Universitat Politècnica de Catalunya

Barcelona School of Informatics

Master in Innovation and Research in Informatics

Machine Learning

- New York City Taxi Trip Duration -

Authors
David Morán
Joel Cantero
Ricard Meyerhofer

Lecturers
Lluís Antonio Belanche

June 23, 2019

Index

1 Introduction

1.1 Project Description

The problem we want to predict is a regression problem that consists in the prediction of New York's City taxi fare price. The total price of a trip, is variable that we think it might be interesting to predict since it does not only depend on the origin and destiny. It is not difficult to see that is a factor that can depend on other variables such as time period, traffic density, tolls, etc.

To solve this problem, decided to use the data of the New York City Taxi Fare Prediction data from a Kaggle challenge[1] which contains around 1.5M of rows and 11 columns where the primary dataset is one released by the NYC Taxi and Limousine Commission, which includes pickup time, geo-coordinates, number of passengers, and several other variables as can be seen on Table 1.

Feature Name	**Description**	**Type**
id	Unique identifier	string
vendor_id	Code indicating the provider associated with the trip record	integer
pickup_datetime	Date and time when the trip begins	date
dropoff_datetime	Date and time when the trip ends	date
passenger_count	Passengers in the vehicle (driver entered value)	integer
pickup_longitude	Longitude coordinate where the trip starts	double
pickup_latitude	Latitude coordinate where the trip starts	double
dropoff_longitude	Longitude coordinate where the trip ends	double
dropoff_latitude	Latitude coordinate where the trip ends	double
store_and_fwd	Indicates whether the trip record was held in vehicle memory before sending to the vendor server	character
trip_duration	Duration of the trip in seconds	double

Table 1: Features description

1.2 Previous Work

This is a very hot topic because it is related to traffic and mobility in general. Without going further, if we think in our dailies, we can think of applications such as Uber or Cabify are able calculate beforehand an approximate price in order to give a price estimation. This could not be done without them being able to know how much time it will take them to go from one point to another. Google Maps can also calculate how much time you will take from two points.

If we look in the same area of datasets and competitions, we can see that this dataset and challenge is very familiar with other challenges that have been popular during these last years. We can find the ECML/PKDD 15: Taxi Trip Time Prediction (II)[6] that was asking to predict how long a taxi will be occupied or also we can find. In a similar fashion, we can find other challenges that have been running predicting other variables such as final location instead of the time as ECML/PKDD 15: Taxi Trajectory Prediction (I)[7] or in case of New York City Taxi Fare Prediction the price of a trip and Chicago Taxi Trips[8] .

This problem has been covered in the academia in many related areas which are: travel-time prediction[3], dynamic travel time[17], short-term prediction[4], mobility patterns[12], flow analysis[13], etc. Inside these topics we can find many sources and most of them are worth a read.

Note that this topics are not exclusively applied to the dataset we are using but that for instance are applied to other cities, to other vehicles such as Buses or applied to certain technique such as SVM's[14].

2 Data exploration

2.1 Pre-processing

In our pre-processing we decided to remove outliers. We found outliers in the coordinates where we decided to remove those points that were the farthest by calculating the mean and taking for pickup and drop-off the 500 values which resulted in removing around 0.05% of data. In the images above we can see that pick-up and drop-off points follow a normality but we can see some peaks of density in the tails.

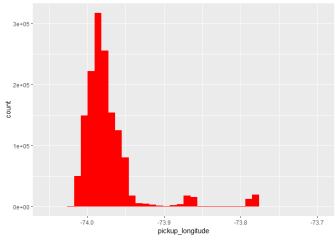

Figure 1: Pick-up longitude

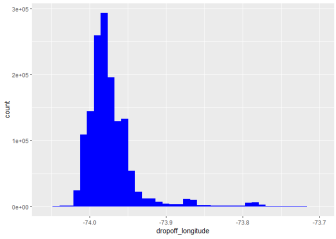

Figure 2: Drop-off longitude

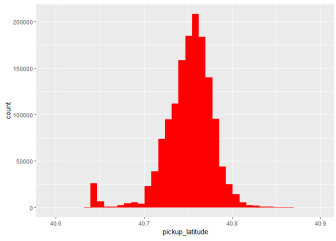

Figure 3: Pick-up latitude

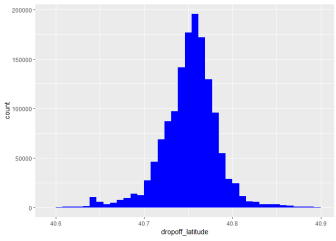

Figure 4: Drop-off latitude

In the same fashion, we also decided to take those trip durations that last less than an hour since only a 0.8% last more than 1h and 0.1% take more than a 2h trip.

We also decided to measure the *duration* to minutes instead of seconds for readability and we scaled and center the latitude and longitude for the modelling part

2.2 Feature selection/extraction

Our objective in this project is to use a reduced number of features to predict the duration time, we want a reduced number of features to avoid over-fitting and also not to have very complex models that will take a huge time executing (considering the data we have). Therefore, from the variables we have, we are going to focus only in the geographical therefore, we are going to remove the id, $vendor\_id$, $pickup\_datetime$, $passenger\_count$, $store\_and\_fwd\_flag$ and $dropoff\_datetime$.

We decided to drop this features because after performing visualizations and studying them, we found them not so relevant to the problem plus the size of the dataset let to heavy computational times that with less features we could reduce. Other variables such as $date\_time$ in both cases (pick-up and drop-off) are variables that we do not really need as we focus in the duration.

2.3 Clustering

We are going to compare different clustering techniques and compare them together with an analysis of which conclusions we can extract from it. In first place, we are going to implement, is K-means. Theoretically, K-means is not the most appropriate algorithm geographic data since our data is not linear so we are going to try OPTICS and DBSCAN that perform better with geographical data. The results of K-means visualizations can be found in the Appendix.

If we compare the difference between K-Means vs OPTICS or DBSCAN, we can see that the difference is not as big as we could expect since the data is not this far geographically talking. In our particular case, once we saw this, we decided to stick with k-means because it allowed us to show data on a map easier because of the way that the other two methods return the results and what is more important in these cases is that is visually pleasant.

Overall we can see that as more clusters we generate the more far the points from Manhattan are which makes sense since there are a lot of short trips in our dataset. Also we can see that pickup (green) clusters are more abundant than drop-off (red) clusters.

2.4 Visualization

In this section we are going to show some of the relevant things that we saw on the dataset.

As we can see in figure 17 and 18, drop-off points are more sparse than the pick up points which makes sense since the data comes from NYC taxis, taxis that pick up will pick up in New York and can drop off anywhere but not the other way. This same idea can be seen in the heat map of figures 19 and 20 and by plotting the lines of 100 random points of our dataset, we see that most of this trips are relatively short which we saw in our data and that we can note if we plot the duration and distance densities where we can see that they follow a normal trend.

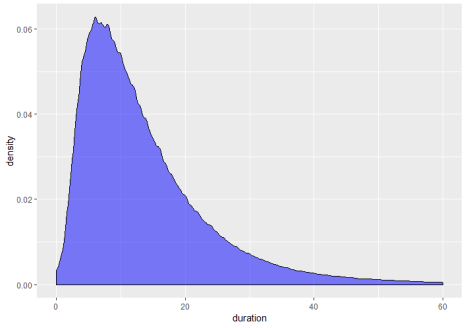

Figure 5: Duration density. Univariate analysis

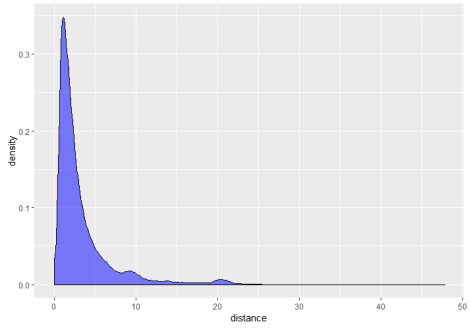

Figure 6: Time density. Univariate analysis

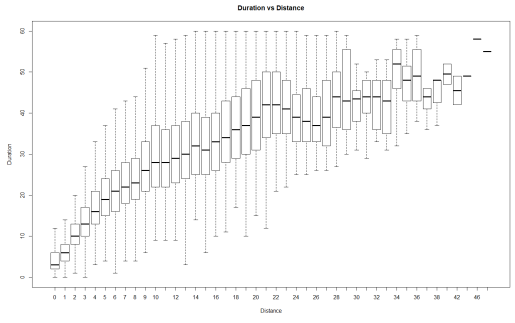

Figure 7: Duration vs Distance. Bivariate analysis

We have not included most of the plots from the data exploration process because we did not find them relevant enough to put them.

3 Modelling

Once the dataset has been pre-processed and analyzed in the visualization and the clustering steps, we proceed to specify the models that will be trained on this dataset. At that point, 1445986 data samples are available for the modelling phase.

Since this is a huge amount of instances, and our computation resources are limited, we decided to establish a train/validation/test split as the validation protocol with a 40/30/30 distribution. This leads to a training set of 578394 data samples, and a validation set of 433796 samples each. This decision was taken for several reasons:

- Due to the high amount of data available, the training times of the algorithms will be very large. This means that, with the same computing time, we will be able to train and validate less models than if we trained with a slower sample size.
- Even with only 40% of data for training, we still have lots of training samples. We are confident that this will be enough to obtain good results, since is difficult to make a model overfit that huge amount of data.
- The final dataset is very low-dimensional, since it only has 5 features. This also makes difficult for the models to overfit, and allows us to obtain good results with a lower amount of data.

The validation protocol will consist on the following: for each proposed technique, the corresponding hyperparameters will be tuned. This is done by training a different model for each combination of hyperparameters using the **train set**. Then, the behaviour of those models will be validated using the **validation set**, with this behaviour being measured as the Mean Absolute Error (MAE) of the predictions. Finally, the model which performed best on the validation set will be selected, and its generalization error will be estimated with the **test set**. This generalization error will also be reported

as the MAE.

Since the problem we have in hands is a regression one, the following techniques will be considered:

- Linear Regression
- Random Forest
- RBF Neural Networks
- eXtreme Gradient Boosting (a.k.a. XGBoost)

In all cases, data will previously be centered and scaled taking as a basis the distribution of the train set. In any case, the target will not be scaled at all.

3.1 Linear Regression

The Linear Regression model is the first approach, since it is a very fast and simple technique that may show us a baseline for the metrics of the rest of the models. It is expected to be the worst of them, but its results might be good enough given the distribution of the data and the low-dimensional input space.

First, classical Linear Regression will be trained on the data, and its performance will be evaluated. If the model overfits the data (which is highly unlikely to happen) Ridge regression will be considered, and its λ hyperparameter will be tuned as stated before.

3.2 Random Forest

Random Forest is a classical, well studied model which usually achieves great performance out of the box. Usually, the number of trees and the number of random picks of samples and variables is tuned, but in this project we decided to keep all of them fixed. This decision was taken for several reasons:

- The training time of the algorithm is directly proportional to the number of trees. Therefore, we cannot afford to have a big number of them, and we fixed it to be 1000. Since it is difficult for Random Forest to

overfit the train set, we thought there is no need to further tune this parameter.

- The number of random data samples picked is another parameter that directly impacts on the training time of the algorithm. We decided to set it to be 10000 without repetition, so the probability of not picking some of the data samples within all the trees stays very low, but at the same time the training time keeps within reasonable.
- The number of variables picked has been set to be all of them, since we only have 5 variables. Trying to work with less of them would probably lead to a very uninformed model which will not perform as good as expected.

The performance of this model in the validation set will be computed anyway, in order to be able to perform a valid comparison with the other methods.

3.3 RBF Neural Networks

Another model that classically performed well in regression is the RBF Neural Network. This technique is highly recommended for low-dimensional problems with high amounts of data, which is precisely our case.

The training strategy for the RBF NN is the following:

1. Perform a K-Means clustering over the training set with M centroids.
2. Compute the design matrix of the hidden layer of the model. Each element of the matrix is computed as $\Phi_{i,j} = exp(-\gamma||x_i - m_j||^2)$, where γ is a hyperparameter that needs to be tuned.
3. Compute the linear weights of the model by performing a linear regression over the design matrix.

In this project, the number of basis functions has been set to 500. A classical heuristic for choosing this number of function is $M = \sqrt{n}$, where n is the amount of data samples. In our case, it turned out to be $M = 760$, but taking that much basis functions lead to a too much big design matrix, which we were not able to perform regression in with our personal computers. Therefore, we decided to reduce that amount to only 500.

As for the γ hyperparameter, the search space was limited to $\{0.1, 0.2, ..., 1\}$, because if the search space was much bigger the amount of time required to process it would be untreatable.

3.4 XGBoost

XGBoost (short for eXtreme Gradient Boosting) is an open-source software library which provides a gradient boosting framework for several programming languages, in particular, for R. It has been proven successful in many challenging tasks and Machine Learning competitions in sites like Kaggle. It is able to perform incredibly fast with huge amounts of data with very good results, although it needs some tuning to take its best.

In this project, the number of training round has been set to 10, which experimentally turned out to be a good non-overfitting number of iterations to train. Then, the maximum depth of the trees and the η parameter are tuned in the following way:

- The maximum depth of the trees has been tuned for values $\{2, 4, 6, ..., 20\}$.
- The η parameter has been tuned for values $\{0.1, 0.2, ..., 1\}$.

4 Experimental Results

The results of the experiments proposed in the previous section can be seen in Table 8. It can be clearly seen that the Linear Regression is clearly underfitting the problem, since its error is way higher than the rest of methods. The other three achieve similar performances, specially the Random Forest and the RBF Neural Network, which have a mean absolute error of around 3 minutes 50 seconds.

On the other side, we have XGBoost. It is clearly the best option of the ones shown, because even though there seems to be a bit of overfit on that configuration (arround 20 seconds between train and validation-test), it achieves the lowest test error with 3 minutes and 41 seconds. It is also important to remark that XGBoost was, together with the Linear Regression, the fastest in both training and prediction.

Method	Linear Regression	Random Forest	RBF Neural Network	XGBoost
Best Parameters			$\gamma = 0.4$	Depth = 14 $\eta = 0.2$
Train MAE	4 min 33 sec	3 min 46 sec	3 min 49 sec	3 min 20 sec
Validation MAE	4 min 33 sec	3 min 49 sec	3 min 48 sec	3 min 41 sec
Test MAE	4 min 32 sec	3 min 48 sec	3 min 48 sec	3 min 41 sec

Figure 8: Experimental Results

In both RBF Neural Networks and XGBoost, some of the available hyperparameters have been tuned. The process has been done sequentially for the RBF Neural Network, due to the high memory requirement, and in parallel for the XGBoost model.

In the case of the RBF Neural Network, only one parameter has been tuned. The different results of the tuning can be seen in Figure **??**, based on the validation error. It is clear that there exists a local minimum in the explored region, since the increase in MAE seems linear when γ is greater than 0.5 and exponential when it is lower than 0.3. Hence, 0.4 is a near-optimal choice for our hyperparameter.

In the case of the XGBoost, two parameters have been tuned. The different results of the tuning can be seen in Figure **??**, based on the validation error. It can be seen that, for certain values of max depth, increasing the η parameter reduces the error, but for depths greater than 10 the effect goes on the opposite way: For those depths, the best value is always 0.2. It is also interesting to notice that using $\eta = 0.1$ works the worse regardless to the max depth chosen. In any case, it can be clearly seen that the optimal hyperparameter combination is found at Max Depth = 14 and $\eta = 0.2$.

5 Proposal

The proposed method is **XGBoost**, with a maximum depth of 14 and a value of $\eta = 0.2$. This lead to a test MAE of 3 minutes and 41 seconds, which is our best estimate of its generalization error. Since the size of the test set

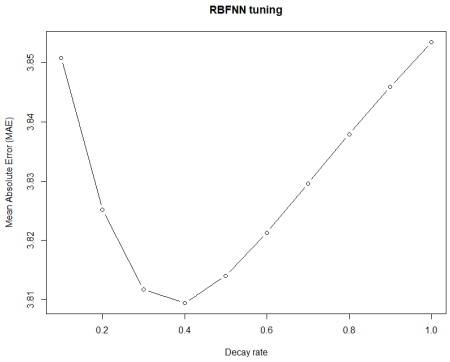

Figure 9: RBF Neural Network hyperparameter tuning

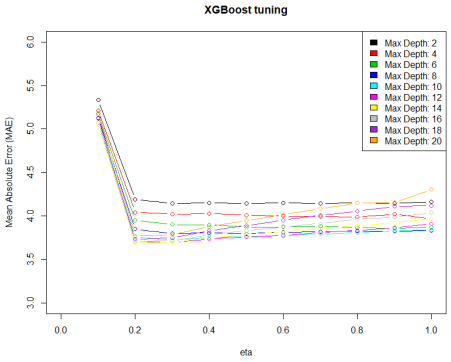

Figure 10: XGBoost hyperparameter tuning

is almost half million data points, we are very confident about the validity of this result.

6 Conclusions and Future Work

We have seen that among the different models we have implemented, the one that performs the best is XGBoost with an error of 3 minutes and 41 seconds. We personally think that it is not a bad result if we consider that our approach had a very reduced number of features but we think that there is room for improvement specially in the analysis part. Some of the improvements that we think would improve our accuracy are:

- **More extensive outlier study of the variables and a less strict feature selection**. This would have definitely improved the accuracy of our models but also would imply to deal with more complex models that would be slower.
- We could have **linked our data with climatic data** so that we could see for instance if there is a relationship between rain and a higher time in trips. We could have also linked our data with other sources such as google maps, events, etc.

On the same way that in the same way there is room for improvement in the

analysis part, we think that there is not much to improve on the modelling part, aside from trying other models. We have tuned our algorithms, we have validated that we do not have any kind of over or under-fitting and from a methodological point of view, we think we did a very good job.

Overall we are satisfied with the results obtained and that we could work on a big dataset. It has been very interesting to work with geographical data so that we had the possibility of projecting our results on maps and create really visual outputs. Is it true that there is room for improvement in the analysis but we chose to work with a reduced number of features for the reasons aforementioned.

7 References

References

[1] New York City Taxi Trip Duration

[2] ECML/PKDD 15: Taxi Trip Time Prediction (II)

[3] Chun-Hsin Wu, Jan-Ming Ho and D. T. Lee, "Travel-time prediction with support vector regression," in IEEE Transactions on Intelligent Transportation Systems, vol. 5, no. 4, pp. 276-281, Dec. 2004. doi: 10.1109/TITS.2004.837813

[4] Xiaoyan Zhang, John A. Rice, Short-term travel time prediction, Transportation Research Part C: Emerging Technologies,Volume 11, Issues 3–4, 2003, Pages 187-210, ISSN 0968-090X

[5] Jacqueline Arriagada, Antonio Gschwender, Marcela A. Munizaga, Martin Trépanier. (2019) Modeling bus bunching using massive location and fare collection data. Journal of Intelligent Transportation Systems 23:4, pages 332-344.

[6] ECML/PKDD 15: Taxi Trip Time Prediction (II)

[7] ECML/PKDD 15: Taxi Trajectory Prediction (I)

[8] Chicago Taxi Trips

[9] Michael Hahsler, Matthew Piekenbrock, dbscan: Fast Density-based Clustering with R

[10] Hanbong Lee, Waqar Malik, Bo Zhang, Balaji Nagarajan and Yoon C. Jung, Taxi Time Prediction at Charlotte Airport Using Fast-Time Simulation and Machine Learning Techniques

[11] Simon Handley, Pat Langley, Folke A. Rauscher, Learning to Predict the Duration of an Automobile Trip

[12] Mohammad Asadul Hoque, Xiaoyan Hong, Brandon Dixon, Analysis of mobility patterns for urban taxi cabs

[13] Marco Veloso, Santi Phithakkitnukoon, Carlos Lisboa Bento, Patrick Olivier, Exploratory Study of Urban Flow using Taxi Traces

[14] Yu Bin, Yang Zhongzhen, Yao Baozhen, Bus Arrival Time Prediction Using Support Vector Machines

[15] Lelitha Vanajakshi, Laurence R. Rilett, Support Vector Machine Technique for the Short Term Prediction of Travel Time

[16] Steven I-Jy Chien and Chandra Mouly Kuchipudi, Dynamic Travel Time Prediction with Real-Time and Historic Data

[17] Steven I-Jy Chien, M.ASCE; Yuqing Ding; and Chienhung Wei, Dynamic Bus Arrival Time Prediction with Artificial Neural Networks

8 Appendix: Clustering

8.1 K-Means

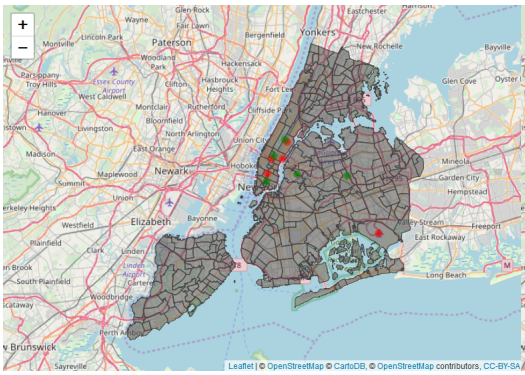

Figure 11: k-means with k=5

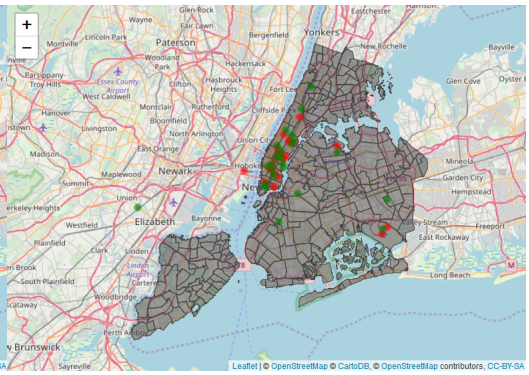

Figure 12: k-means with k=25

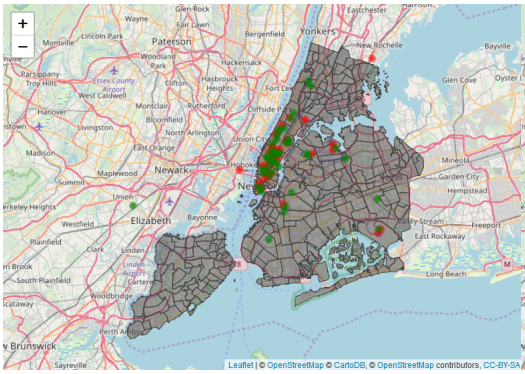

Figure 13: k-means with k=50

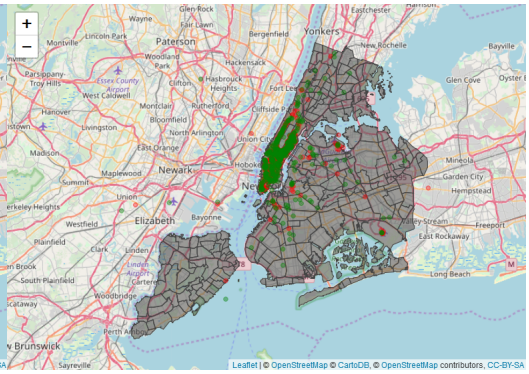

Figure 14: k-means with k=500

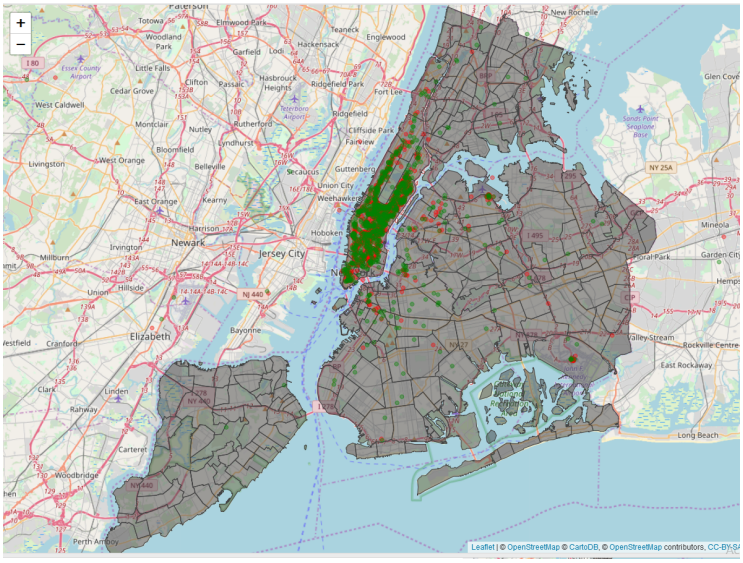

Figure 15: k-means with k=1000

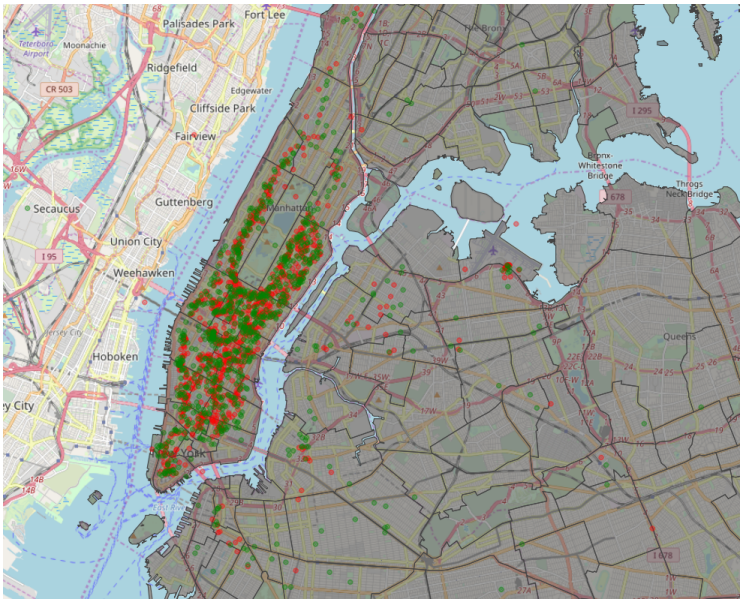

Figure 16: k=1000 zoomed data

9 Appendix: Visualization

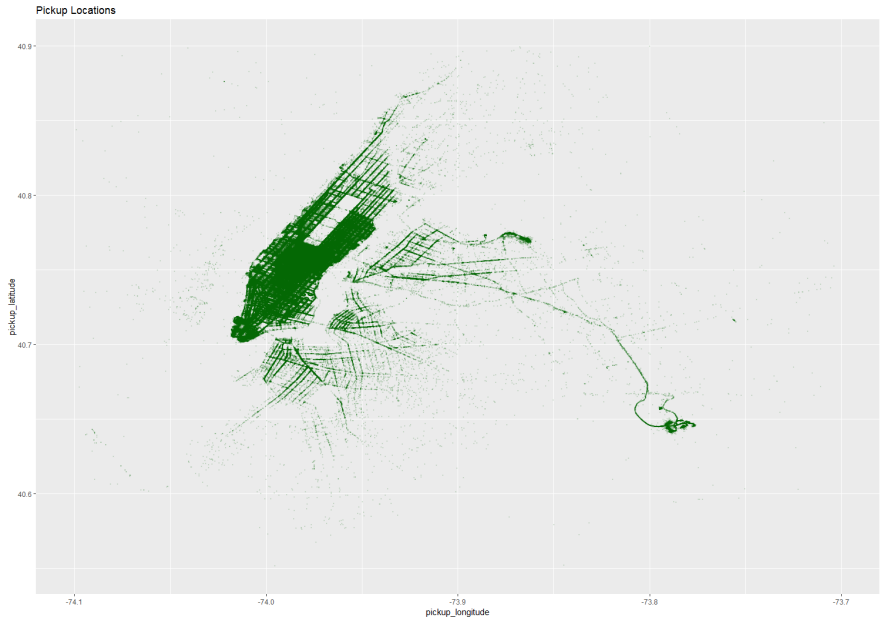

Figure 17: Raw dataset pickup points

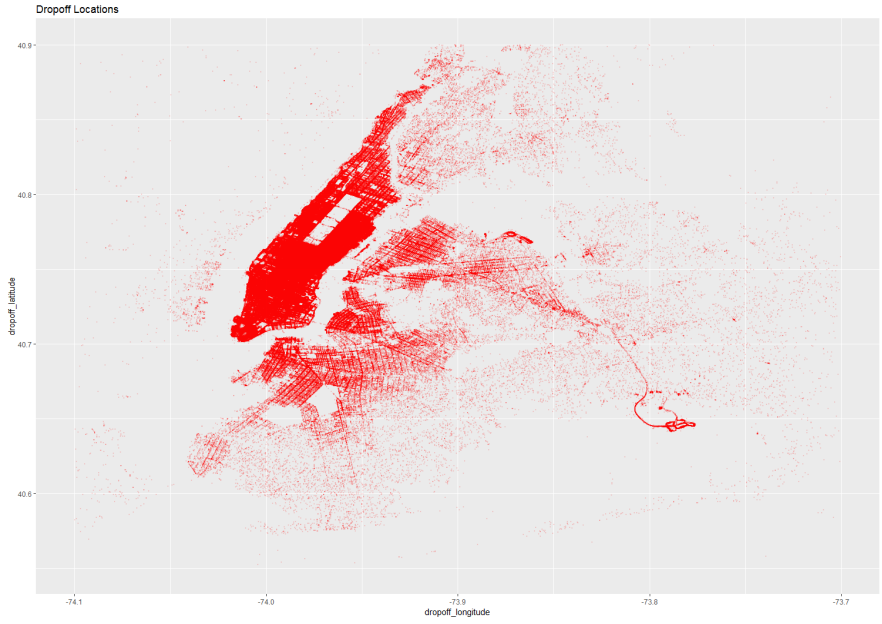

Figure 18: Raw dataset drop off points

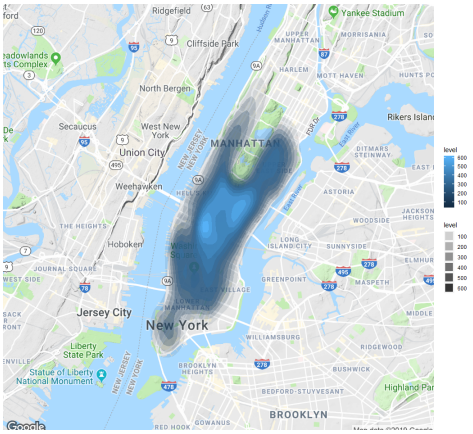

Figure 19: Coordinates of where the taxi ride started (heat map)

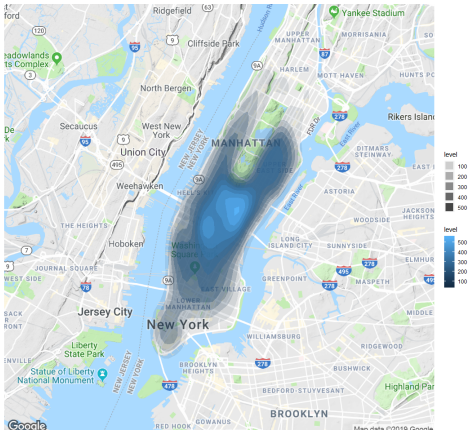

Figure 20: Coordinates of where the taxi ride ended (heat map)

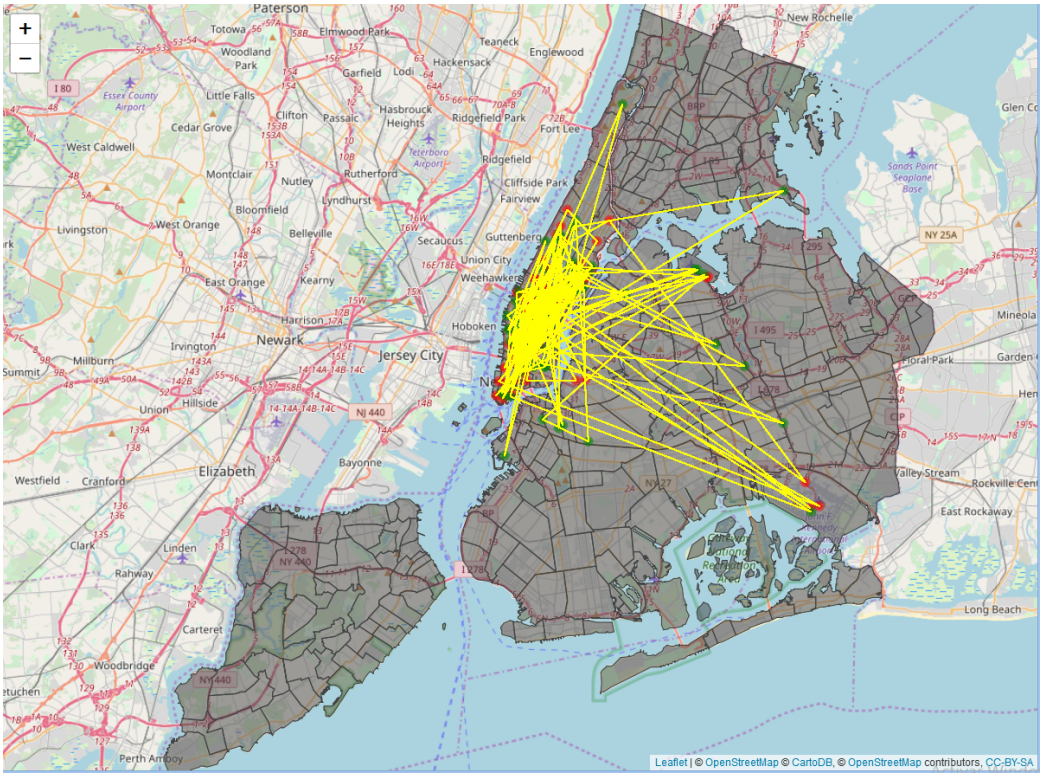

Figure 21: 100 pickup/drop-off points represented and gathered by a line