NEXTWAVE

EY Datascience Challange 2019

Authors: Lawrence Adu-Gyamfi and Adrià Fenoy

Challange Description

The **EY NextWave Data Science Challenge 2019** focuses on how data can help the next **smart city** thrive, and boost the mobility of the future. Global urbanization is on the rise, with more than 50% of the world's population living in cities; according to the UN, that number will reach 60% by 2030 – that's nearly 1.5 billion more than in 2010.

While this trend creates great opportunities for cities, it also presents challenges to governments on how to upgrade infrastructure, alleviate congestion and address pollution. Electric and autonomous vehicles, along with the explosion of the ride sharing economy, are helping to address these challenges which also disrupt mobility and demand innovative solutions.

In parallel, public authorities have more information than ever on how citizens move around in the city. However, a gap exists between having this data and using it to improve the user travel experience for citizens. Forward-looking authorities have a chance to innovate infrastructure to make their city a better place to live in a better working world.

Challange: Predict whether the trajectories, recorded from GPS signals, end in the city center after 15:00.

Dataset

The data consists in **~1.000.000 trajectories** recorded by **~170.000 unique devices** during one day. Each trajectory has 11 fields:

Variable name	**Description**
hash	Represents the unique identifier of a device
trajectory_id	Represents the unique identifier of a trajectory associated to a device
time_entry	Indicates the local time for the ending point of the trajectory (HH:mm:ss)
time_exit	Indicates the local time for the ending point of the trajectory (HH:mm:ss)
Vmax	Represents the maximum velocity registered in the course of a trajectory
Vmin	Represents the minimum velocity registered in the course of a trajectory
Vmean	Represents the average velocity registered in the course of a trajectory
x_entry	Entry x coordinate (cartesian projected position)
y_entry	Entry y coordinate (cartesian projected position)
x_exit	Exit x coordinate (cartesian projected position)
y_exit	Exit y coordinate (cartesian projected position)

Methodology

With the goal to obtain the best predictions for the test set, we followed the following steps. First of all, we transformed the data completely in order to avoid the problem of diferent number of trajectories for each hash. We got **91 statistics** representing different features of the data. Additionally, we took the last trajectory for the test set and evaluated whether it was in the center or not. We used this information as labels to set up a **classification problem**.

After the data preprocessing and data cleaning part, we trained two Macine Learning models: **XGBoost** and **Feedforward Neural Network**. We evaluated which performed the best with the train set (XGBoost gave us the best predictions). Finally, we used the best model to predict whether the trajectories in the test set were ending in the city center or not, which is the goal of the challange.

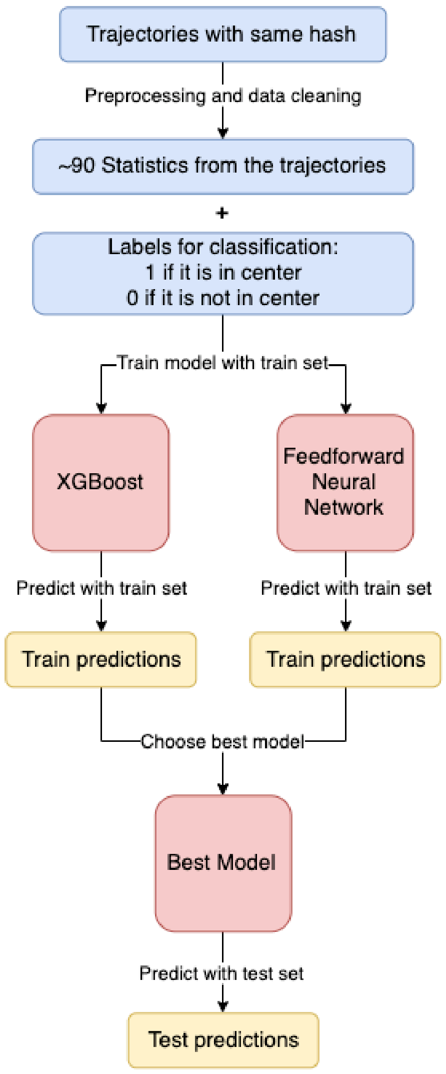

Figure 1: Methodology we followed to get our predictions

Data Preprocessing

The original format of the data is not adequate to fit it directly into a model. To solve this, first of all, we **grouped the trajectories by hash**. Then, we converted each trajectory to two **triplets (x, y, t)**, one corresponding to the entry point and the other to the exit point. All the triplets are fed in the same list, independently if they are entry or exit points, we only care that the device was there at a given time.

The last trajectory for each hash (the one with the latest time) is treated differently. The entry point is added to the same list as other points, while the exit point, which is the one we are interested in predicting for the test set, is divided in two parts. The time of that point corresponds to the **prediction time**, and is another attribute which later on will be fed into the models. The (x, y) coordinates are evaluated whether they are in the center or not (encoded as 1 or 0), and this is saved as the **label** which we use later to train our models.

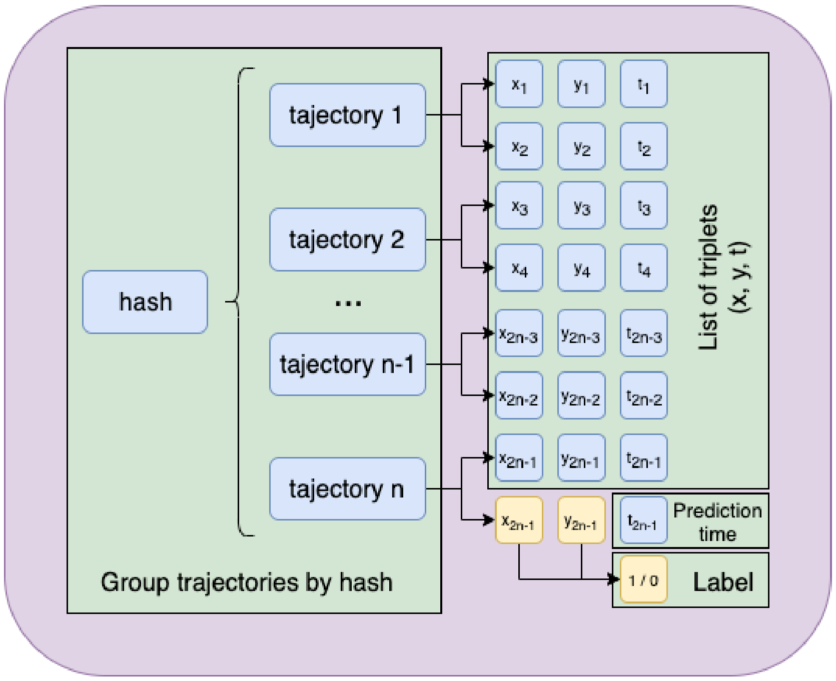
hash
tajectory 1
tajectory 2
...
tajectory n-1
tajectory n
Group trajectories by hash
x_1 y_1 t_1
x_2 y_2 t_2
x_3 y_3 t_3
x_4 y_4 t_4
x_{2n-3} y_{2n-3} t_{2n-3}
x_{2n-2} y_{2n-2} t_{2n-2}
x_{2n-1} y_{2n-1} t_{2n-1}
List of triplets (x, y, t)
x_{2n-1} y_{2n-1} t_{2n-1}
Prediction time
1 / 0
Label

Figure 2: Preprocessing to have the data in a format easier to work with.

Data Cleaning

Up to now, we had different number of **points (triplets)** corresponding to each device. To solve this, we extracted a total of **91 statistics** from each list of points and the prediction time. All statistics were **averages, maximums or minimums** with respect different inputs or other statistics. Also, we estimated the **average velocity** of the device at each point. In the end, we obtained a list of statistics with the same length independently of the initial size of the list of triplets.

Additionally, we removed from the training set the devices with extreme values in the aforementioned statistics. Mainly, we removed devices with average velocities which were too high. After data cleaning we ended with the **96% of the original training set**, as we did not want to purge data excessively.

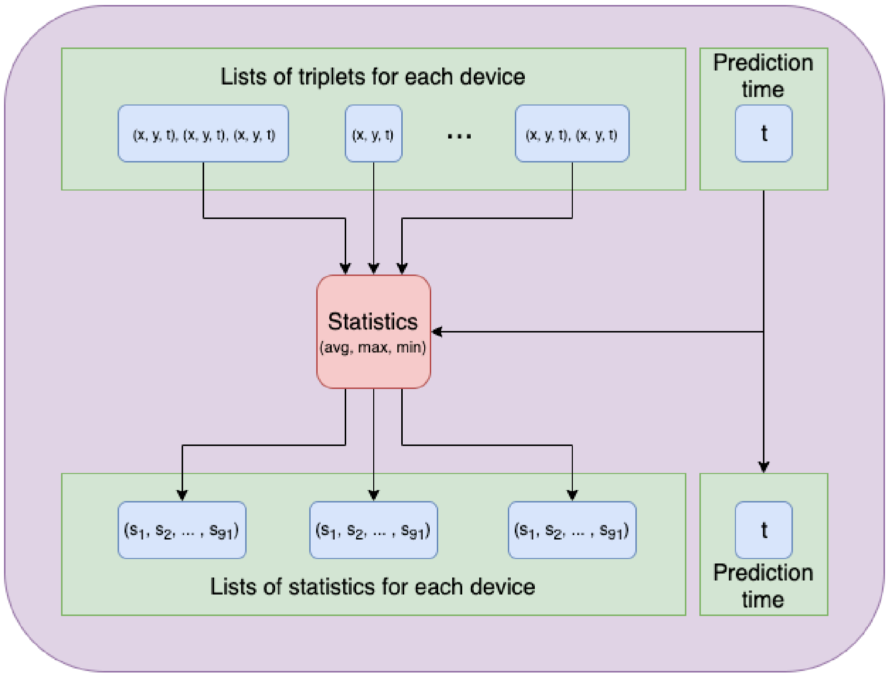

Figure 3: Process to compute statistics which cold be meaningful for the model.

Insights

First thing we noticed is that a lot of devices, ~93%, are **described by last point**: if last point was in the center, they are in the center, and vice-versa. There were ~4% of devices, which were not described by last point, that could be described by **first point**. The rest of the data was described by some other point or some of the statistics we created from them. There was just a few data that could be not described due to some weird behaviour.

We also noticed that according to some of the statistics we had created, we were able to give a **criterion** that tells us if we should describe a device by the last point, first point or any other statistic. An example of this is the statistic **time difference from last point**, which for values close to 0 gives more than 95% accuracy. Here, there is a **trade-off** between how much data we want to describe with this information and the error we want to obtain: if we want to describe more data we will accumulate more error. This decision has to be made for all our 91 statistics. In order to solve this problem we decided to use **machine learning** techniques that learnt the best way to describe data according to these statistics, finding by themselves the optimal trade-off.

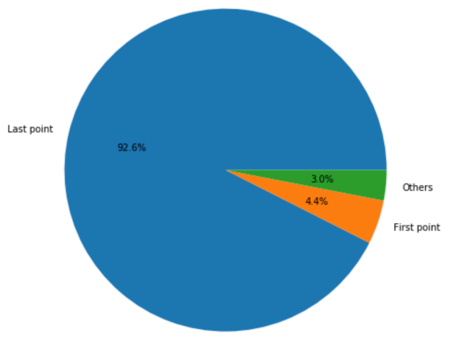

Figure 4: Pie chart showing how much data can be described by different points.

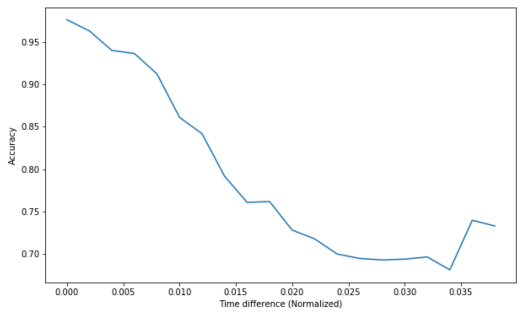

Figure 5: Accuracy of the model for different time difference from last point.

Another approach we had in mind was to obtain more **collective statistics** such average velocity, most frequent destination, and others. To do this we used all the points recorded by the GPS signal to draw a map (main roads and locations can be extracted from this map). Then, we used k-means clustering to obtain **500 cluster** of most crowded locations. Our idea was to compute these statistics for each cluster and add them to each point according to the cluster each of them belong.

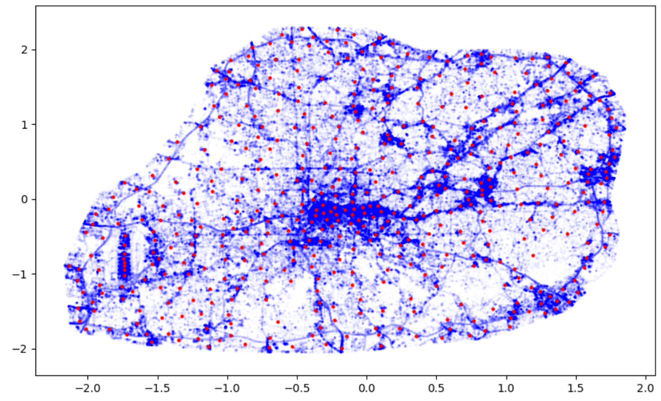

Figure 6: 500 clusters resulting form a k-means clustering.

We also noticed that there is a **time dependence** in the percentage of people in the center during the day, being from **10h to 14h** the most crowded hours. We think that this information could be added to the model as a prior, or we could add some correction to the results based on this information, but we didn't explore this further.

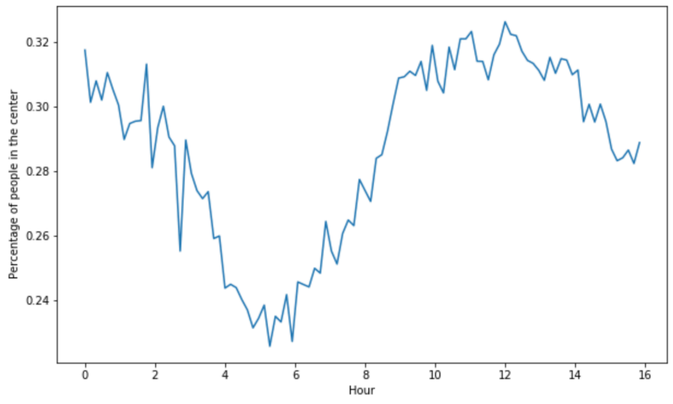

Figure 7: Percentage of people in the center at each time.

Model: XGBoost

XGBoost is a framework that implements **gradient boosting**. Despite it has implementations for many programming languages, we used its implementation in R. Gradient boosting is based on sequentially ensembled **decision trees** with each subsequent set up to correct the errors (residuals) of the previous one. By this procedure, each subsequent tree is able to give more accurate predictions, resulting in a very robust model.

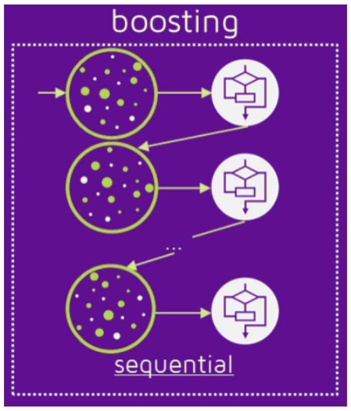

Figure 8: Tree boosting schema.

- **Model Training**

  The training data was split into two portions (80% for training and 20% kept for validation). In order to obtain the best out of the model, we tuned the some of the multiple hyper-parameters of the model using k-fold, $k = 5$, **cross validation**:

  - Maximum depth of each tree
  - Learning rate: The amount of information to carry on to subsequent trees from previous ones.
  - The minimum loss reduction to be considered before making further splits at nodes of trees.
  - The number of variables to consider when building each tree.

- **Model Validation**

  The final model was then used to make predictions on the validation set in order to evaluate the performance. The major characteristics of the model we have looked at the accuracy, precision and recall and ultimately the F1-score.

Model: Feedforward Neural Network

Feedforward neural network is the simplest of the **neural networks** models. They are composed of different artificial neurons structured in layers. Each layer process the inputs from the previous one and provides a representation of the data. This approach achieves to give a high level representation of the data after a few layers. Then we ask the model to **classify** the inputs according to this **high level representations**.

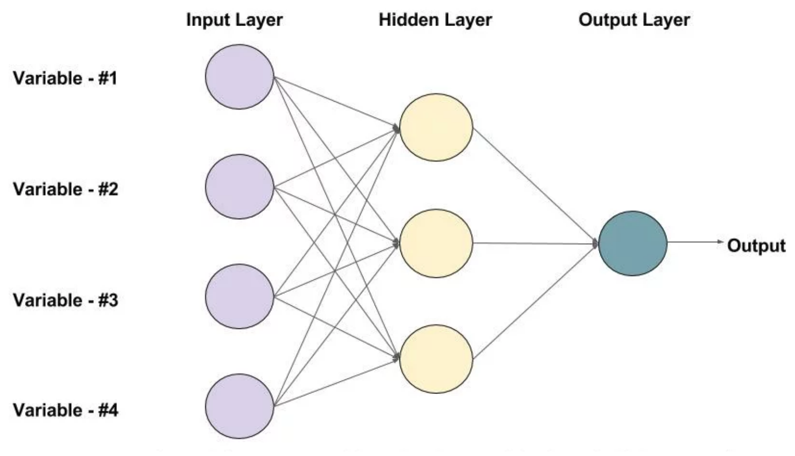

Figure 9: Neural network schema.

- **Model Training**

  The training data was split into two portions (70% for training and 30% kept for validation). The model hyperparameters were estimated by trial and error.

- **Model Validation**

  Neural networks are usually trained over many epochs, but training them too much may cause overfitting on the training set. In order to know were we should stop training we measured the **ROC-AUC** score after each epoch and picked the model with highest score (ROC-AUC takes values from 0 to 1, being 1 the best model).

Results

With our approach we achieved to get the **second position in Spain**. This was such a competitive challenge and all participants obtained very close scores ones with each other. Our best approach was the **XGBoost model**, despite the neural network was really close one. In both cases we achieved to improve a lot the naive solution of predicting that people did not move since last time the were tracked.

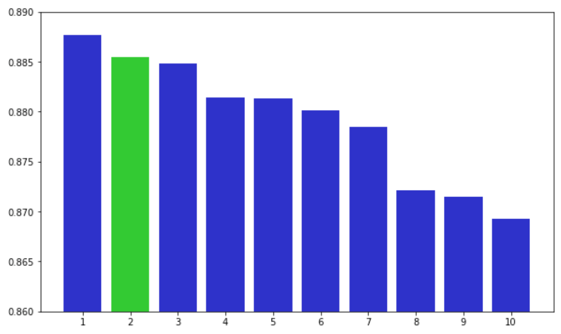

Figure 10: Regional results in Spain for the EY Nextwave challange.

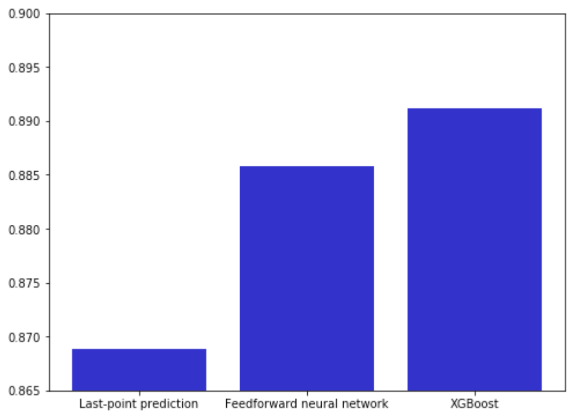

Figure 11: Scores of our models compared to the naive one.

As explained before, we used **ROC-AUC score** to track overfitting for the neural network. We also obtained the **ROC curve** for the XGBoost model to evaluate the true positive rate vs false positive rate, which are very important factors if we want to obtain a good F1 score in our submissions.

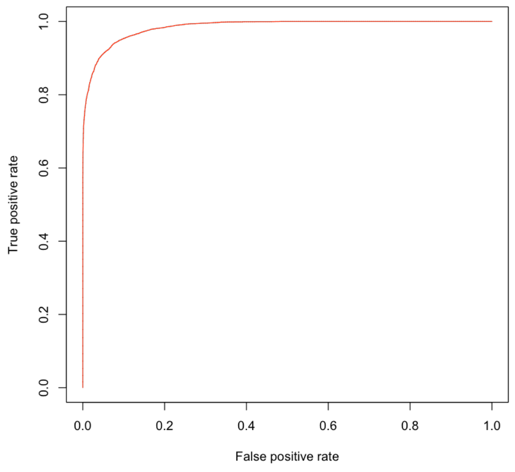

Figure 12: ROC Curve for XGBoost.

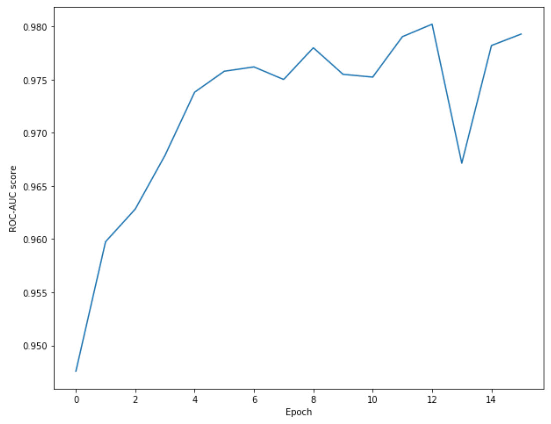

Figure 13: ROC-AUC for each epoch of training for the neural network.

Possible Model Improvements

1. **Collective statistics in data expansion:** To increase our model predictions, we could compute collective statistics (such as mean velocity and most frequent destinations), and input them to the model to boost its performance. This collective statistics could be computed for each cluster in our **k-means clustering** results, and added to the individual statistics according to the cluster which each input belongs.

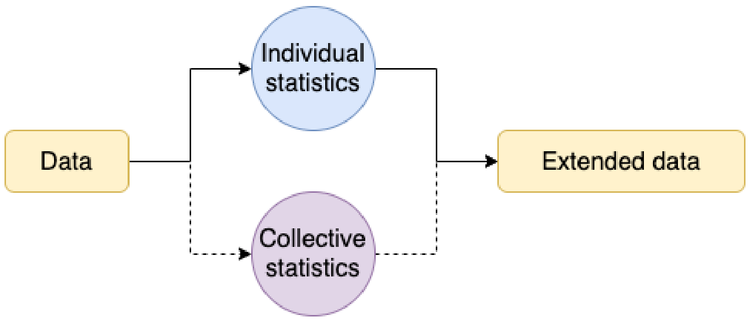

Figure 14: Possible model improvements with collective statistics in data expansion.

2. **Ensembled model:** We tried two models and picked the best one according to best overall performance on the train set. But we can do better than that, we can train a new model to select the best prediction according to other model predictions for each kind of input. This technique is named **model ensembling** and can give better predictions than each model by its own.

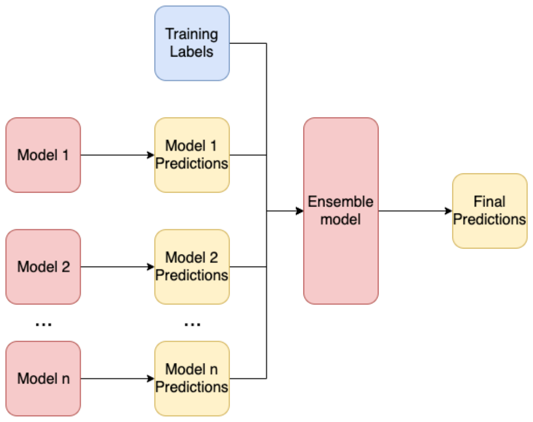

Figure 15: Possible methodology improvement with ensembled model.

Applications and Future Work

In the **era of artificial intelligence and data**, the concept of living in an smart city will become something the society will be concerned about in a few years. **Smart cities** are those which make use of the technology in order to manage resources efficiently.

In that sense, we think that our findings can contribute to **optimize traffic** by creating more efficient routes and thus reducing the emissions of the vehicles. We also saw that there is a lot of people which move following similar behaviours. This brings up opportunities to create new public transport routes, or optimize its schedules.

Another area which benefits about our findings is the **commerce**. For instance, understanding how and when people move, shopkeepers can decide where they should open his new business, which is the best hours to keep the shop open, and how many people they need in the shop during different hours in the day or different epochs of the year.