# Estimating Taxi Trips Duration with Supervised Machine Learning

**Alexander D. Rodriguez, Santhosh Malgireddy**
University of Oklahoma
660 Parrington Oval, Norman, OK 73019

**Abstract**

In this paper we experimented with some supervised learning techniques to predict the duration of taxi trips, for which we coded our own ML algorithms. Even though we could get a predictive power very near to a reliable implementation as scikit-learn, we had computational troubles because our decision tree takes much more time than its scikit-learn counterpart.

## Introduction

For our long SL project, the dataset that we choose was originally published by the NYC Taxi and Limousine Commission (TLC). The data was sampled and cleaned for the purposes of Kaggle playground competition. Based on individual trip attributes, we should predict the duration of each trip in the test set. In this context, we hypothesized that by our machine learning algorithms could be as predictive as the ones in scikit-learn and that we will be able to predict the trip duration with a Root Mean Squared Logarithmic Error (RMSLE) lower than 50% of the competitors.

## Related Work

Zhang et al. (2016) had a similar problem to ours in a bike-sharing system where they wanted to predict the trip destination and duration. For the trip destination inference they used MART (Multiple Additive Regression Trees) and the estimation of the trip duration they used Lasso. To build these models they used features from the user (gender, age), the departure time, and the bike stations location. In our problem, we don't have personal information about the users of the service, but still we can study if the taxis from one company differ with the ones from the other. In addition, as suggested in this paper, we are going to build features related to the departure time (month, weekday, hour of trip), and try with a Lasso model for comparison. Furthermore, this paper suggests using Manhattan distance instead of Euclidean as it is accurate in estimating travel distances in a city. We will adopt this suggestion if we need to use distances.

Li et al. (2015) worked in a very sophisticated model to estimate the traffic in a bike-sharing system by subsuming several other machine learning models (for usage prediction and trip duration) along with a Markov chain (for the weather). A contribution that we think may help to predictability of our models is the addition of meteorological conditions and relevant event in the city. While they use these two pieces of information for the estimation of the usage of bicycles, we can use these data for other purposes, as the estimation of usage is not important. We can use the meteorological conditions to see if it affects the duration of the trips, as in a rainy-day people may use more their cars and we could have more traffic, and, consequently, more duration for the trips. In addition, knowing the upcoming events to be held in the city (especially near the start and end of the trip) may also help to estimate the trip duration as they can tell us something about the traffic.

Balan et al. (2011) presented a real-time trip information system that provides passengers with the expected fare and trip duration of the taxi ride they are planning to take. They created a predictive model for this by averaging the duration and fare of the most similar trips. To find similar trips, they considered two aspects: time frame and origin-destination. The time frame was inserted as hours and day of the week. The origin-destination had more preprocessing as they converted the geographic coordinates (latitude and longitude) to zone numbers. This was done because using geographic coordinates would make slow and tedious the search for the most similar trips. For the zoning process, they tried a static and a dynamic approach. The static was based on pre-chosen zone sizes, and the dynamic was based on dynamically adjustable zone sizes for each trip that were based on similarity to the historical records. Even though the dynamic approach was more accurate, it usually took longer to compute than the static. Thus, the static was chosen. Even though we don't have time limitation as they did (because we are not building a real-time system), we will adopt their static zoning approach because ultimately this model should be implemented in a real-time system; otherwise, this model would be useless. Having our static zones, we will experiment using the most similar trips (as this paper) using kernel regression and other with another more sophisticated regression technique.

Ferreira et al. (2013) created a visualization tool for big data related to taxi trips in a city called TaxiViz, which has the capability of connecting to the database and retrieve and plot the required data in a timely manner. Note that they didn't perform prediction in their work, but they got interesting conclusions from their visualizations. In their case study, they examine how trip frequency vary for each day of the week for each neighborhood. They suggest that by looking at these patterns we can get a sense of the socio-economic level of each neighborhood. Additionally, their work recognizes the importance of considering transportation hubs to understand the flow of taxis around the city. They mention airports and major train stations to be major hubs in NYC. This approach of analysis is similar to the one presented by Balan et al. (2011) as they both analyze taxi trips by location. Thus, we decide to not only add zoning to our model, but also pinpoint trips related to these major transportation hubs as they may have some predictive value.

Veloso et al. (2011) used taxi trip data to examine human mobility in Lisbon, in which they used additional data sources to approach this complex problem. One of the data sources used is a collection of more than 10,000 points of interest (POIs), which were grouped in eight categories (e.g. recreation, education, shopping). These POIs were mapped to a squared gridded map of the city. We'll search for this data and, if available, we plan to add it to our model. Additionally, they considered meteorological conditions as Lin et al. (2015) did; however, they grouped them in three states: sunny, cloudy and rainy. We plan to use this classification in our model because (as we mentioned before) weather can be a good predictor for trip duration. The rest of this work is related to interesting visualizations of human mobility that they can do with a more complete dataset than ours.

Yuan et al. (2011) presented a probabilistic framework to recommend taxi drivers where to find their next customers based on people's mobility patterns, and people where to move to pick up taxis more easily by understanding the taxi drivers' pick-up behavior. This work classifies drivers by their performance on getting more passengers, which made us think about doing something similar to our problem. We could classify drivers using clustering based on the time they take to reach their destination because there are slow and fast drivers. Having several observations from the same drive would allow us to know how about above or below the average velocity (distance/time seen in the train dataset), which will provide us insight about the pace that the driver usually goes. Therefore, we could classify drivers in three: slow, medium, and fast pace. However, this could not be implemented because the IDs for the drivers were unique (i.e. only one observation per driver).

Because every predictive model needs a good exploratory data analysis, we also reviewed some data visualization sources.

Schneider (2017) went through several visualizations searching for interesting relationships within the available dataset and datasets from other sources. He considered in his analysis taxi rides going to major transportation hubs, as suggested by Ferreira et al. (2013), noticing the following when examining rides from a NYC neighborhood and an airport. The variability of the duration of the trips (over hour of the day) is similar to every airport within the same origin neighborhood. However, for different neighbors the variability may be very different. Wondering which variable we could use to explain that variability, we think that it may be explained by considering the events that are developed on each neighborhood, which may bring more traffic, and consequently, a higher trip duration. These findings support our idea taken from Li et al. (2015) about using the events around the city as input for our model.

The Kaggle user ZymonStrength (2017) published a kernel about data visualization for the competition. Looking at it was important because it helped to recognize that there are several peculiarities of our dataset. First, there are some observations which its coordinates indicate a place outside NYC, and sometimes the places are in the middle of the sea. Removing these wrong coordinates would be the best for our model to not learn from those outliers. Second, there are trips that 0 passengers were reported, and only those presented negative values for trip duration. This could suggest that these passengers were charged because they canceled for making the taxi go to their place, but they didn't get in. It would be worth to investigate to decide either delete them or not.

## Approach

We started with exploratory analysis on the dataset to have an idea of how to clean and build new predictive features. Then, we coded our own ML algorithms and compared their performance against their counterparts in scikit-learn.

### Pre-processing

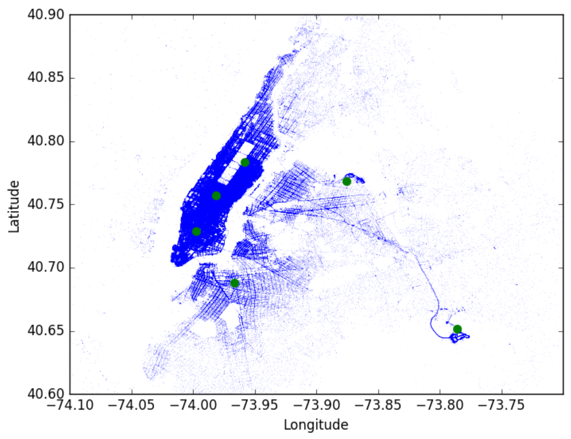

*Fig. 2 New York taxi trips positions. Blue: pickup and drop-off positions filtered. Green: center of positions clusters*

On the data-preprocessing side, we worked on several features. First, we converted precise time-day information to something three simple classes: rush hours, early morning, and other. We also found external weather data available and we added the temperature in Celsius, and whether it rained or snowed in that day. Furthermore, we found that there were trips that were too far from the limits of the city, so we proceed to delete them (they were very few). Additionally, we clustered locations finding that 6 is the most suitable number of clusters for the trips that we have in the train dataset (Figure 2). Finally, we calculated the Manhattan distance of each trip and after that we deleted all the columns that we replaced with more our new columns.

## Learning Algorithms

We coded two algorithms from scratch: elastic net with SGD (stochastic gradient descend) and decision tree.

For elastic net, the objective function is defined as follows.

$$E = 0.5\,(Y - \hat{\beta}X)^2 + \alpha \cdot \lambda \cdot \|\hat{\beta}\|_1 + \alpha \cdot (1-\lambda) \cdot \|\hat{\beta}\|_2$$

The partial derivatives of the L1 and L2 penalties are sign($\beta_i$) and $\beta_i$, respectively; however, in the case of L1, this is a simplification because the derivative of L1 is not the function sign in all the domain (because it is not differentiable for $\beta_i$=0). Other more complex methods as coordinate descend can handle this in a more proper way, but as we wanted to use gradient descend, we made this simplification. Thus, the update rule for the coefficients is the following.

$$\beta_j \leftarrow \beta_j + \gamma \cdot \left( (y_i - \beta_j \cdot x_{ij}) \cdot x_{ij} - \alpha \cdot \lambda \cdot sign(\beta_j) - \alpha \cdot (1-\lambda) \cdot \beta_j \right)$$

where $\gamma$ is the learning rate, $\alpha$ is a constant that multiplies the regularization terms, and $\lambda$ is the L1 ratio, a.k.a. mixing parameter for the regularization terms. In addition, SGD request that on each epoch/iteration the dataset is shuffled to ensure robustness.

In addition to this optimization approach, we tried Simulated Annealing to see how the results can change. Here is where the *novelty* of our work lies because we added something to the Simulated Annealing to make it work more efficiently. Instead of evaluating how well a solution (i.e. a set of coefficients) is performing in the whole train dataset to then move to other position in the solution hyperspace, we evaluated the error in a subset of the train data. Because this subset is small and randomly chosen, it makes the training faster and more robust. We named this approach as the *Batch Simulated Annealing*. There are several parameters that can be tuned in this method (e.g. initial temperature, temperature length, cooling ratio). However, we chose two that we hypothesize as important: batch size and the size of the step of the neighbor generator.

On the other hand, CART (Classification and Regression Tree) is a recursive partitioning algorithm with the representation of pure binary tree in which the splits are in binary nature (i.e. at each level the decision tree splits up the input node into two sub nodes based on the variable condition). Each root node represents a single input variable and a split point on that variable and the leaf nodes of the tree contain an output variable which is used to make a prediction. The splitting rules that we consider for our regression tree to grow are mean squared error (MSE) reduction and mean absolute error (MAE) reduction. Our algorithm steps are the following.

1. Data: Get a list of independent variables and their type (either categorical or continuous)
2. Best Split: Find Best Split for each of the independent variables with MSE reduction criterion or MAE reduction criterion
3. Best Variable: Select the best variable for the split
4. Split: split the input data into left and right nodes
5. repeat step 2-4 for each node until it meets stopping criterion (in our case, max depth or min observations in leaf)

As stated above, in step 2, if the input variable is categorical in nature we do check a binary condition for each value of that variable, and if it is continuous in nature we do split that variable values into k=100 number of equal bins and will consider each bin for binary condition for splitting. Best splitting feature value will be picked up based on splitting criterion for which the value is minimum. This procedure is repeated until it reaches the given depth of the tree.

To sum up, the learning algorithms that we have are two, and we made some substantial modifications to each of them to set up four experiments:

1. Elastic nets with SGD
2. Elastic nets with Batch Simulated Annealing
3. Decision tree with MSE split criterion
4. Decision tree with MAE split criterion

## Results

Before using our algorithms for predictions in the test dataset, we tuned the parameters in splitting the train dataset into a new train and validation dataset. We made this split several times and each time choosing a different randomly selected validation dataset. For the elastic net, we modified the learning rate and the l ratio (as in Figure 3). The best combination of this two were 1e-5 and 0.4 respectively.

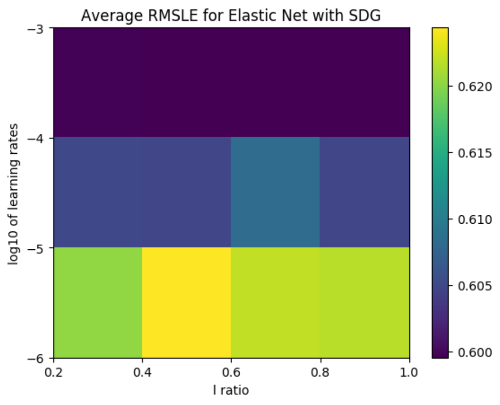

*Fig. 3 Elastic net with SGD tuning. Best results were with l ratio of 0.4 and learning rate of 1e-5.*

Similarly, for the decision trees we tuned other two parameters: max depth and min number of observation per leaf

(Figure 4). It can be seen that the parameters that change the most the results are l ratio and maximum depth, respectively.

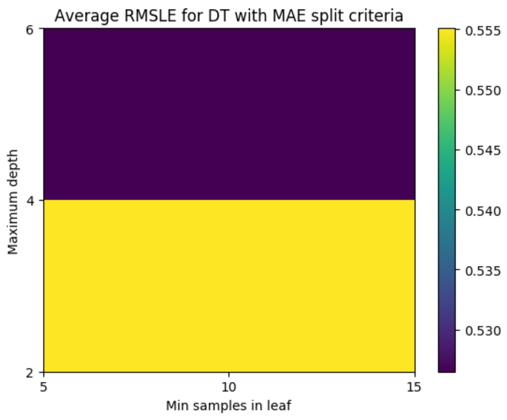

*Fig. 4 Decision tree with MAE split criterion tuning. Best results were with max depth of 4 and min sample in leaf of 5.*

The tuning results for MSE split criterion are similar; however, the ones for Batch Simulated Annealing are not as the error across all the combinations are is very high. Still, it's important to note that the batch size and the size of the step of the neighbor generator were significant tuning parameters (i.e. the variation across different values of these parameters was important). Due to space constraints, we restrain of including their contour charts.

| **ML algorithm** | **Experiment** | **RMSLE** | **RMSLE Standard Dev.** |
|---|---|---|---|
| Elastic Nets | With Stochastic Gradient Descend | 0.6110* | 0.0206 |
| | With Simulated Annealing | 9.5907* | 4.1276 |
| Decision Trees | With MSE split criterion | 0.6901 | NA |
| | With MAE split criterion | 0.5253 | NA |

*Table 1 Results with our coded ML algorithms.*
** Average error from 30 repetitions*

Table 1 presents the results of our coded ML algorithms, each of them have two experiments as explained before. Because the two approaches in elastic nets are stochastic, they were run 30 times each to get the expected RMSLE. Note that the one that performs the best is DT with MAE split criterion. Note that the Simulated Annealing approach performed bad in comparison with the rest, which makes it less desirable.

It is important to make a note in some implementation details for our decision tree. The average training time for it was 3 hours for approx. 100,000 observations in a 3.4 GHz machine. This high training time may be a consequence of the number of iterations for the continuous feature values. Even though we sort them to then split them into bins based on the feature range of values (this to avoid splitting every continuous value of the feature), this approach helped but didn't completely solve the problem of our model training time.

| **ML algorithm** | **Tuned parameters** | **RMSLE** |
|---|---|---|
| Elastic Net with SGD | eta0: 1e-3, 1e-4, **1e-5**, 1e-6 l1_ratio: 0.2, 0.4, **0.6**, 0.8, 1.0 | 0.5975* |
| Decision Trees with MSE criterion | Max depth 3,4,5,**6** Min sample 4,**5**,6 | 0.4881 |
| Random Forest | Max depth 3,4,5,**6** Min sample 4,**5**,6 | 0.4718 |
| Gradient Boosted Trees | Max depth 3,4,5,**6** Min sample 4,**5**,6 | 0.4702 |

*Table 2 Results with scikit-learn algorithms. Chosen values for tuned parameters in red.*
** Average error from 30 repetitions. Standard dev.: 0.0007*

In Table 2, the results from scikit-learn implementations are presented with their tuning parameters. Comparing our results with them we have the following conclusions. Our elastic net with SGD implementation was close in performance to the one in scikit-learn; however, the mean values for the RMSLE are statistically different. In addition, the decision tree with MSE from scikit-learn outperformed both of our implementations of decision trees. However, it is important to note the big difference that the criteria of split can make in the predictive performance.

With the results from our decision tree with MAE split criterion, we could reach the top 65% of the Kaggle leaderboard. If we used the boosted trees from scikit-learn, we could read up to the top 55%. This suggests that to be competitive in this competition, we have to work more in preprocessing the data, add more external data that is available online and use more sophisticated modeling techniques as model stacking.

## Conclusions

In conclusion, our approach and coded ML algorithms could reasonably predict the taxi trip duration. Our ML algorithms could not perform as well as the ones implemented in scikit-learn, but they were close. Our elastic net was trained in a similar time than the one in scikit-learn; however, our decision tree took way more time. Still, our decision tree effectively accomplished the challenge of handling continuous variables and categorical variables of many categories. Finally, we suggest building more features, add more external data and use more sophisticated approaches for modeling.

## References

● Bhatti, S., Desmaison, A., Miksik, O., Nardelli, N., Siddharth, N., & Torr, P. H. (2016). Playing doom with slam-augmented deep reinforcement learning. arXiv preprint arXiv:1612.00380.

● Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaśkowski, W. (2016, September). Vizdoom: A doom-based ai research platform for visual reinforcement learning. In Computational Intelligence and Games (CIG), 2016 IEEE Conference on (pp. 1-8). IEEE.

● Lample, G., & Chaplot, D. S. (2017). Playing FPS Games with Deep Reinforcement Learning. In AAAI (pp. 2140-2146).

● McPartland, M., & Gallagher, M. (2008, October). Learning to be a Bot: Reinforcement Learning in Shooter Games. In AIIDE.

● Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1, no. 1. Cambridge: MIT press, 1998.

Harvard