Université de Liège

ELEN062-1 Machine Learning

Report Project 3 : Taxi's destination prediction

BOXUS Florent
DEMEURE Audric
TEMOSCHENKO Maxime

Teacher: L. Wehenkel
P. Geurts

Academic year : 2023-2024

1 Introduction and Motivations

In this project, we aim to give a succinct but complete description of our problem consisting in predicting the destination of taxis in Porto city. This crucial problem can be generalized to many cities by adapting the approach on the feautures depending the city characteristics. This problem is crucial for route optimization, resources allocation, reduced traffic jams...

This problem is much more complex then it seems to be since we need to deal with a complex dataset. The complexity of the dataset comes from the fact that:

1. The available dataset is huge. There are more than 1,700,000 registered trajectories. One must decides the number of sample to use according to his computer power that is limited. Usually in machine learning, the more data we have the better, but if one can find out a size for the train set from which he is able to make good predictions while he is able to compute it in a reasonnable amount of time, this would be ideal.
2. The dataset is real data. This means it contains many erronous information. Then, we must find a way to deal with it by identifying outliers, extract them from the training set... This is where the smoothing operationswill take place.
3. There are 9 different types of features. One must find out which ones are the most relevant for our goal. This is the role of the preprocessing of the data to extract the most crucial features in order to predict the destination from them.
4. The shape of the features is far from being trivial. The main difficulty lies in the 'POLYLINE' feature since it contains all the coordinates of a trajectory saved every 15 seconds. Because all the trajectories don't have the same length, but machine learning algorithms we will use require a fixed amount of feature, one will have to find the right number of coordinates to use and deal with the trajectories counting less than this number of coordinates.
5. In the point above we mentionned the fact that there was already quiet a lot of difficulty in dealing with the existing features. But nothing prevents us from creating new ones provided they bring usefull information that improve our prediction performance
6. One our data is preprocessed, one must choose the most adequate model to predict the destination among the large number of existing possibilities. Studies must be performed to notice the one with the best accuracy while paying attention to the variance since we don't want a drastically changing prediction.
7. Finally, we must find a way to evaluate our model's performance. Indeed, in the given statement, we were allocated 3 submissions a day. We can't allow us to rely only on these three and we must find out a reliable method. To be usefull, this scoring method must respect the following condition. If the scoring using this method is better, the scoring on the leaderboard will be better.

Taking all of these into account, we will try to have the best score as possible given our limited computing power. The results are computed using the mean Haversine distance since we want to compute the distance between two points given their coordinates(in degrees). The final displayed result is the mean Haversinedistance of all the computed predictions.

2 Preprocessing

2.1 First crucial steps

After loading the train set, one must first consider the number of samples to use. This question is complex since it depends of the used model. We will answer it in the section related to the choice of the model. This operation can't be performed on the test set since we need to predict all the asked trajectories destinations.

Secondly, one removes all the samples which have the 'MISSING DATA'=True, indicating that it's known from the dataset there are missing data points for this sample. One must be pragmatic here since it's not because we have done this operation that our dataset is perfectly cleaned. Indeed, outliers are still present, the data is sometimes badly encoded...

2.2 Smoothing

In this section, we want to remove the outliers coordinates from the 'POLYLINE' column. We analyzed separately latitude and longitude coordinates. The mean and standard deviation were calculated for each trajectory's latitudes. We considered that if a point's longitude was outside 3 times its standard deviation with respect to the mean, the speed to achieve this was too high and do we considered it as an outlier and we decided to remove this coordinate from the trajectory. We applied the same with latitude but this time the ones outside 1.5 were removed. This allowed to remove a majority of points were the speed was too high. We followed the approach described by [3], tested for others values than 3 times and 1.5 timesbut it revealed that 3 was indeed the most suitable value. The graph 1 represents the latitudes before and after smoothing. We can see that some outliers have been removed(the change is too big to be reached in 15 seconds, it will help to build a reliable model later on. Because of this smoothing that remove almost every outliners, we did not manage to use another preprocessing to the 0.1% left of outliners.

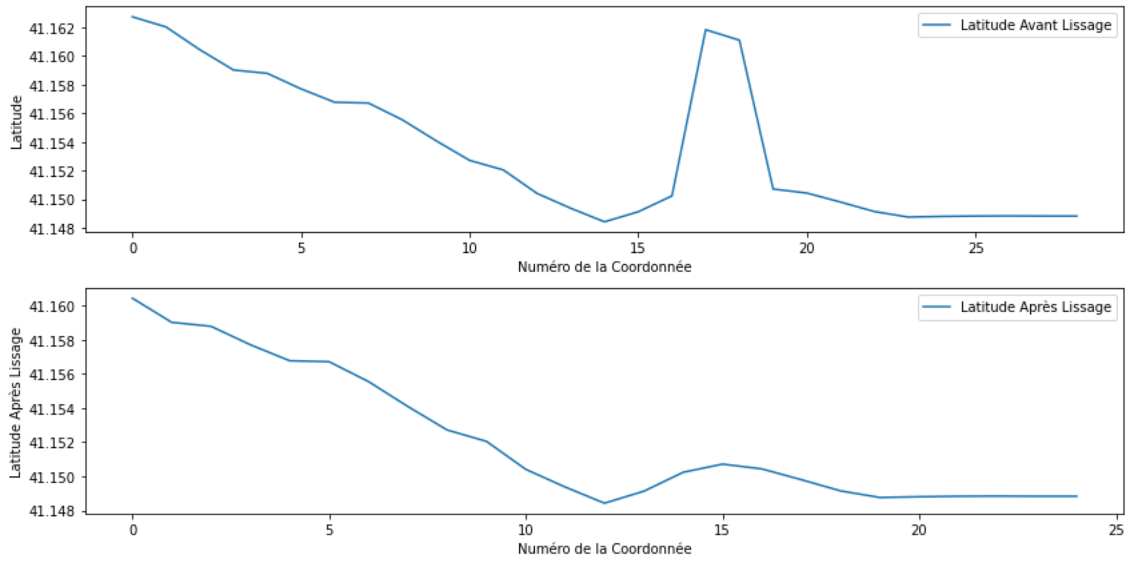

Figure 1: Latitude smoothing

2.3 Protocol of Feature Selection

In thus section, we aimed to describe which features we decided to select and which ones to remove.We have determined empirically the feature set by testing various feature set and comparing the different scores. This was the simplest to implement and it's intuitive. It allows a hands-on understanding of the impact of features on model performance.

We could have decided to go further in the analysis of the feature selection. For example, we could have applied a filtering technique. But even if it's efficient to handle high dimensional dataset, they ignore interactions effects because features are usually considered one by one.

We also interested ourselves in wrapper methods ([5], which are very powerful tools for selecting a subset of relevant features. Those can capture model interactions efficiently, have good performance and are model specific. They are based on dimensionnality reduction to avoid overfitting and enhance model performance.

However, when we tried to implement it, the computation cost was way too high to be executed, so we had to turn back to empirical testing.

2.3.1 CALL TYPE

This feature is giving us the information about the type of trip. This feature was added and removed from the features to see if it improved the prediction of the models and it showed that it was making the prediction slightly better so we kept it.

2.3.2 DAY TYPE

This feature is supposed to represent if we are trying to predict a trip during a working day, a weekend/holidays or a day before these lasts. We removed it since in the dataset, all the 'DAY TYPE' are A. So it was useless to use this column of the test set.

2.3.3 TIMESTAMP

We made big use of this feature, partially to compensate the lack of information from the 'DAY TYPE' column. We divided this in 3 parts like in [4]. One part is 'TIME OF DAY' extracting the quarter of our of the beginning of the trip from 'TIMESTAMP'. One 'WEEK' (from 0 to 52) to represent the week of the trip and finally one 'DAY WEEK' attribute ranging from 0 to 7 to represent the day of the week. This offered significant improvements with respect to simply use the 'TIME STAMP' attribute. After extracting those informations, we remove the 'TiME STAMP' attribute.

2.3.4 POLYLINE

This is the most crucial feature of interest. As we said in the introduction, the main difficulty inherent to it is the fact that its length might vary and what's more, we are only given a partial (but we don't know how much partial is) length of its original one in the test set. We tried several approach for selecting the number and which ones to select. It also depended on the model how we will detail later.

1. For the single vector machine, due to its limitations in terms of computation power, we tried to take the 5 first tuples of coordinates and the fives last tuples after having removed the three last tuples from the 'POLYLINE' column, exactly as they did in the reference [3]. However, unlike the test set they had in the reference, we didn't have complete trajectories with the part we had in the test set, so this featurization was not generalizable to our case.

2. After that, we tried to use the n last tuples of coordinates in the 'POLYLINE' colum in both the train set and the test set. Nevertheless, even if it showed some efficiency in predicting the destination, it was prompt to overfitting. Once more, because the partial cut in the 'POLYLINE' column didn't was not the whole trajectory coordinates like in the train set (50), it was not generalizable for our goal

3. Finally, we tried to get as close as possible from the test set with our training set. So we decided to go for the following procedure: for each trajectory from both the train set and the test set, we take a random number x from one to the length of the trajectory, and we take the x first tuples of coordinate. But because we need a fixed number of features called n, we apply padding of the last tuple at the end of the cutted trajectory until we have the right number of features. If the trajectory after the cut was still bigger than the target length n after the cut, we take only the first n tuples of the cutted trajectory. With this approach that showed the best results for us, we needed to determine how to choose the value n. The table 1 represents the error using our scoring that will be explained later for many n values, we logically choose its minimum that correspond to the average length of the trajectories in the train set whatever the model we used (except for the SVM we will explain later on why)

n	Score
5	2.109
10	2.095
20	2.048
50	2.04
60	2.08
80	2.07
100	2.12

Table 1: Score with target length n

2.3.5 Direction

The first feature we decided to add is the direction.We define the direction using the first and the last tupple or coordinates of the trajectories (before we cut and padd them). It can be computed using the Haversine formula:

$$\theta = \arctan 2\left(\sin(\Delta\text{lon}) \cdot \cos(\text{lat}_B), \cos(\text{lat}_A) \cdot \sin(\text{lat}_B) - \sin(\text{lat}_A) \cdot \cos(\text{lat}_B) \cdot \cos(\Delta\text{lon})\right)$$

where $\Delta\text{lon} = \text{lon}_B - \text{lon}_A$. This gives the orienttation between $[-\pi, \pi]$. One must be carefull and make the conversion of the coordinates in degrees in radians to use it. The adding of this feature provided a good improvement with our scoring method.

2.3.6 Speed

Once again, we add a new feature. This time we evaluate the speed between each tuple of coordinates using the following protocol: To determine the speed v between two coordinates $(\text{lat}_A, \text{lon}_A)$ and $(\text{lat}_B, \text{lon}_B)$ over a time interval Δt, we can use the Haversine formula to calculate the distance:

$$\text{distance} = R \cdot 2 \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\text{lat}}{2}\right) + \cos(\text{lat}_A) \cdot \cos(\text{lat}_B) \cdot \sin^2\left(\frac{\Delta\text{lon}}{2}\right)}\right)$$

where R is the Earth's radius, $\Delta\text{lat} = \text{lat}_B - \text{lat}_A$, and $\Delta\text{lon} = \text{lon}_B - \text{lon}_A$.

Then, the speed v is given by:

$$v = \frac{\text{distance}}{\Delta t}$$

For each trajectory, the speed feature is the average of the speed computed between each of its successive tuples of coordinates before we cut and pad the polyline.

3 Scoring method

The goal of this section is to explain how we assessed the model we will later explain. Indeed, we could not afford ourselves to judge which model is the best only on the 3 submissions per day we made. So we implemented our own method. It has to be fast but coherent: if our own score was better then the score in the submission had to be better.

To this end, we must create a fixed size test set. To do so we used the test split function that allowed us to separate the data set of the desired length into a training set and into a validation set. At first, we were directly scoring the model using this test set of varying length since it was a fixed proportion of the initial data set which has a varying length itself. So we fix the validation set size to 500 samples by taking the 500 first samples of the varying length test set. We than loose the variation of the score due to the size of the validation set.

We then preprocess this validation set to make it similar to the test set so there are no missing values in the data set.

We than save the destination of the trajectory as the 'REAL' column. After that we keep in the 'POLYLINE' feature only the first half of the tuples coordinates. We did that because the test set that we try to estimate only has a pat of the coordinates, and we don't know which part. So we assumed that on average half trajectories were kept and the other thrown so we implemented it that way.

Finally we preprocess our test set in order to predict the destinations (the 'REAL' column) and we predict our model. We then use the provided Haversine function and we take its mean as a final score.

The goal of this scoring method has been succesfully achieved since it was representative of what we submitted. Indeed, if our own score was reduced, the submission score was also reduced. Note that our own score was more optimistic than the score in the submission (We were around a mean Haversine distance of 2.05 km with our score while the score on the leaderboard was 2.7 km).

n	Score
10	2.42
11	2.38
12	2.389
13	2.442
14	2.4
15	2.501
16	2.36
17	2.32
18	2.32
19	2.35

Table 2: Score with number of clusters k

4 Model Analyzed

4.1 Support Vector Machine approach

In this section, we tried to follow the same approach than in the reference [3]. This approach only makes use of the 'POLYLINE' feature. Indeed, they took the 5 first tuples and the five last tuples until the 3rd last tuple.

After that, they partition the destinations in 15 areas defined by their centers coordinates. To do so, they needed to cluster the data. We used the kmeans as clustering algorithm. The main goal of K-means is to partition a dataset into K distinct, non-overlapping subsets (clusters), where each data point belongs to the cluster with the nearest mean. But we need to tune the k parameter to do so, we follow the guidelines given in the course by [2]: We locate locate the knee in the intra-cluster variance curve. Unfortunately as shown in figure 2, such a point doesn't exist, or at least we can't see it graphically. We must find another way to choose it. There is no way to perform cross validation here as we saw in the course! Choosing a value of k has the same effect than underfitting the model while choosing a k too high is equivalent to overfitting. So we tried empirically to determine the optimal value of k with k ranging from 10 to 20 values (higher values could give a better score but it would be overfitting the data set and harder to genelarize to the test set, and we observed a minimum value for k=15. Some representative values with our scoring method are given in the table. Please note that we train the data we used for training the model and testing the model is 15000 samples, with a test rate of 0.1 but limiting ourseleves to the first 500 lines as explained earlier. The table **??** shows us that the optimal value for k is not so obvious to find since the score is fluctuating with the value of k.

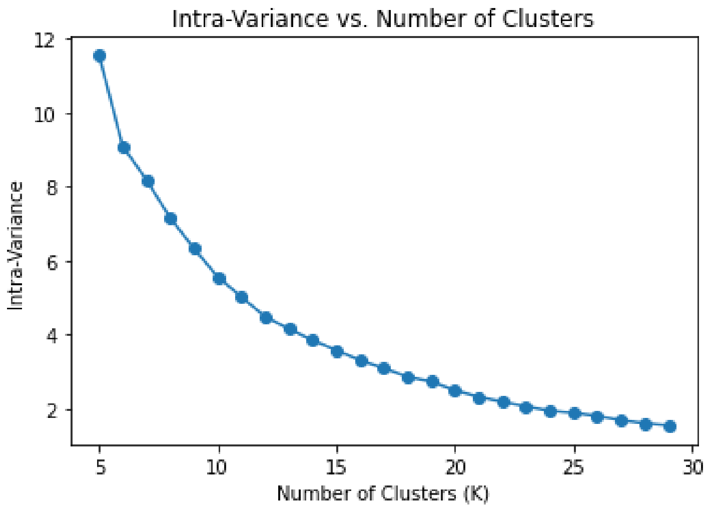

Figure 2: Intravariance vs k

We can also try several kernel with k = 15 like in the paper to determine the best one to use without changing the other parameters. We can see in table 3 that the kernel that exhibits the best score is the linear one, we found the same result than in the paper.

The advantages and drawbacks of this approach are:

Linear	2.4011
Polynomial	2.5192
RBF	2.4051
Sigmoïd	4.0517

Table 3: Score with different kernels

m	score
10	2.64
50	2.21
100	2.13
370	2.1

Table 4: Score for different values of m

- Instead of explicitely defining a mapping ϕ in which the data is linearly separable, we directly compute the dot products, making it less computationnaly heavy.
- It's less prone to overfitting in high dimension.
- Effective in small or medium datasets. However, it's not the case here. It's a limitation of this model.
- It's sensitive to outliers and to noise. That's why the preprocessing steps,especially the smoothing, are crucial.
- The choice of the parameters is tricky. Here the main difficulty lies in choosing k for the clustering as we saw.

4.2 Naive Model

We also implemented the naive model explained in the paper [4]. This model uses two key aspects: Clustering destinations and using the direction as a main feature. One must first compute the direction of each trajectories in the train set using the haversine formula as we explained in the preprocessing steps. After that, we cluster all the different destinations. In this paper, they chose a k for the kmeans much higher than we did in the SVM attempt.This is because this will be our only feature for the model to predict the final destination. The clustering is performed on the destinations of the train set. For each trajectory in the test set, one follows the next steps:

1. If the length of the trajectory is shorter than a threshold value m, we return as a prediction the last tuple of coordinates of the POLYLINE feature
2. Otherwise, we calculate the direction of this trajectory yusing the start and end tuples of the POLYLINE.
3. We evaluate the direction between the starting point of the trajectory and each cluster center. If it is smaller than a treshold angle δ, we store it in ϵ.
4. We return as a prediction for the final destination the average of the cluster centers stored in ϵ

We implemented this model and we were disappointed by its performances. Indeed, in the paper, they used $m = 370$ but this was very restrictive in our case since the mean length of the trajectories we consider is 50. Using the above algorithm, this will lead to returning most of the time the last tuple as a prediction. The table **??** shows the results for different values of m. Scores are quiet good especially for high m since we just return the second to last coordinate. As m decreases, we see that our score get worse so this method is far from being the best one

4.3 Multi-Layer Perceptron

One tries to implement a multi-layer perceptron with a single hidden layer of 500 perceptrons. One clusters the final destinations because it is difficult to predict the coordinates with such a simple neural network. However, we have encountered difficulties using a neural network : it is difficult to tune , computing power expensive and appropriate usage of such a model is out of the scope of this course. With this simple neural network, one obtains `3.92`. We could have tested various possible layers but as stated before, we had limited computing power so we have decided to focus on simpler methods.

Number of samples	Optimal max depth	Score
10000	12	2.38
20000	12	2.221
50000	12	2.38
100000	14	2.25

Table 5: Scores and optimal max depth with the number of samples

4.4 Decision Tree

This method is simpler to interpret than multi-layer perceptron and less complex, however it was easier to tune the latter and obtain better results. We have applied a grid seach to determine which of the parameters was the best. For the criterion it was straightforward that the 'entropy' was the best choice possible. We decided to perform a gridsearch here to tune the hyperparameters. Here is how it works:

1. Define a grid of paramaters. Here we proposed two options for the criterion:'mean squared error', friedman mse', 'absolute error'. For the max depth hyperparameter, we make it range from 6 to 70 using a step of two. The grid search will test every possible combination of its hyperparameters
2. The cross validation step consist in training the model with the selected hyperparamaters on several subsets of the data.
3. We define a scoring method, we will use the opposite of the haversine distance since the grid search method seeks to maximize the score metrics
4. Return the hyperparameters that maximize the scoring method

The plot 3 shows the gridsearch for 20000 samples, we found that the optimum criterion was 'mean squared error' which was independant from the number of samples. This could have been different if there was a lot of outliers since the mean squared error tends to increase the weights of outliers, but the smoothing function played its role. However, the 'max depth' hyperparameter is dependant of the number of samples. For 20000, we can see on figure 3 that the optimal max depth is 12.

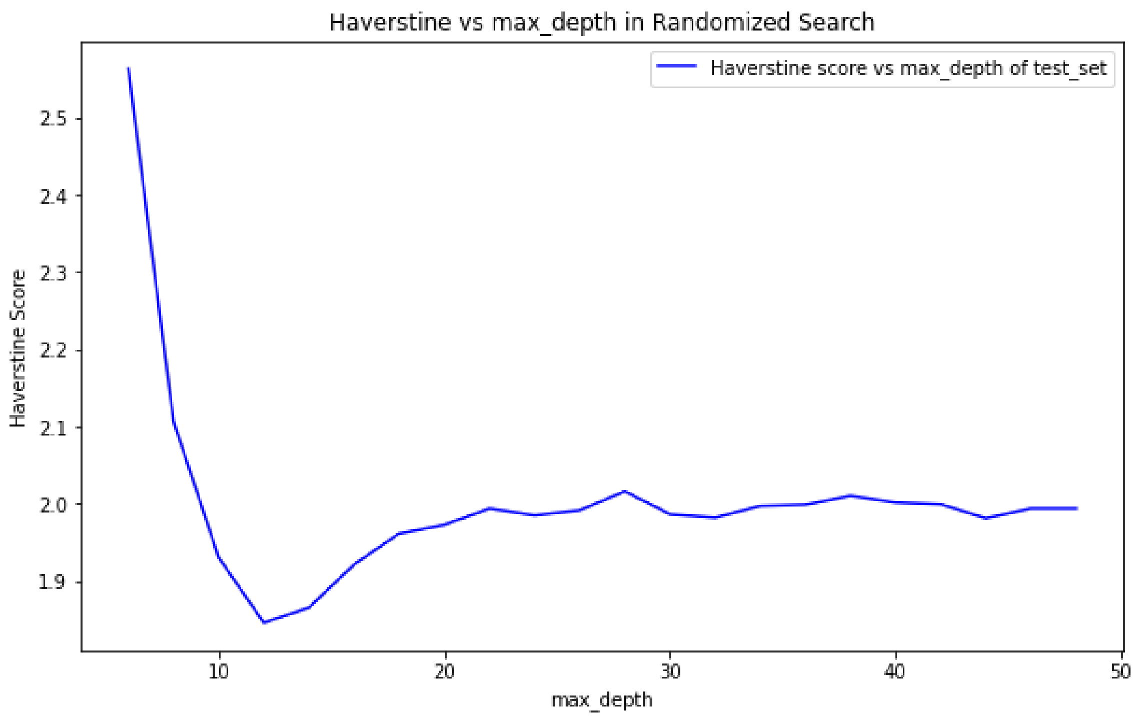

Figure 3: Gridsearch results with a cross validation with 5 folds

The table 5 shows the evolution of both the score and the optimal value of the max depth hyperparameter. The score decreases with the number of samples, and the complexity (lying in the max depth hyperparameter) also increases

4.5 Decision Tree with clustering

In this section, we tried to transform the regression problem into a classification problem, using the same preprocess than before, but we use a kmean clustering on the final destination and we label all the coordinates tuples, that are now represented by a single integer number. We then replace the 'POLYLINE' feauture by 'CLUSTER IN' and we replace the 'END DEST' by 'CLUSTER OUT' that are the values to be predicted by the algorithm. We can finally replace the cluster predicted by the centers of the clusters by conserving the mapping thanks to a dictionnary.

We first used the same preprocess than explained earlier, but we finally used another preprocess method because its scores were much better when we applied this method. However, tuning the number of cluster k was also difficult as we saw earlier in the SVM model. We tried with several model, the best one was the decision tree classifier with the criterion fixed to 'entropy' and max depth to None with a randomized search. We didn't manage to go under the 3km with this prediction so we moved on another method.

4.6 Random Forest

The Random Forest model served as the primary method in this project. To optimize its performance, we conducted parameter tuning using the GridSearch method. The GridSearch was executed in a 4 x 3 configuration, exploring four different max_depth values (None, 15, 20, and 25) and three n_estimators values (150, 200, and 250). The focus just on these parameters because of computional constraint and because these are the most important parameters compared to others. Additionally, we implemented 5-fold cross-validation to enhance the robustness of our model evaluation. Note that this score is not the real score, it is the score of the cross validation.

max_depth	**n_estimators**	**Score**
None	150	1.235
None	200	1.267
None	250	1.233
15	150	1.334
15	200	1.333
15	250	1.296
20	150	1.244
20	200	1.276
20	250	1.277
25	150	1.235
25	200	1.235
25	250	1.283

Table 6: GridSearch of RandomForest

As we can see in this table the best one is the combination of None with 250 estimators.

To illustrate our choice of the estimator, we generated multiple Random Forest models using a consistent sample length of 100,000. The results are presented in two separate plots (due to computational constraints). It's important to note that the scales of the plots are not identical.

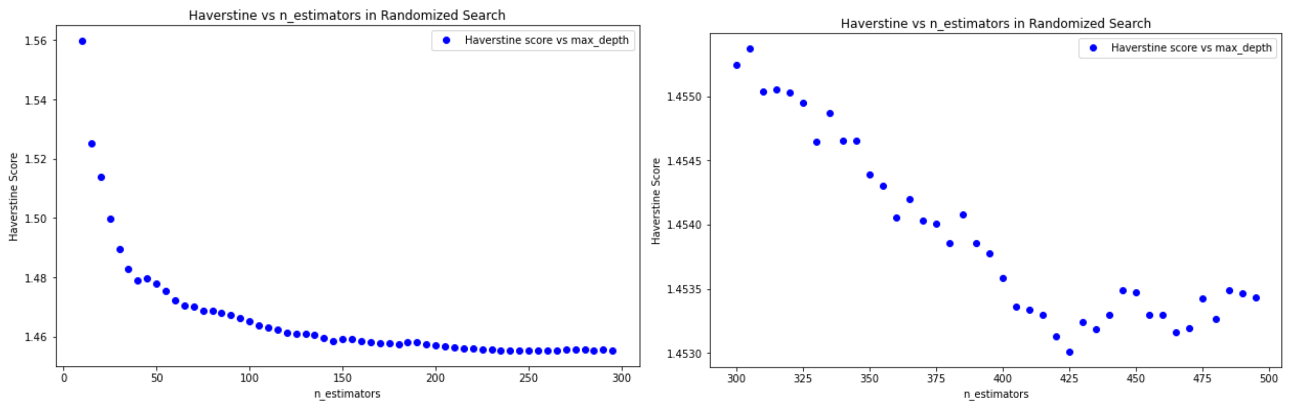

Figure 4: Haverstine vs estimators

The analysis reveals an improvement in model performance with an increasing number of estimators, peaking notably at 425. However, despite this apparent optimal point, we opted against utilizing it in our final model. The decision was based on the trade-off between computational time and marginal gains in performance. Employing 425 estimators considerably extended the runtime, offering only marginal improvement in scores compared to a configuration with 250 estimators. Consequently, we determined that the efficiency gained from using 250 estimators outweighed the relatively modest enhancement in model performance achieved with 425 estimators.

4.7 Other Models

In this section, we mention other models that we experimented with, but did not invest much effort in tuning their parameters. These models include Ridge Regression, AdaBoost, GradientBoosting, and Linear Regression. The decision to explore these models stemmed from their reputation for good performance, although in our experimentation, they did not exhibit superior performance compared to the Random Forest model.

We obtained scores for these models without extensive parameter tuning. The following table summarizes the scores for these models alongside the Random Forest model, providing a basis for comparison.

Model	**Score**
RandomForest	2.07623021429058
KNN	2.664451570232382
Linear Regression	2.6524689317354673
Gradient Boosting	2.104900040709429
AdaBoost	10.223842951820911
Ridge Regression	2.6524616182336733

Table 7: Recapitulation of the score of the models

Thus, from this table, it becomes evident that Random Forest stands out as the optimal choice to commence our modeling endeavors.

5 Conclusion

In this project, one could essentially obtain a better score on three fields : the preprocessing, the studied method, and the parameter tuning.

Especially, the preprocessing played an important role in order to reduce the haversine score while the other parameters were fixed. We have smoothed the data, and determined a optimal set of features empirically. One can notice the following set of features : division of timestamp into quarter of an hour, fixed random subsequence of polyline with associated padding and the addition of the direction and the speed in the feature set.

The method studied has obviously an important impact on the haverstine score. In this project, we have studied seven methods : SVM, Naive model, Multi-Layer Perceptron, Decision Tree , Decision Tree with clustering and Random Forest.
The limitation of the SVM approach is that it is an efficient method for small/medium dataset meanwhile the dataset of this project contains more than 1 million entries.
The main issue encountered with the MLP was the lack of computing power and the difficulty to tune it.
Decision Tree is a simpler method but we could tune it easily. However, in order to prevent overfitting, we have also explored randomforest which ends up to be the best-scoring method among the list. On the latter, we have tuned the number of growing trees and the `max_depth` associated.

In summary, our project showcases the iterative and systematic approach taken to enhance predictive accuracy using haversine metrics, emphasizing the importance of thoughtful preprocessing, method selection, and parameter tuning in the development of a robust and effective predictive model for taxi's destinations. It shows the significance of these factors in the successful implementation of machine learning solutions for location prediction tasks.

Finally, we summarize all our models with their score in this table :

Model	**Score**
RandomForest	2.07623021429058
KNN	2.664451570232382
Linear Regression	2.6524689317354673
Gradient Boosting	2.104900040709429
AdaBoost	10.223842951820911
Ridge Regression	2.6524616182336733
MLP	3.95
dtree with clustering	3.0
SVM	2.32
Naive	2.1

Table 8: Recapitulation of the score of all models

References

[1] School of computer Sciences, Instructor: Aarti Singh, Sub-lecturer: Mariya Toneva,Machine Learning 10-315,Classification – Decision boundary

[2] Pierre Geurts and Louis Wehenkel, Université de Liège, Institut Montefiore, https://people.montefiore.uliege.be/lwh/AIA/

[3] A comparison of machine learning methods for predicting final vehicle destinations,Sakthi Kumar Arul Prakash, Kyshalee Vazquez-Santiago,Jorge Vásquez-Albornoz, Carnegie Mellon University https://www.researchgate.net/publication/350123930_A_comparison_of_machine_learning_methods_for_predicting_final_vehicle_destinations?fbclid=IwAR0MbFp1fB-h-_wHPrSwRG9pq_KlLzMUpOLvwGEa0IzB1U0iAhqfRwnuntw

[4] Destination prediction for taxis in Porto,Ziyad BENOMAR and William TIOULONG,Supervised by Yannig GOUDE

[5] Understanding Wrapper Methods in Machine Learning: A Guide to Feature Selection,Aris Muhandisin,https://arismuhandisin.medium.com/understanding-wrapper-methods-in-machine-learning-a-guide-to-feature-selection-23f71059abf8#:~:text=Wrapper%20methods%20treat%20feature%20selection,a%20specific%20machine%2Dlearning%20model.