LIÈGE université
School of Engineering

University of Liège

ELEN062-1: Introduction to machine learning

Project 3 - Competition

Authors :
Louis HOGGE s192814
Simon LOUVEAU s194100
Tom WEBER s203806

Professor : L. Wehenkel
Professor : P. Geurts
Year : 2023-2024

Contents

1 Introduction

For this project, we received a dataset that comprises a full year of trajectories for all 442 taxis in Porto, Portugal resulting in a table of 1,710,670 rows. Columns described the `TRIP_ID, CALL_TYPE`, `ORIGIN_CALL`, `ORIGIN_STAND`, `TAXI_ID`, `TIMESTAMP`, `DAY_TYPE`, `MISSING_DATA`, and a `POLYLINE` that contains the trajectories. The ultimate goal was to design a solution that predicts the destination of a taxi trip given its partial initial trajectory.

This report shows how we processed the data (dataset analysis, pre-processing, splitting), which algorithms we used, how we adjusted the hyperparameters, the results obtained and finally a conclusion and remarks on possible improvements.

2 Dataset analysis & preprocessing

As mentioned in the introduction, the dataset is very large. We therefore had to analyze it in order to extract the interesting features and, above all, to get rid of those that were unnecessary. Here's how we did it:

- `DAY_TYPE`: this column contained only A's, so it didn't provide any valuable information. We therefore deleted it.
- `MISSING_DATA`: In order to manage missing data, we simply decided to keep only rows containing the value False and then delete this column.
- `CALL_TYPE`: We've replaced the alphabetical values with numerical values, so that we can take advantage of the information.
- `ORIGIN_CALL` & `ORIGIN_STAND`: As these columns contained NaN values when CALL_TYPE was not B (= 1), we decided to replace them with 0s, allowing us to take advantage of the added information without corrupting the dataset.
- `TIMESTAMP`: In order to give algorithms the ability to discover time-related features, we decided to divide this column into a number of weeks, days and quarters of an hour of the day. We then deleted the original column.
- `DURATION`: We decided to add a column to the dataset showing the duration of the journey in seconds, calculated by multiplying the polyline length by 15, with each gps position recorded every 15 seconds.
- `POLYLINE`: As far as polylines were concerned, the first step was to find a way of reducing and standardizing the number of GPS coordinate points on which our models would operate. To do this, we decided to generate a boxplot of the dataset's polyline sizes.

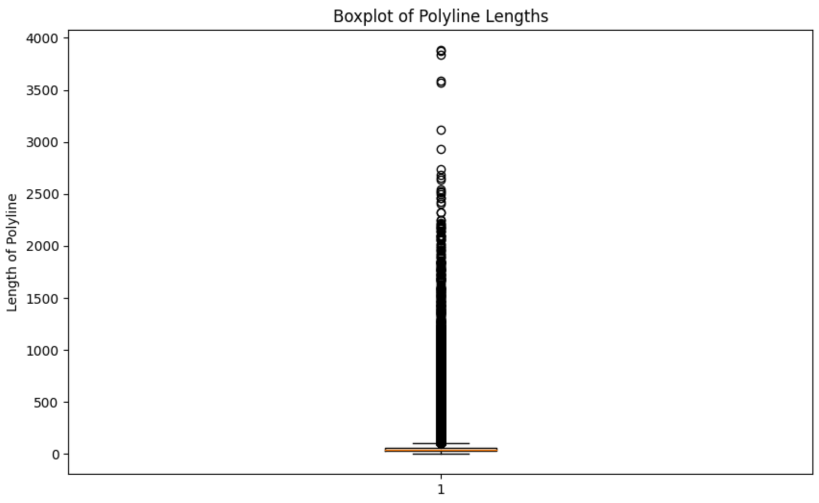


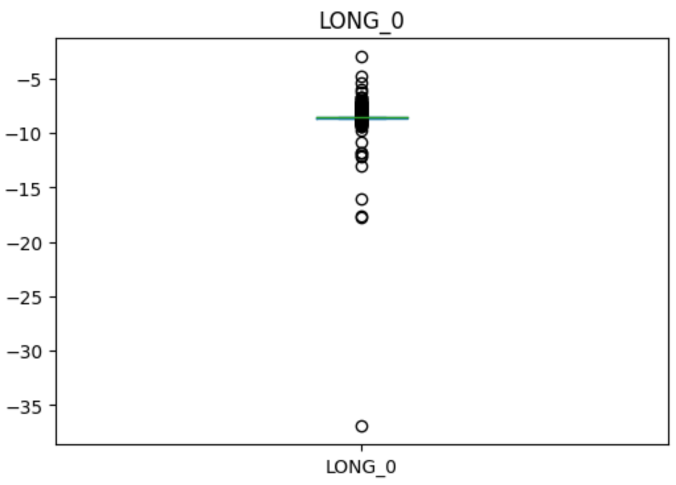
Figure 1: Boxplot of the full dataset's polyline lengths

Noticing that these were mostly short, we decided to determine by trial and error a small number of selected points, and this was set at 3 (which is potentially not the optimal solution, but which is, for lack of time, ours). So, firstly, for polylines of size less than or equal to 2, the first point is repeated 3 times. Then, for polylines of size equal to 3, we retain the first point and repeat the second 2 times. After that, for polylines of size equal to 4, we use the first, second and third points, and finally, for polylines of size greater than 4, we use the first point, the point in the middle of the polyline and a point drawn randomly from the end of the polyline. The destination being chosen for all situations as the last point of the polyline.

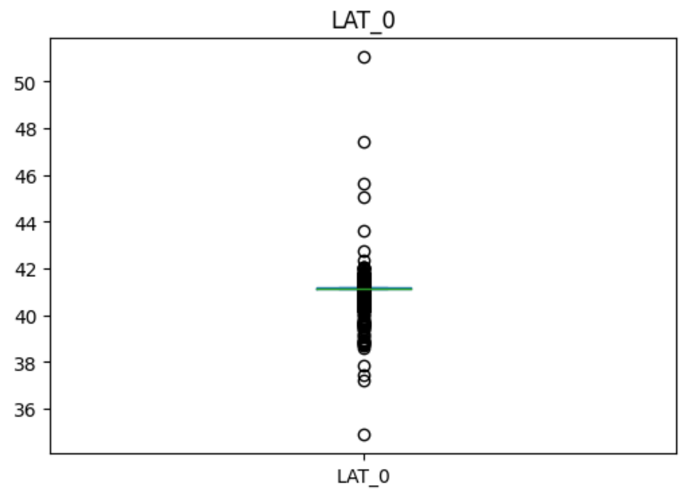
Once we had decided to keep 3 points per polyline, we quickly realized that this could be detrimental to predictions on longer polylines, as the destination of these could be very far from the selected points. This is why we decided to add an outlier management step, which consists in assigning a randomly chosen point in the window located after the last point selected and the last point of the polyline. It's on the basis of the previously generated boxplot, and in particular its maximum non-outlier threshold, that we decide whether such a process is useful or not.

3 Longitude & latitude analysis

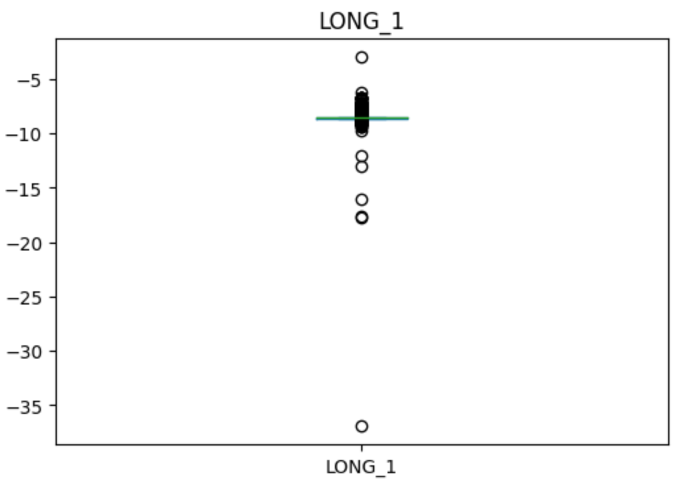
In order to discover possible outliers, we would have liked to carry out a longitude and latitude analysis of the entire data set, but given the large number of data points, this seemed too time-consuming. We therefore decided to carry out this analysis after reducing the number of points used in the previous step.



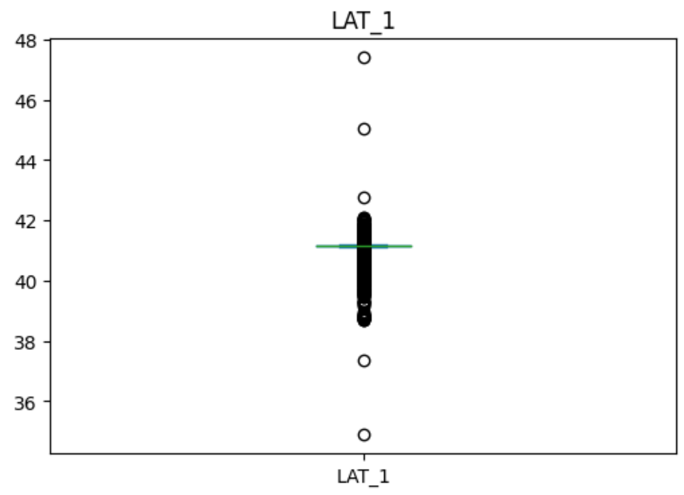
(a) Longitude of 1st point



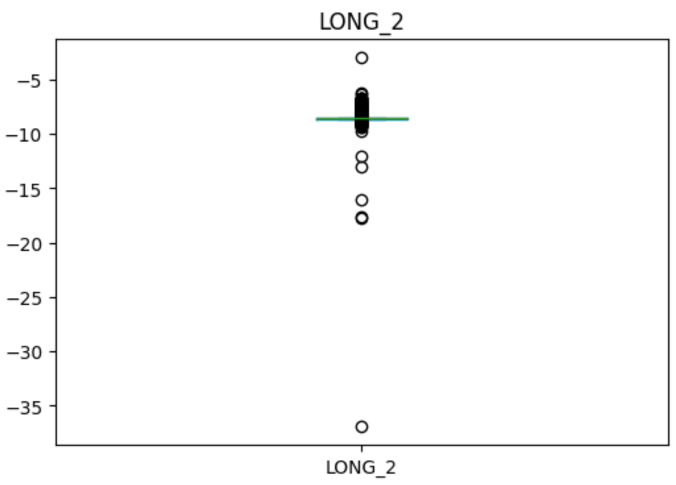
(b) Latitude of 1st point



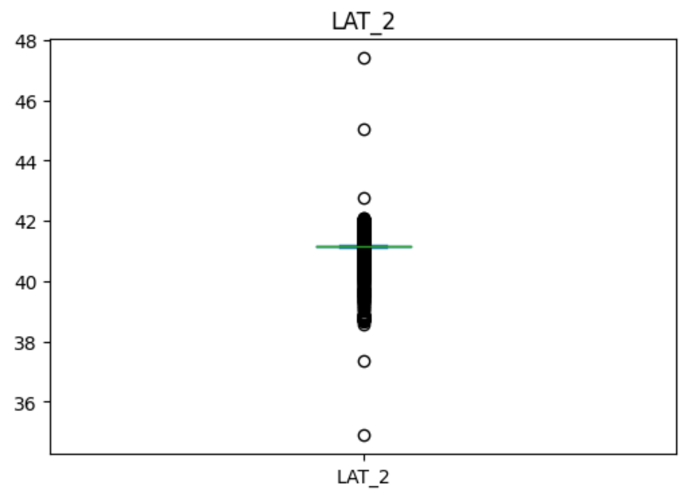
(c) Longitude of 2nd point



(d) Latitude of 2nd point

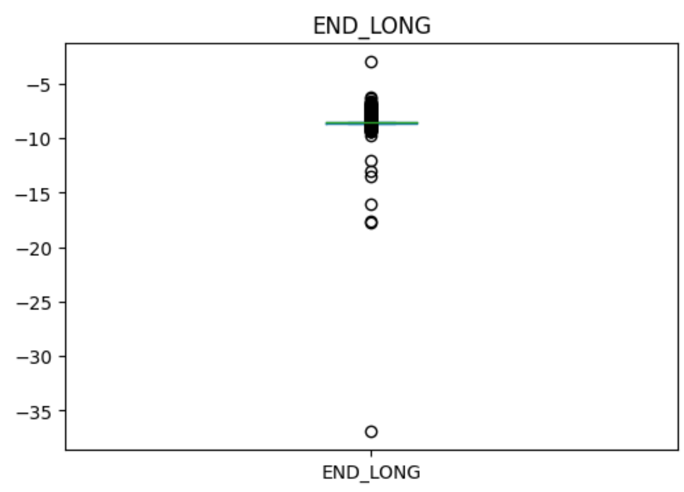


(e) Longitude of 3rd point

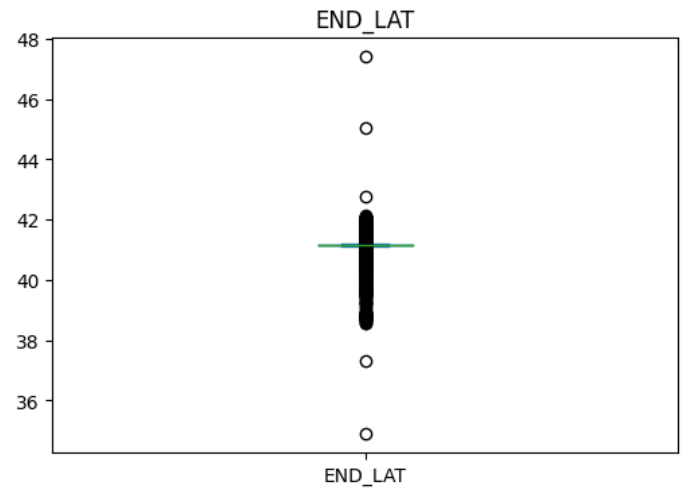


(f) Latitude of 3rd point

Figure 2: Longitude and latitude of selected points - 75% of dataset size



(g) Longitude of destination point



(h) Latitude of destination point

Figure 2: Longitude and latitude of selected points - 75% of dataset size

Once this analysis had been carried out, we hit a dead end: how to deal with these outliers ? One solution might have been to determine a zone around the city of Porto beyond which we would not take the points into account. We would then replace these points with the maximum non-outlier threshold. However, for lack of time and confidence, we preferred to avoid implementing such a process.

4 Dataset splitting

Once all this analysis and preprocessing was done, all that remained was to split the dataset into an X and a y on which to train the models. To do this, we simply placed the 3 previously selected points in X and the destination in y.

5 Model hyperparameters tuning

Hyperparameters tuning is the technique of choosing the optimal values for the hyperparameters of a machine learning model. They are used to control the behavior of the learning algorithm and can impact on the performance of the model. There are several methods for adjusting hyperparameters. Here are the ones we used:

5.1 Grid search

Grid search is an automated method. We simply specify a grid of hyperparameter values to search for, and then each model concerned is trained and evaluated for each combination. The one with the best performance is selected. We used that that approach using 5-fold cross-validation on our dataset for *k-nearest neighbors*, *decision tree* & *ridge* models.

Grid search, however, has the potential to be computationally costly due to the need to train and assess the model for each combination in the grid. When dealing with huge grids or sophisticated models, this may become impractical. This explains the use of manual tuning as well.

Note that for *kNN*, we were able to use grid search up to 10% of the dataset size, after which we set its parameters to the last values obtained with grid search.

5.2 Manual tuning

Manual tuning involves manually experimenting with various hyperparameter combinations and assessing the model's performance for each one. This technique was used for *random forest* model.

5.3 Hyperparameter Evolution by Dataset Size

Algorithm	**Dataset Size**	**Hyperparameter 1**	**Hyperparameter 2**	**Hyperparameter 3**
K-Nearest Neighbors	0.1%	'algorithm': 'auto'	'n_neighbors': 512	'weights': 'uniform'
	1%	'algorithm': 'auto'	'n_neighbors': 128	'weights': 'uniform'
	10%	'algorithm': 'auto'	'n_neighbors': 256	'weights': 'distance'
	30%	/	/	/
	50%	/	/	/
	75%	/	/	/
Decision Tree	0.1%	'max_depth': 10	'min_samples_leaf': 1	'min_samples_split': 10
	1%	'max_depth': 10	'min_samples_leaf': 1	'min_samples_split': 2
	10%	'max_depth': 10	'min_samples_leaf': 2	'min_samples_split': 2
	30%	'max_depth': 100	'min_samples_leaf': 4	'min_samples_split': 10
	50%	'max_depth': 100	'min_samples_leaf': 4	'min_samples_split': 10
	75%	'max_depth': 1000	'min_samples_leaf': 4	'min_samples_split': 2
Ridge	0.1%	'alpha': 0.1	/	/
	1%	'alpha': 0.1	/	/
	10%	'alpha': 0.1	/	/
	30%	'alpha': 0.1	/	/
	50%	'alpha': 0.1	/	/
	75%	'alpha': 0.1	/	/

Table 1: Evolution of Best Hyperparameters per Algorithm as a Function of Dataset Size

6 Model analysis

	KNN	**DT**	**RIDGE**	**LR**	**RF**
SCORE	3.666	3.482	3.373	3.332	3.194

(a) 0.1% of dataset'size

	KNN	**DT**	**RIDGE**	**LR**	**RF**
SCORE	3.669	3.177	3.202	3.195	3.021

(b) 1% of dataset'size

	KNN	**DT**	**RIDGE**	**LR**	**RF**
SCORE	3.629	3.159	3.267	3.267	3.108

(c) 10% of dataset'size

	KNN	**DT**	**RIDGE**	**LR**	**RF**
SCORE	3.667	3.124	3.129	3.128	3.006

(d) 30% of dataset'size

	KNN	**DT**	**RIDGE**	**LR**	**RF**
SCORE	3.661	3.072	3.227	3.227	3.044

(e) 50% of dataset'size

	KNN	**DT**	**RIDGE**	**LR**	**RF**
SCORE	3.638	3.248	3.281	3.281	3.084

(f) 75% of dataset'size

Figure 3: Comparison of measurement for different of dataset'size

As we can in the different Tables 3, increasing the size of the training dataset can have different effects:

- **BENEFITS**

  1. **Improved Generalization:**
     KNN, Decision Tree, and Random Forest: These algorithms can benefit from a larger dataset as they tend to generalize better with more diverse examples.

2. **Reduced Overfitting:**
   Ridge Regression: Increasing the dataset size can help in reducing overfitting, especially when dealing with a large number of features.
3. **Enhanced Model Stability:**
   Linear Regression: A larger dataset can lead to a more stable and reliable linear regression model, reducing the impact of outliers.

- **DRAWBACKS**
  1. **Computational Complexity:**
     kNN: The computation of distances in kNN can be computationally expensive with larger datasets, making predictions slower. For example, apply GridSearch on kNN with 30% of the dataset is taken way too long on Kaggle so we had to tune the parameter of kNN to find the best and then put in the main programm.
  2. **Overfitting Risk:**
     Linear Regression: While linear regression models can benefit from more data, there's a risk of overfitting if the model becomes too complex relative to the dataset.
  3. **Tree Depth and Complexity:**
     Decision Tree and Random Forest: Increasing the dataset might lead to deeper trees, which can result in overfitting. Proper tuning of hyperparameters is essential.
  4. **Diminishing Returns:**
     All Algorithms: There might be a point of diminishing returns, where the additional data provides minimal improvement in model performance.
- **NOTE**
  As we can see on the different tables, the score become stable after a certain size of dataset for all algorithms so it means that better performance cannot be achieve this way and the improvement has to come from somewhere else like a better preprocessing or scaling the data.

6.1 k-Nearest Neighbors

K-nearest neighbors (KNN) is a non-parametric approach which bases its predictions on the mean of the target values of the closest K neighbors in the training data. This method is influenced by the scale of the features, often necessitating preprocessing steps for optimal results.

6.2 Decision Tree

A particular kind of model called a decision tree bases its predictions on a set of guidelines that it has acquired through training data. It builds a structure like a tree, with each leaf node representing a predicted value for the target variable and each inside node representing a choice based on one of the input features.

6.3 Ridge

Ridge regression is a type of linear regression model which incorporates a regularization element into its objective function. This component is designed to constrain the size of the coefficients, aiding in avoiding overfitting. It is particularly useful in situations where there is a high number of predictors and a significant risk of overfitting.

6.4 Linear Regression

A popular and straightforward technique for simulating the relationship between a dependent variable and one or more independent variables is linear regression. It uses a least squares method to estimate the coefficients of the model and makes the assumption that the relationship between the variables is linear.

6.5 Random Forest

The ensemble technique known as "random forest" aggregates the forecasts from several decision trees. Using bootstrapped samples of the training data, it builds a set of decision trees, each of which predicts the target variable. The average of the predictions each individual tree provided makes up the final forecast.

6.6 Models that we could not use

So in our research to find better algorithm to determine the final destination, we have found AdaBoostRegressor and StackingRegressor that are ensemble learning techniques that aim to improve predictive performance by combining the predictions of multiple base models.
For AdaBoost, its strengths are reducing bias and increasing accuracy and being robust to overfitting. but it is sensitive to noisy data and outliers since we think that we have handled the outliers correctly, it could have some good performance.

For Stacking, its strengths is that it can capture complex relationships in the data and utilize the strengths of different models but it requires careful tuning of hyperparameters and is computationally more expensive(which is not really a problem with the power of GPU of the Kaggle notebook).

However, due to the disposition of the output (y) in a 2D array and the need of those algorithms to have y in a 1d array they could not be used.

7 Conclusion & Improvements

We chose random forest as the best regression method as by looking at the results, one can see that random forest performs better and gives the best scores. Something we could have done better was to fine tune hyperparameter even more maybe on better preprocessed data.