missing in POLYLINE

On the other hand, the testing set contains the same features except for POLYLINE. It contains only a prefix of POLYLINE instead, that is a list of the first points of the trajectory. The given partial trajectories do not have the same size, and we do not have any information about what they represent for the complete trajectory (its half? a given proportion? some distribution? ...), therefore we can only assume that the cut is random. Of course, we are also provided the arrival point of each trip that is the objective feature.

1.2 The loss function

As we consider GPS coordinates, the natural loss function to consider is the Haversine distance defined by, for any two GPS [Longitude, Latitude] coordinates (λ_1, ϕ_1) and (λ_2, ϕ_2) in radians

$$D_{\text{hav}} := 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Where $R := 6371km$ is the earth radius.

1.3 First Insights

The problem is first atypical because of the nature of the features we are given. While we have some categorical and numerical features, we also have the feature POLYLINE that is a list of variable size. It is clear that it is the most important feature among all the ones we have, but there is a big difficulty on how to deal with it and exploit it. A central question in our analysis and our methods will be to quantify the "similarity" between two trajectories given their prefixes.

The number of features we have is relatively small, therefore we do not need any dimension reduction techniques. However, the categorical features need to be vectorized, for example using one-hot encoding or an embedding, but then we might need dimension reduction because some features might take a lot of different values (ORIGIN_CALL, ORIGIN_STAND, TAXI_ID).

An advantage we have is the very large number of rows in the training dataset ($> 1.5 \times 10^6$ ROWS), it is therefore easier to clean the training data by simply removing all the rows that are incomplete or that present irregularities (and still keep $> 10^6$ rows). However, a hidden difficulty is to deal with such amount of data using limited memory and computational power. For simple models this will not be challenging, but for more advanced models (like neural network), we will need to do the training through many mini-batches.

To facilitate the training, we can also extract some more meaningful features. For example, from the TIMESTAMP, we can extract the day of the week, the hour, ..., which might be more significant than just the timestamp. We can also extract the bearing of the trajectories (angle between the departure and arrival points), that indicates the average direction in which the taxi moved during the trip, ...

As mentioned in our introduction, this challenge have been studied by a lot of very competent teams, and many articles give ideas of some approaches used to solve it. By diving into them, we retained a key idea that came up very often, that is the discretization of the space. Instead of of predicting the arrival point, many papers try to predict to which cluster the arrival point belongs, to which cluster the trajectory belongs, ...

2 Data Prepossessing

2.1 Data Exploration

Frequent Routes and Concentration Points

The proposed challenge, as we have described it, is a very difficult problem: from the few features we have, predicting the final destination of a cab cannot be done with a good accuracy. However, additional information can be found by exploring the different lines of the training set.

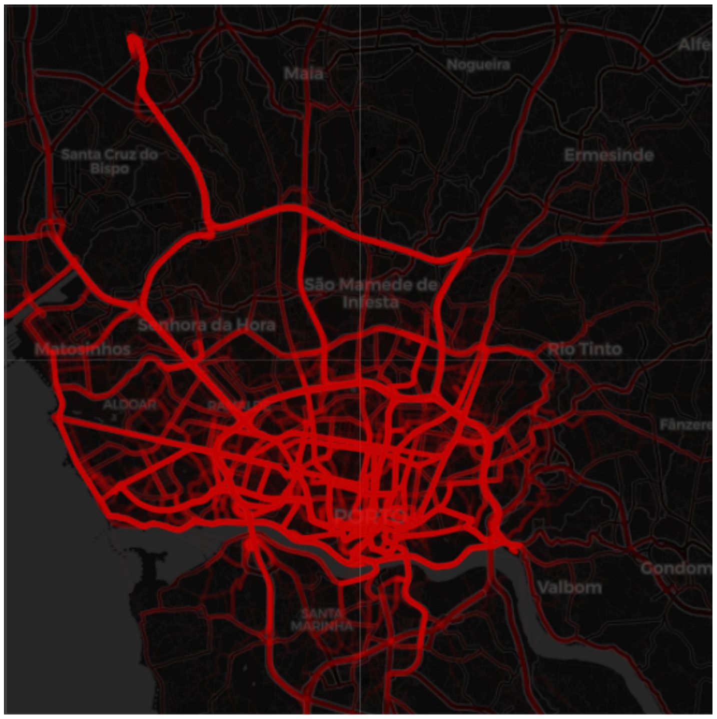

Figure 1: Visualization of all the trajectories in the training set. Trajectories with more intense red color are the more frequented ones.

Starting with a visualization of all the routes in the training set in Figure 1 (the more red the routes are, the busier they are), one can see that some routes are very busy: these are usually the major routes leading from the outskirts of the city to the center, or from the center to specific points outside the city. For example, in Figure 1 we can see a very busy road going to the northwest of the city, its arrival point is in fact the airport of Porto. Finally, as can be seen from the figure, frequent routes are not that numerous, and the models we are going to train should implicitly learn that a cab that has started on a frequent route will probably continue its trajectory on that route.

Having known the most frequent routes, we would also like to know the points where cabs stop most often, or even if the arrival points have a certain distribution on the map. For this purpose, we represent on Figure 2 the arrival points of all the routes of the training set, we observe that they are very concentrated towards the center of the city, but we observe that there are other concentration points, again for example the airport in the north south of the city.

All of this gives us hope that implicit rules can be learned by our prediction models, and that the destination prediction problem we have can be solved to some extent.

TIMESTAMP

One of the most important features is without a doubt TIMESTAMP. The time the journey is made is very strongly correlated to the length of the journey, its direction and its destination. Typically, mid-week trips around 8:00-9:00 am will be to the center of the city, where businesses and directions are

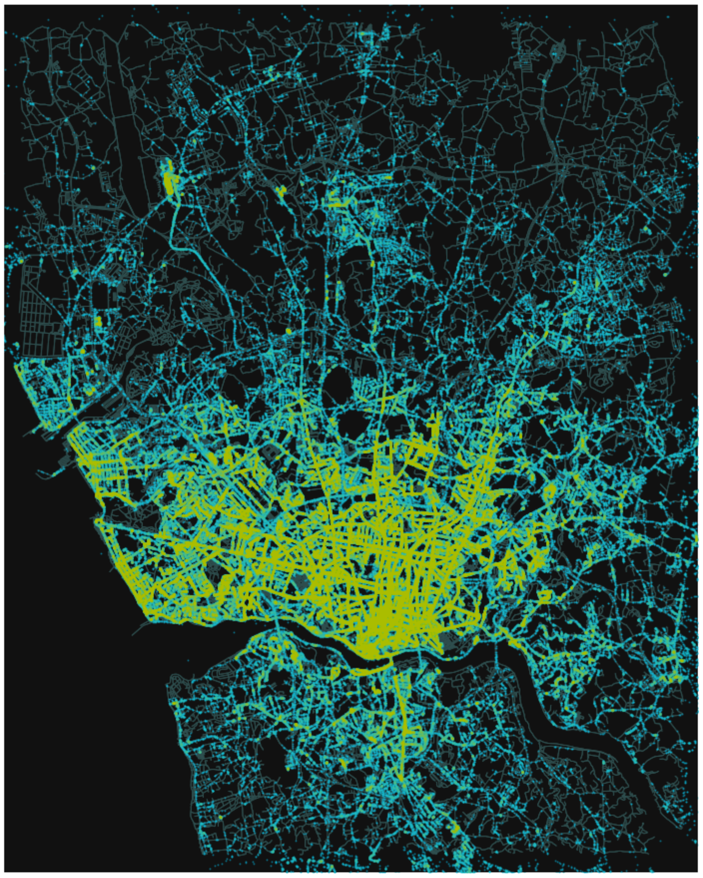
Figure 2: Concentration of the arrival points of the trajectories in the training set. The color varies from none to blue to yellow when the concentration increases.

concentrated, while evening trips will be in the opposite direction; weekend trips will be more random, and will generally be to places where restaurants and other entertainment centers are concentrated.

TIMESTAMP can be exploited to the fullest by correctly labeling holidays, public holidays, events taking place on a certain day of the year and their location etc. We will only extract the week, day and time of day (quarter hour). We will explain better how we use it in Section 2.2.

ORIGIN_STAND

The Origin Stand corresponds to the taxi station from which the customer has called the cab. Thus, the positions of the 63 different stands correspond to privileged starting point for the different trajectories. For the whole 1 710 670 trajectories in the dataset, only 806 579 start at a cab station, i.e. approximately 47 %.

The distribution is extremely unbalanced, with a lot of origin stands pretty unused. Two stands can be enlighted, the stand number 15 and the stand n°57. They correspond respectively to the train station of Campanha and the Downtown subway station which can explain the great traffic at these positions.

ORIGIN_CALL

The origin call corresponds to trips that have been demanded with a phone number. The ORIGIN CALL value corresponds to an identifier corresponding to a unique telephone number. 364 770 trips have been demanded with a telephone which corresponds to 21% of the dataset. The number of distinct identifier is 57 105 which means that in average a telephone number is used for demanding 6 trips. However, the distribution is very unbalanced as we can see in 1. In fact, a single telephone identifier has been used for more than 50.000 trips which we cannot really explain.

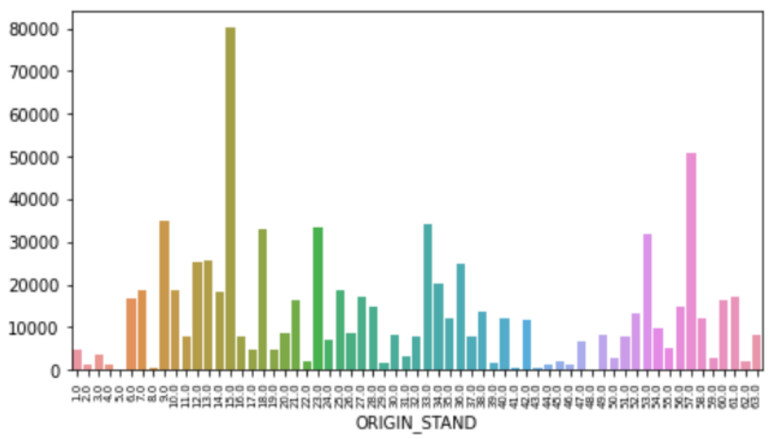

Figure 3: Distribution of trajectories to the origin stand

count	mean	std	min	25%	50%	75%	max
57105	6.38	243.34	1.00	1.00	2.00	4.00	57571.00

Table 1: Repartition of the number of demanded trips by telephone identifiers

TAXI_ID

The TAXI id is a unique identifier given to a cab for a total of 448 vehicles. Again, we can observe a certain variance in the number of trips made by each cab as in 2 even if it is less outrageous than with the origin_call feature.

count	mean	std	min	25%	50%	75%	max
448	3813.45	1644.02	1.00	2694.00	3732.00	5023.00	10746.00

Table 2: Repartition of trips by taxi identifiers

It remains to check if this feature can influence the destination. Can we observe a certain tendency in the destination by only taking a look at the taxi identifier.

In 4, we show the average departure point of the 30 most active taxis (The taxis that have made the most trips). We can clearly see that all the cabs have not the same behaviour and start from different points. We could think that each taxi driver leave in a certain area of the city and thus is more frequently in certain areas. Thus, the TAXI_ID could be a useful feature for our models.

2.2 Data Cleaning

We begin by removing all the rows where MISSING_DATA = True. Next we remove all the rows having trajectories that present irregularities: some trajectories have some missing points, or even capture wrong GPS coordinates at some moment. To get rid of them we compute for each row the speeds between all the successive GPS coordinates in POLYLINE, and we remove all the trajectories where the speed exceeds $200km/h$. By doing so we find ourselves with $1509203 \approx 1.6 \times 10^6$ rows.

A second easy step is to encode the the categorical data by counting the number of values of each one and replacing each value by an index in that range. The mappings between the values and the indices can be stored for example in a dictionary in order to apply the same transformation to the test set.

We also write a function to create from TIMESTAMP three features indicating the week of the year $\in \{0, \dots, 51\}$, the day of the week $\in \{0, \dots, 6\}$ and the quarter hour of the day $\in \{0, \dots, 95\}$. These features give better information than just the timestamp, because we expect some sort of periodicity in the trajectories depending on the day, the hour, ...

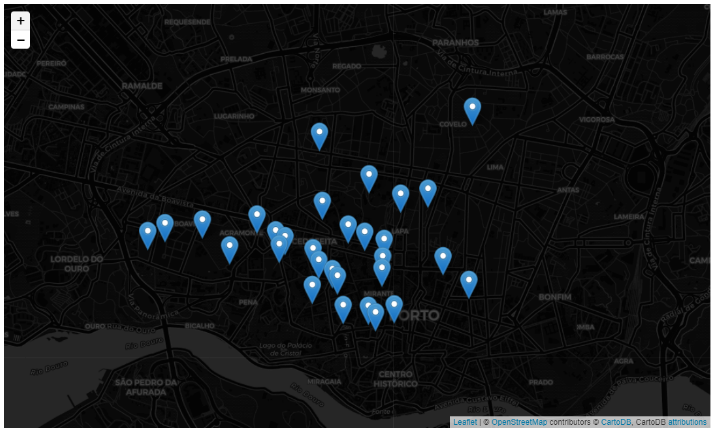

Figure 4: Average departure point for the 30 most active taxis

For the training, we can therefore remove the columns TRIP_ID, TIMESTAMP and MISSING_DATA.

The functions for the cleaning are gathered in the file data_cleaning.py in our code. The code cleans the train set and creates a new file "train_clean.csv". The function "my_encode()" must be called on the test set before evaluating the models we present later, in order to encode the categorical data and create the additional time features.

3 First Approaches

In order to evaluate our results, we present a table of some rankings and scores obtained by teams who participated in the challenge (see Leaderboard in [1])

Ranking	Score
#1	2.03489
#20	2.23584
#100	2.58857
#250	2.98068
#350	4.18867

3.1 Naive model

In this section we will look at a rather naive model, and we will see that by improving it a little bit we get relatively very good results. We will explain how we built and improved this model step by step, and we will see how we can achieve a loss of 2.562 using it, beating 78% of the submissions in the Kaggle challenge.

We start with a rather naive idea, which is simply to give as prediction the last point available in POLYLINE. No learning is necessary, and this strategy on the test set gives a loss of 2.962.

A second idea is to exploit the direction of movement of the taxi. It is true that if we have only a small affix of POLYLINE it is difficult to assert the final direction where the taxi goes, but if the affix is big enough we can have a good estimation of it. What we call direction is in fact the bearing (angle) between the departure and arrival points of the trajectory, it is given by the formula

$$\theta := \operatorname{atan2}\left(\sin(\lambda_2 - \lambda_1)\cos\phi_2, \sin\phi_2 - \sin\phi_1\cos\phi_1\cos(\lambda_2 - \lambda_1)\right)$$

for any two GPS [Longitude, Latitude] coordinates (λ_1, ϕ_1) and (λ_2, ϕ_2) in radians, where atan is the function defined for any $x \in \mathbb{R}$ and $y > 0$ by: with $\varphi := \tan\left|\frac{y}{x}\right|$

$$\operatorname{atan2}(y, x) = \begin{cases} \varphi \operatorname{sgn}(y) & x > 0, \\ \frac{\pi}{2}\operatorname{sgn}(y) & x = 0, \\ (\pi - \varphi)\operatorname{sgn}(y) & x < 0. \end{cases}$$

Roughly, what we want to say is that if the given affix of the trajectory given in POLYLINE is sufficiently large, then we will predict an arrival point that points in the same direction as the bearing between the departure point and the last point in POLYLINE. but still we need to suitably choose a point in that direction. Our training set comes here: we extract the arrival points of all the trajectories in the training set, and ideally we need to make a mean-shift clustering to get the points where the density of their distribution is locally maximal, but it turns out that simply making a mini-batch k-means clustering gives clusters good enough for this purpose, because the zones where points are very concentrated will contain in the end more cluster centers. The advantage of the mini-batch clustering is that it is very fast compared to other clustering algorithms.
We make the clustering on the arrival points with 1500 cluster centers, and we define the prediction algorithm 1

The algorithm contains two hyperparameters m and δ, and a function bearing_err to be defined. What we do in this algorithm is that for a given row of the test set, we get the POLYLINE first, and then if its length is smaller than m (the bearing cannot be estimated correctly) we simply return the last point we have, and if its length is larger than m then we assume that the bearing between the first and last point of POLYLINE estimates correctly the bearing between the first point and the arrival point, and we take all the cluster centers that have a "similar" bearing and we return their mean.
For the similarity between the bearings we should define an appropriate function bearing_err(θ_1, θ_2) that is 2π periodic with respect to $\theta_2 - \theta_1$ and maximal when $\theta_2 - \theta_1 = \pi$. We take therefore

$$\text{bearing\_err}(\theta_1, \theta_2) := \left|\sin\left(\frac{\theta_2 - \theta_1}{2}\right)\right|,$$

Algorithm 1: Naive Bearing Prediction

```
Input  : a row $\mathbf{R}$ from the test set, the cluster centers $\mathcal{C}$
Output: a prediction of the arrival point of the trajectory of $\mathbf{R}$
1  traj $\leftarrow$ $\mathbf{R}$.POLYLINE;
2  $P_i, P_f \leftarrow$ first and last point of traj;
3  if length(traj) $< m$ then
4      return $P_f$;
5  end if
6  $\mathcal{E} \leftarrow \{\}$;
7  $\theta_{\text{traj}} \leftarrow$ bearing$(P_i, P_f)$;
8  for $M \in \mathcal{C}$ do
9      $\theta \leftarrow$ bearing$(P_i, M)$;
10     if bearing_err$(\theta_{traj}, \theta) < \delta$ then
11         $\mathcal{E} \leftarrow \mathcal{E} \cup \{M\}$
12     end if
13 end for
14 return $\frac{1}{\#\mathcal{E}} \sum_{M \in \mathcal{E}} M$;
```

and delta is the error tolerated for this error: we accept to take a cluster center in $\mathcal{E}$ if bearing_err$(\theta, \theta_{P_i M}) < \delta$. Figure 5 shows how the algorithm works.

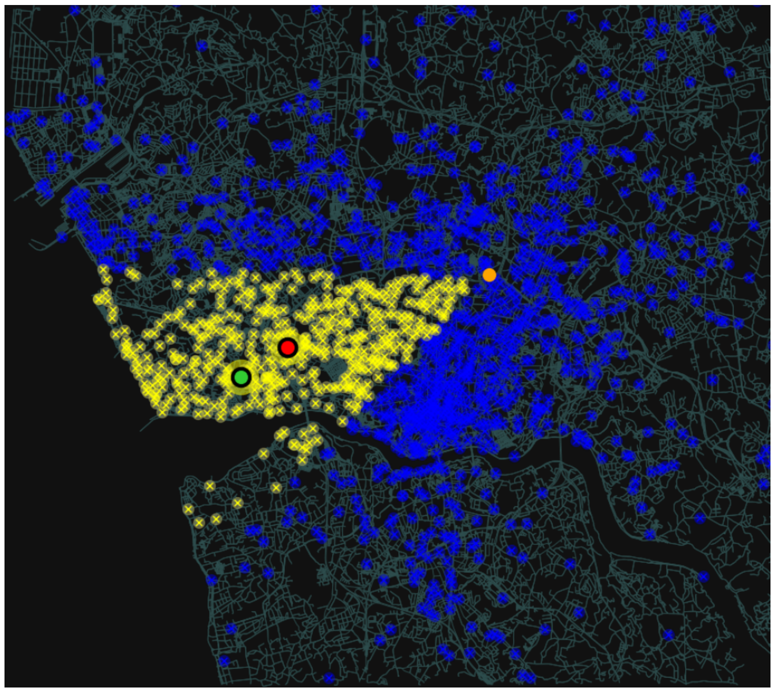

Figure 5: The orange point is the departure point of the trajectory, the green one is its arrival point, and the red one the prediction given by algorithm 1. the points represented by $\times$ are all the 1500 cluster centers. the ones in yellow are those kept by the algorithm and the others are blue. This figure was done with $\delta = 0.3$.

By sampling a validation set of size 10000 from the training set and tuning the hyper-parameters m and δ, we find that the values minimizing the loss are

$$\delta^{\star} = 0.86 \quad \text{and} \quad m^{\star} = 370,$$

and we obtain a loss = 2.564.

In order to improve a little bit more this naive model, we can try to determine the distance between the departure and arrival point of our trajectories. By doing so, we can make a We can push this naive model even further by considering the other features. If we can estimate the Haversine distance L_{hav} between the departure and arrival points of each trajectory, we can put put higher weights on the cluster centers that are at the right distance from the departure point. L_{hav} naturally will depend on many features, in particular on the time features (week, day, hour, ...). We can now try to draw a dependency between those and L_{hav}: while we can think of some elaborate models, we can simply compute compute its mean value for every triplet (week, day, quarter_hour), and this will result in a $52 \times 7 \times 96$ matrix, which can be easily stored and manipulated. It is normal to have a non-smooth dependency with respect to the week and the day, however the dependency with respect to the quarter hour should be smooth. For that, for each couple (week, day) we replace $L_{\mathrm{hav}}(\text{week,day,quarter\_hour})$ by the the average of the two neighboring values, we obtain a smooth dependency as shown in Figure 6. which also avoids overfitting the training set.

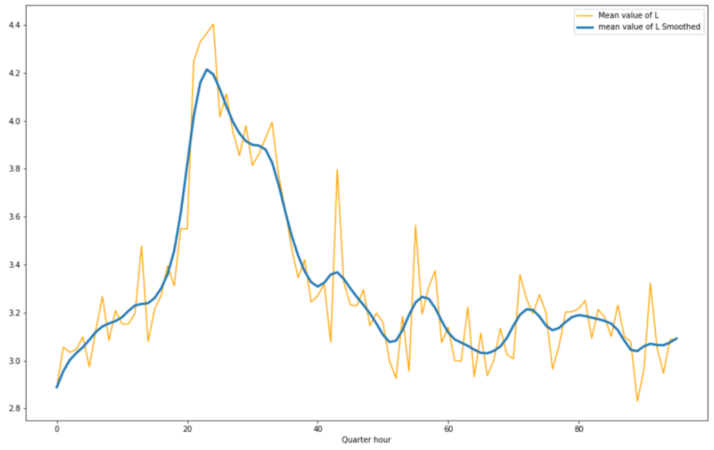

Figure 6: Smoothing the values of $L_{\mathrm{hav}}(\text{week, day}, \cdot)$

Finally, in the output of Algorithm 1, we will return instead a weighted average of the elements of $\mathcal{E}$, where each cluster center $M \in \mathcal{E}$ will have a weight proportional to

$$\exp\left(-\alpha|L_{\mathrm{hav}}(\text{week,day,quarter\_hour}) - D_{\mathrm{hav}}(P_i, M)|\right)$$

where P_i is the departure point of the trajectory. Again, by tuning α using the validation set, we obtain a minimal loss for $\alpha = 3.05$.

Results

Using this algorithm on the test set gives a loss of 2.562, which is better than the team ranked 84th in the competition, and beats $\approx 78\%$ of the teams.
Although this is a naive model, the results obtained are pretty good. More importantly, it is a very strong model in the sense that the training is very fast: all the steps we described earlier (clustering, computing the L_{hav} matrix, ...) takes less than 20 minutes.

3.2 K-nn

Now, in the next two approaches, we try to tackle the problem by constructing a distance between the paths of our dataset. The particular form of the problem leads us to use clustering or nearest neighbor methods to predict the destination. In both cases, building a distance allows us to rely on classical techniques. However, we do not have standard metrics to compute the distance between two paths in $\mathbb{R}^2$.

Hence, we take up the idea of considering the direction of the path as calculated in the previous section and then apply nearest neighbor methods. The intuition is the following: if two trajectories have a similar direction angle and if the last available points in POLYLINE are close, then they should have close end point.

We therefore rely on two parameters to build this metric: the direction and the last point available. The chosen metric is therefore a balance between the two:

$$d(T^1, T^2) = ||p_1^f - p_2^f||^2 + \alpha|\sin\left(\frac{\theta_1 - \theta_2}{2}\right)|$$

where θ_i is the direction of the trajectory T^i computed as in the previous section and α is a balance parameter to trade off between the direction and the final point.

When computing the distance between our training dataset and the new trajectory, we have to choose carefully the target points to compare to the last point available. We cannot choose the final one of course, because the whole metric will be biased and near from the naive predictor that just returns the last point available. Thus, our approach is to compare the last point available to the half trip point of the dataset trajectories. As we do not know where the cut for the prediction has been done (The last point could be as well at the very beginning of the true path and at the very end).

On 7, we see an example of the 5 nearest neighbours computed for the blue trajectory. Thus, we see there is a balance between the distance with mid position (The blue markers are the half trip point, we see that they are concentrated in a close area, because of the first term of our distance.) and the global direction in which the trajectories are going (All the trajectories seem to go to the North-West). Thus, the balance between those two parameters is very important and the parameter α has to be fine-tuned carefully with a grid search.

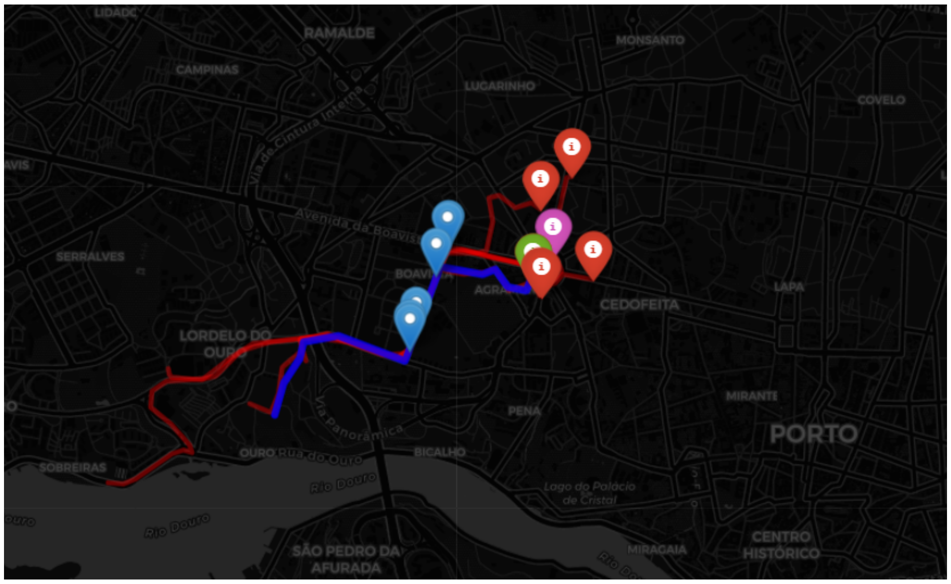

Figure 7: Nearest neighbour method applied to the blue trajectory. In red, the five nearest neighbors for the blue trajectory with the described distance. The green marker is the true destination of the considered trajectory. The red markers are the arrival point of the red curves. The blue ones are the half trip points for the red curves. In purple, the average of all the nearest desination points. Computed with K = 5 and $\alpha = 10$

Then, we can simply take the average arrival point of the nearest neighbours to predict the destination of the target path. We could have chosen to weight this average according to the distance of each of the closest trajectories but we did not make this choice as we did not notice any significant improvement during our tests. Because it was part of the first approaches, we do not manage to explore and finetune more deeply the model. We could have for example fine tune more carefully the K parameter or find a way to add the addtional features into the model.

3.3 Results

Like most nearest neighbor techniques, this technique has the drawback of being difficult to scale for large volumes of data. Indeed, nearest neighbor methods do not require a training phase. On the other hand, for each prediction, the model calculates the distance from the target trajectory to all the trajectories in the dataset, and then has to order them to select the five nearest. We had at our disposal a very large dataset (more than one million paths). It was then difficult to scale our approach to this scale with our means and our knowledge, so we restricted ourselves to a smaller dataset composed of 20,000 trajectories (15,000 for the training, 5,000 for the test).

α	1	5	10	15	20
Mean Haversine Error	2.8651	2.6432	2.6046	2.6293	2.7562

Table 3: Test error for the K-NN model with different values of α, with K = 10

Here we achieved the best performance with $\alpha = 10$. For smaller values of α, the model is reduced to simply predicting the average destination of the closest trajectories to the last point without taking into account the direction. Conversely, if α is too big, only the global direction is considered. With $\alpha = 10$, we found a kind of balance between both, achieving 2.6046 for the loss, which is less well than the previous the model.

4 GMM+Clustering

Let's now describe a more complex method that takes into account the entire trajectory to predict taxi destinations. Inspired by the methods developed in [2], we will proceed in two steps:

- Firs, we classify the trajectories in different clusters in order to enlight some typical trajectories.
- Second, for each of the clusters, a Gaussian mixture model is used to model the set of points in the different trajectories of the cluster.

With these two steps, we can then proceed to destination prediction as follows: With a target trajectory, we can compute the likelihood associated with the Gaussian models in each cluster. We can then compute the weighted average of the destinations of each each cluster to be able to predict the destination.

Then, the final destination is computed by taking the average destination of the K Nearest neighbors.

4.1 The clustering method

In order to get back to the standard clustering methods (hierarchical clustering for example), we need to define a metric between two trajectories. In order to simplify the problem, we first avoid taking into account the directional aspect of the path. Thus, a path going from point A to point B will have to be very similar to a path from point B to point A. The problem is therefore reduced to finding a metric between two curves in R^2, knowing that we have for each of them a discretization at different points. Roughly speaking, the SSPD distance between two trajectories can be understood as follows: for each point of curve 1, we calculate its distance to the second curve, then we average all these quantities. Then, to obtain a distance, we symmetrize the process by doing the same thing with curve 2.

Let's describe this in detail:

Definition 1 *A trajectory T^i is defined as a succession of points $T^i = [p^i_1, \dots, p^i_{n_i}]$ where $p^i_k \in \mathbb{R}^2$ corresponds to the longitude and the latitude of the vehicle at time i with n_i the length of the trajectory T^i*

In order to transform this succession of discrete points of trajectory T^i into a curve in $\mathbb{R}^2$, we use linear completion between each successive points. In fact, the curve $\mathcal{C}(T^i)$ associated to the trajectory T^i is a piecewise linear curve in R^2 composed by segment $[s_1, \dots, s_{n_i-1}]$ where $s_k = [p_k; p_{k+1}]$. For the sake of brevity, we will assimilate T^i and $\mathcal{C}(T^i)$.

Definition 2 (Segment Path Distance) *For two trajectories T^1 and T^2, we define the Segment Path Distance by:*

$$D_{SPD}(T^1, T^2) = \frac{1}{n_1} \sum_{i_1=1}^{n_1} D(p^1_{i_1}, T^2)$$

where $D(x, T) = \min_{y \in T} ||x - y||^2$

Because this distance is not symmetric in T^1 and T^2, we define the Symmetrized Segment Path Distance (SSPD) by:

$$D_{SSPD}(T^1, T^2) = \frac{D_{SPD}(T^1, T^2) + D_{SPD}(T^2, T^1)}{2}$$

Given such a metric we can compute clusters using the distance matrix and hierarchical clustering algorithms. For memory purposes, we have restrained our study to only 15000 trajectories. Indeed, because we compute the distance matrix for the clustering, storing in memory a matrix of size $N \times N$ with $N = 10^6$ is not feasible.

Then for the clustering part, as in [2] we used a hierarchical clustering with complete linkage and 200 clusters.

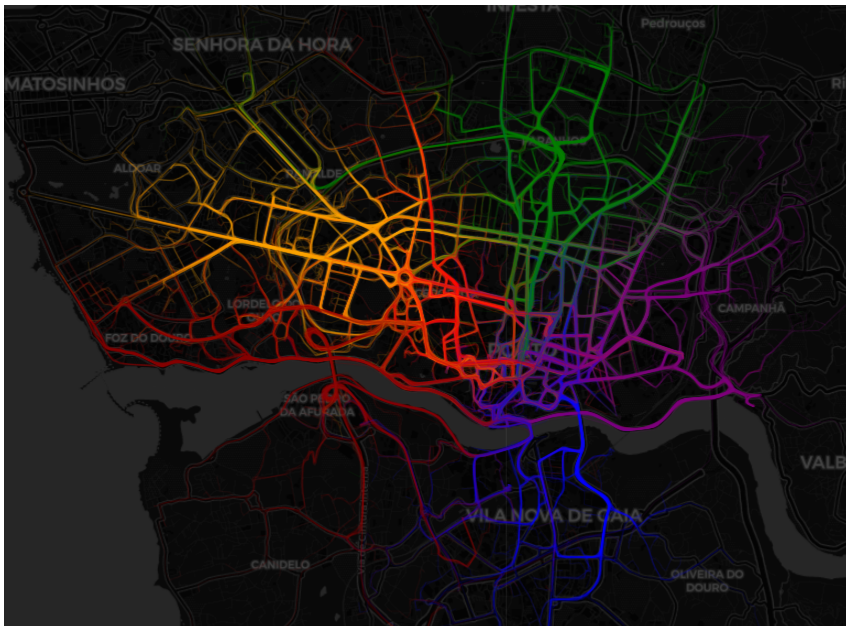

Figure 8: Hierarchical clustering with SSPD distance (200 clusters, 7 most frequent represented)

4.2 Gaussian Mixture Models

After having identified different types of paths within the dataset, we now need to find a way to associate to a path the clusters that correspond the most to it. More precisely, we want to elaborate a statistical model that would give us a score for each cluster. For this, we only consider the set of all the points that compose the cluster. Then, we model each of these set of points by using a Gaussian mixture model. Thus, we can compute the compatibility between a path and a cluster by simply using the likelihood score.

To be more precise, for the cluster $\mathbf{K}$, we consider the set of points $\mathcal{D}_{\mathbf{K}} = \{p | p \in T \text{ with } T \in K\}$ on which we will compute a Gaussian mixture model in $\mathbb{R}^2$ on this set. For a parameter $m \in \mathbb{N}$, we will try to optimize the parameters $\Theta = \{\lambda_1, \mu_1, \Sigma_1, \ldots, \lambda_m, \mu_m, \Sigma_m\}$ in order to maximize the following likelihood:

$$\mathcal{L}(\Theta | \mathcal{D}_{\mathbf{K}}) = \prod_{p \in \mathcal{D}_{\mathbf{K}}} \sum_{i=1}^{m} \lambda_k \phi_k(p) \tag{1}$$

where ϕ_k is a Gaussian distribution on R^2 of parameters μ_k, Σ_k and $\sum_{i=1}^{m} \lambda_i = 1$.

We can now obtain the parameters of our Gaussian Mixture Model on cluster K:

$$\Theta_{ML} = \underset{\Theta}{\operatorname{argmax}} \mathcal{L}(\Theta | \mathcal{D}_{\mathbf{K}}) \tag{2}$$

Thus, doing that for every cluster we have at our disposal a GMM for each cluster of trajectories. We select the optimal m parameter by using the BIC criterion. For the sake of computational issues, we have decided to do the model selection procedure only on a few clusters to select m and generalize the value to other clusters.

We found that in average choosing 20 components for every GMM give the best performance.

Now, in order to associate a trajectory to the different clusters, we can simply use the log-likelihood associated to each Gaussian mixture model. For a trajectory $T = [p_1, \ldots, p_n]$, we can compute for a cluster K:

$$l_K(T) = \sum_{i=1}^{N} \sum_{k=1}^{m} \lambda_k \phi_k(p_i) \tag{3}$$

After doing this with all the clusters, we normalize the scores to make them sum up to 1.

Now, we can also come back to our clustering model and present one of the method we used to evaluate the quality of the obtained clusters.

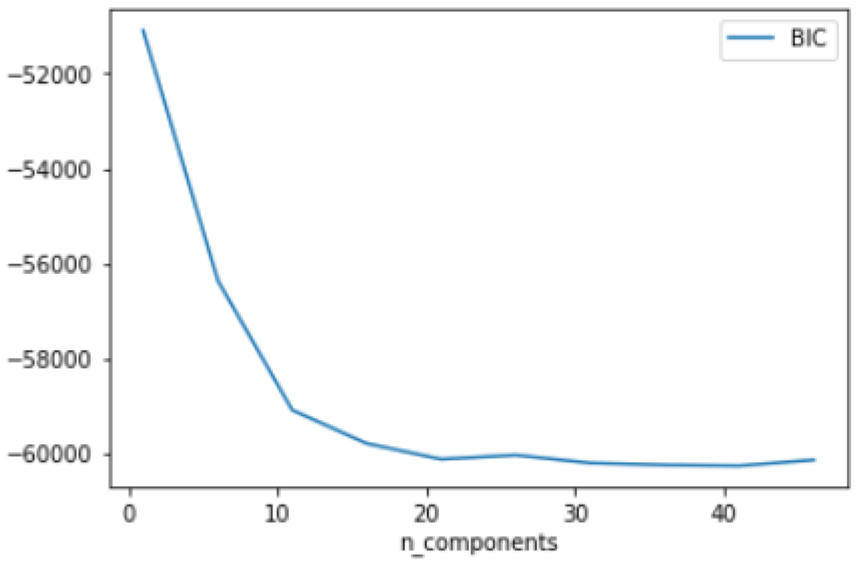

Figure 9: BIC criterion for the biggest cluster (200 clusters)

To ensure the quality of the clustering, we make sure that our cluster scoring method is consistent. Indeed, in our prediction task, we will consider only a prefix of the real trajectory. We want to be sure that the clustering is capturing some kind of path typology. One sound check could be to see if the half path give similar likelihood results than the whole one. It would be a sign of robustness of our clustering. To do so, we used the @K metric, very well known in recommendation system to evaluate this.

First, we compute the K top clusters for the whole trajectory. Then, we compute the K top clusters for the half-trajectory. The precision @K consists in looking at the percentage of common top clusters returned by both computations on the top K clusters.

For example the precision@1 corresponds to the percentage of trajectories where both for the half and the whole trip the top likelihood cluster is the same.

K	1	2	3	5
Precision @K	0.57	0.77	0.87	0.91

Table 4: Precision@K for clustering on 1000 trajectories from the training set with 200 clusters

The Precision@K score are relatively good. For K = 1, the model predict the good cluster on the training set more than half time. It could sound bad at first sight. In fact, it can be explained by the fact that with 200 clusters, with only half of the trajectory, a few clusters are very likely to be associated to it. It is shown by the fact that the Precision@K metric increases fastly when K increases.

4.3 Destination prediction

After describing the clustering and trajectory scoring method, we now describe how we proceed to predict the destination. First, we associate to each cluster an average destination as follows for a cluster K:

$$s_K = \frac{1}{\#K} \sum_{T=[p_1,\ldots,p_n]\in K} p_n \tag{4}$$

The final destination is then calculated by taking the average of the s_K weighted by the log-likelihood of the trajectory associated with each cluster.

$$p_{\text{destination}} = \sum_K w_K s_K$$

where $w_K = \frac{l_K(T)}{\sum_{C \text{ Cluster}} l_C(T)}$

4.4 Adding additional features

The model presented so far does not take into account at all the other features we have at our disposal like the day of the week, or the time of the day. In order to do so, we do the following:

- For the day of the week, we simply separate this feature between the working week (Monday to Friday) and the week-end (Saturday and Sunday) to avoid to much categories that do not capture much information.
- For the hour of the day, we separate the 24 hours into 4 periods of time: 0 pm to 6 pm, 6 to 12, 12 to 18 and 18 to 24.

Instead of generating a new GMM model taking into account those additional features, we will focus on correcting the weighting of the different clusters using these new features. Clearly, if the trajectory to be studied takes place between 6pm and midnight, we will give more importance to clusters containing many trajectories that take place between 6pm and midnight. This technique makes it possible to take this new information into account with little effort. To ensure its feasibility, it is necessary to make sure that the clusters do not have a uniform distribution of the variables considered. In other words, if all the clusters have as many trajectories of each category, the weight correction will have no effect.

To be more precise, we proceede as following. We define for a cluster K

$$\alpha_K^{\text{day}}(d) = \frac{\#\{\text{Trajectories in cluster K happening at day d }\}}{\#\{\text{Trajectories in the dataset happening at day d}\}}$$

and similarly for the hour of the day:

$$\alpha_K^{\text{hour}}(h) = \frac{\#\{\text{Trajectories in cluster K happening at hour h }\}}{\#\{\text{Trajectories in the dataset happening at hour h}\}}$$

We then modify our computation of the cluster weights for a given trajectory T:

$$w_K(T) = \frac{l_K(T)}{\sum_{C \text{ Cluster}} l_C(T)} \alpha_K^{\text{hour}}(h(T)) \alpha_K^{\text{day}}(d(T))$$

où $d(T)$ correspond au jour où s'est produit T et h(T) l'heure.

4.4.1 Weakness of the model

Let's enlight some drawbacks of our process:

First for the clustering part, the choosen metric being very simple, it has the drawback of erasing some important characteristics of the trajectories we enlight here:

- The temporal component is completely erased. The points are considered independently of their temporal position in the path. The temporal aspect is only considered to find the points that follow each other in the path in order to complete the curve linearly. The fact that a car took twice as long to make the same trip will have very little influence on the final metric. Intuitively, this simplification seems reasonable. It reduces the noise that could be generated by the vagaries of urban traffic such as traffic jams for example.
- Secondly, the direction in which the vehicle is travelling is not taken into account. Indeed, for a trajectory T, if we define T^- =by replacing a trajectory $T = [p_1, \ldots, p_n]$ by $T^- = [p_n, \ldots p_1]$, we obtain the same object from the SSPD metric's perspective. This is explained by the fact that $\mathbf{C}(T) = \mathbf{C}(-T)$. Thus, our model treat the same way a trajectory going from A to B and the inverse trajectory moving from B to A. Thus, when predicting the destination, the model could easily predict the destination in the wrong way extending the path from the beginning instead of the end. In order to avoid that, we rely on the fact that typical trajectories are more likely to go in one direction than another (downtown to outside against the reverse).

The first simplification seems to be rather reasonable. For the second, one can try to temper it by adding an additional weighting on the different points of the trajectory. Indeed, intuitively, the points at the end of the path are more important than the points at the beginning of the path to classify it as a typical trajectory.

We therefore introduce a new way to calculate the score of a trajectory with a cluster using the weighting:

$$l'_K(T) = \sum_{i=1}^{N} \omega_i(N) \sum_{j=1}^{m} \lambda_j(K)\phi_j(p_i)$$

where we have chosen logarithmic weights for $\omega_i(N)$.

4.4.2 Results

We present here the several results we obtain for the different variants we mentionned.

Model	GMM	GMM + features	GMM + punderation	GMM + punderation + features
Mean Haversine Error	2.5146	2.4863	2.5021	2.4690

Table 5: Test error for the different variants of SSPD-GMM models (200 clusters with 20 components for every GMM trained on 15000 trajectories

We see that adding the additional features: the day of the week and the hour of the day indeed improve the standard model as we might have guessed with the data exploration. Similarly, adding the logarithmic punderation to the different points of the target trip slightly improve the model. The model has been computed only on a dataset of 15000 trajectories. The fact that we had to compute the distance matrix for all the trajectories severely limit the size of the dataset on which we can work in a reasonable amount of time and space. Thus, it already takes several hours to compute the SSPD distance on all pairs of the 15000 trajectories which show the weak scability of this approach.

We also could have improved the model by computing the BIC criterion on each cluster to find the optimal number of gaussian components. Moreover, here we do not take into account the aforementioned features TAXI_ID, ORIGIN_STAND and CALL_ID even though we mentioned in the data exploration section that those features could capture some information.

5 Neural Network

The solution we explore in this section is the one proposed by the winning team of the challenge, and it is explained in [3].
The neural network structure used is not very complicated. It can roughly be sumarized as follows

- We use the mean-shift algorithm to cluster the set of all the arrival points of the training set, we denote $(M_i)_{i\leq C}$ the obtained cluster centers,
- The categorical and time features are encoded using an embedding,
- POLYLINE is reshaped and adapted to have a same format for all the rows,
- all these features are fed to a MLP with one hidden layer with the activation function ReLU,
- the output of the MLP is a vector with C coordinates,
- it is transformed to a vector $(p_i)_{i\leq C}$ with non negative coordinates adding up to 1 using Softmax,
- the predicted destination point is the average of the C cluster centers with weights $(p_i)_{i\leq C}$: $\sum\limits_{i\leq C} p_i M_i$.

Embedding for the Categorical and Time Features

As we saw in the Data Prepossessing section, we extract from the timestamp three features: week, day and quarter_hour. Since these and the categorical features all take a finite number of values, we make an embedding to simplify their manipulation later. The following table, taken from [3] shows the number of values taken by each of them and the embedding we associate to it.

Metadata	Number of possible values	Embedding size
Client ID	57106	10
Taxi ID	448	10
Stand ID	64	10
Quarter hour of the day	96	10
Day of the week	7	10
Week of the year	52	10

The embedding is included in the neural network, and thus it is learned at the same time as the weights of the MLP.

Dealing with POLYLINE

With no surprise, POLYLINE is again the central feature and the most difficult one to exploit. In order to implement a neural network, and to run it on the test set, we need training features to be the same as the test features, and we can only have numerical features. This is obviously not true for POLYLINE since on one hand the training set provides the whole trajectory while the test set gives only an affix, and on the other hand POLYLINE is a list with a length that differs from a row to another.

To make POLYLINE in the training set of the same nature as the one in the test set, the authors of [3] propose to split each trajectory in all the possible ways, and create a new row for each split. A row where the trajectory contains N points $\{X_1, \dots, X_N\}$ will thus generate N each having an affix $\{X_1, \dots, X_j\}$ with $1 \leq j \leq N$, and all of them having the same values for the other features. With our big training set, this makes an even bigger one (≈ 45 times bigger), and training a neural network using it will require very important computation resources.
To get around this limitation, we have split each trajectory in 5 random positions, which will result in a training set 5 times bigger, but that we could handle.

Then, to solve the problem of the variable size of POLYLINE, we retain from each trajectory $\{X_1, \dots, X_N\}$ 20 numerical entries:

- if $N > 10$, we keep $\{X_1, \ldots, X_5, X_{N-4}, \ldots, X_N\}$
- otherwise, we keep $\{X_1, \ldots, X_N, \underbrace{X_N, \ldots, X_N}_{10-N \text{ times}}\}$.

This way we guarantee having 10 points, each containing 2 numerical values. We could also repeat the first point instead of the last in the case where $N < 10$.

The Hidden Layer and the Output of the Neural network

The authors of the article propose doing a meanshift clustering of the arrival points of all the trajectories in the training set, this gave ≈ 3000 clusters. In our case, we will limit ourselves to 1500 clusters, and we will simply use a K-means clustering instead (much faster) since all that matters in the end is having centroids concentrated where the points are more concentrated.

As we explained before, the feature POLYLINE gives 20 numerical entries, and the other features give 60 entries after the embedding. the MLP in the neural network suggested in [3] contains one hidden layer with 500 neurons, we restrict ours to only 300 neurons with ReLU activation functions, each taking 80 entries and returning 1500 values. the resulting vector of size 1500 is turned into a stochastic vector using Softmax, and the final prediction is the average of the centroids with the weights given by Softmax. Figure 10 shows the structure of the neural network.

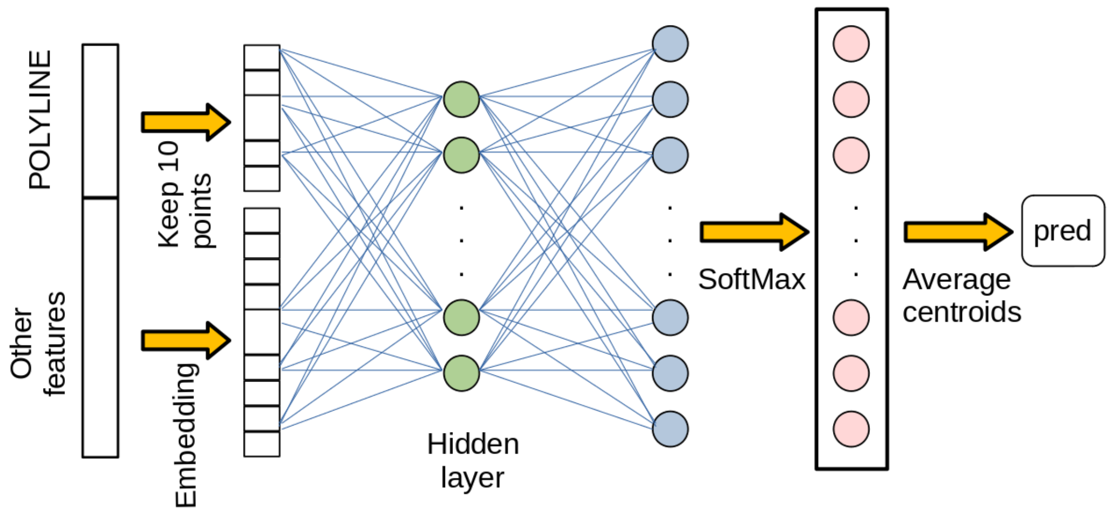

Figure 10: Structure of the neural network

The Loss Function

The authors of [3] claim that the training is done more efficiently using the equirectangular distance defined for any two GPS points [Longitude, Latitude] (λ_1, ϕ_1) and (λ_2, ϕ_2) by

$$D_{\text{equirec}} := R\sqrt{\left((\lambda_2 - \lambda_1)\cos\left(\frac{\phi_2 - \phi_1}{2}\right)\right)^2 + (\phi_2 - \phi_1)^2}$$

We will use for the training this same function. We can easily verify that when $\lambda_2 - \lambda_1$ and $\phi_2 - \phi_1$ are small, D_{equirec} is a good approximation of D_{hav}. Of course, for the evaluation of the model, we will use the the Haversine distance.

Implementation

We implemented this neural network using Pytorch. We faced many challenges due to the large amount of training data we have. While Pytorch is a powerful tool because it enables

Results

The neural network we implemented is more simplified than the one in [3]: it contains less neurons in the hidden layer and the number of clusters we considered is less than in the article, moreover, we only ran the training on 100 epochs, while [3] does not mention how many epochs were taken. We could not run it for more epochs because it takes a long time, not only for the training, but also for just shuffling the data after each epoch (takes up to 10 minutes on our laptops because of the huge training set).
We did the training with mini-batches of size 256, and for the optimization algorithm we used a stochastic gradient descent (SGD) with a momentum = 0.9 and a learning rate = 0.01.

It is normal to expect having a more important loss than the winner team. Figure 11 shows the evolution of the loss over 100 epochs when predicting the destinations of the test set with our neural network.

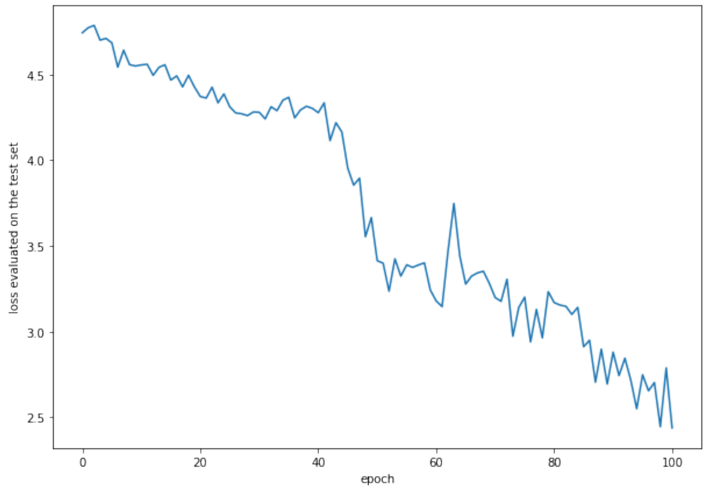

Figure 11: The evolution of the loss evaluated on the test set during 100 epochs

It seems like the loss will decrease even more if we run the training for more epochs, but it not clear if the neural network will converge soon or not.
The final loss we obtain is 2.4968, that would get the ranking 67 in the Kaggle competition.

6 Conclusion

To conclude, we present a brief summary of the obtained results with the best performances obtained with each model.

Model	Naive Model	KNN	GMM	NN
Mean Haversine error	2.562	2.6046	2.4690	2.4968

Table 6: Summary of the results

We have presented two naive approaches and two more subtle models. GMM et NN models have better performance with a slight advantage to GMM. However, we can hope to reach higher performances with neural networks by training on a larger number of epochs. We may also underline the fact that the GMM model has been trained only on a small portion of the dataset (only 15 000 trajectories) and only using the day of the week and the hour of the day as additional features. Thus, the neural network model is more scalable and may also be more adapted to other practical settings like online learning.

Moreover, we can underline the fact that all our work seem to identify two main approaches: the neural network model uses the arrival point of all the trajectories to partition the space into big zones of arrival. Then, the neural network is used to train a function to associate trajectories to these clusters. In some way, it looks like we have divided the city of Porto into a few zones adapted to the problem and the problem is to to assign a trajectory to these different zones. The approach of the GMM model is the opposite one. Instead of clustering the points, we cluster the trajectories themselves. The goal is to enlight some typical paths used by the taxis and then use this typology to make the prediction.

We found on the literature a few variations on those approaches. For example, [4] adopts an approach similar the neural network model. However, instead of using the mean shift or K-means clustering, he uses a KD-Tree partition of the space into discrete regions and then try to predict the right zones with a neural network.

Similarly, [5] use another metric to build a clustering on the trajectories. This method is based on formalizing a Haussdorf distance between two trajectories.

It could thus be interesting to compare these two paradigms more precisely, highlighting the advantages and disadvantages of each.

References

[1] Kaggle comper. https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/overview.

[2] Philippe C Besse, Brendan Guillouet, Jean-Michel Loubes, and François Royer. Destination prediction by trajectory distribution-based model. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2470–2481, 2017.

[3] Alexandre De Brébisson, Étienne Simon, Alex Auvolat, Pascal Vincent, and Yoshua Bengio. Artificial neural networks applied to taxi destination prediction. In *Proceedings of the 2015th International Conference on ECML PKDD Discovery Challenge - Volume 1526*, ECMLPKDDDC'15, page 40–51, Aachen, DEU, 2015. CEUR-WS.org.

[4] Patrick Ebel, Ibrahim Emre Göl, Christoph Lingenfelder, and Andreas Vogelsang. Destination prediction based on partial trajectory data. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1149–1155. IEEE, 2020.

[5] Jinyang Chen, Rangding Wang, Liangxu Liu, and Jiatao Song. Clustering of trajectories based on hausdorff distance. In *2011 international conference on electronics, communications and control (icecc)*, pages 1940–1944. IEEE, 2011.