NYC Taxi Travel Time Prediction through Leveraging Geographical Information

Sungjun Choi
MS Machine Learning
Carnegie Mellon University
sungjun2@cs.cmu.edu

Niles Christensen
MS Machine Learning
Carnegie Mellon University
nchriste@andrew.cmu.edu

Amrit Singhal
MS Machine Learning
Carnegie Mellon University
amritsin@cs.cmu.edu

Abstract

In this work, we present a method of leveraging known geographical separations to better estimate travel times by car between two points in a city, specifically New York City. This allows for the creation of multiple simpler models that can be combined into an ensemble by taking advantage of information about possible crossing points.

1 Introduction

In this work, we set out to find ways to accurately predict the time taken for a taxi to travel between two particular areas, given the starting area, ending area, and the time and date of the trip.

We sought to leverage domain knowledge of travelling within New York City. Specifically, New York City is divided into 5 boroughs, large regions of the city. Notably, many of these are divided by bodies of water which can only be crossed at certain locations, specifically bridges and tunnels. Furthermore, anyone who has spent time in New York will know that traffic conditions vary greatly between boroughs, meaning that using one model to predict traffic throughout the entire city may prove non-optimal.

To this end, we designed our models around the idea of using smaller sub-models to predict traffic that begins and ends in similar areas of the city, and combining the outputs of these models with reasonable guesses about how the regions were crossed between, i.e. which bridge or tunnel was used.

2 Background and Related Works

2.1 Gradient-Boosted Regression Trees (GBRT) and XGBoost

Regression trees, like decision trees, recursively split the feature space into approximately homogeneous subspaces [5]. A simple model, often the mean of training outputs, is fitted in each of the subspaces. Gradient-boosted regression trees or Gradient Boosting Machines(GBMs) is a boosted version of regression trees [3], similar to AdaBoost [2]. However, unlike AdaBoost, which iteratively reweights training samples based on classification correctness of the previous weak learner and chooses the next learner based on the weights, GBRTs choose trees to greedily approximate non-parametric gradient descent. In detail, GBRT first assumes an additive, or "boosted", model F of the form

$$F(\mathbf{x}; \{\beta_m, \theta_m\}_{m=1}^{M}) = \sum_{m=1}^{M} \beta_m f(x; \theta_m)$$

where M is the number of boosts (i.e. trees) and each $f(\,\cdot\,;\theta_m)$ is a tree. We minimize the risk

$$R(F) = \mathbb{E}_{\mathbf{x},y}\, L(F(\mathbf{x}), y) = \mathbb{E}_{\mathbf{x}}\left[\mathbb{E}_{y|\mathbf{x}}[L(F(\mathbf{x}), y)|\mathbf{x}]\right]$$

where L is the loss function for the task at hand. Friedman approaches this problem in a non-parametric manner and minimizes the risk by conducting steepest descent at each $\mathbf{x}$ to minimize $r(\mathbf{x}) = \mathbb{E}_{y|\mathbf{x}}[L(F(\mathbf{x}), y)|\mathbf{x}]$. In other words, we assume the solution takes the form

$$F(\mathbf{x}) = \sum_{m=0}^{M} \beta_m g_m(\mathbf{x})$$

where $f_0(\mathbf{x})$ is the initial model, $\beta_0 = 1$, β_m is obtained by line search, and

$$g_m(\mathbf{x}) = -\frac{\partial \mathbb{E}_{y|\mathbf{x}}[L(F(\mathbf{x}), y)|\mathbf{x}]}{\partial F(\mathbf{x})} = -\mathbb{E}_{y|\mathbf{x}}\left[\frac{\partial L(F(\mathbf{x}), y)}{\partial F(\mathbf{x})}\bigg|\,\mathbf{x}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$$

where the last equality holds under sufficient assumptions.

We approximate the above on the whole domain to obtain our model $\hat{F}$ by utilizing parametrized models (regression trees specifically) as follows: at each iteration m, we first approximate $g_m(\mathbf{x})$ using its "data-based analogue" [3]:

$$\hat{g}_m(\mathbf{x}_i) = -\left[\frac{\partial L(\hat{F}(\mathbf{x}_i), y)}{\partial \hat{F}(\mathbf{x}_i)}\right]_{\hat{F}(\mathbf{x})=\hat{F}_{m-1}(\mathbf{x})}$$

Then we search for the regression tree that best fits the above analogues for the whole sample:

$$\hat{\theta}_m = \arg\min_{\theta,\rho} \sum_{i=1}^{N} \left(\hat{g}_m(\mathbf{x}_i) - \rho f(\mathbf{x};\theta)\right)^2$$

Again, $\hat{\beta}_m$ is chosen via line search, and using $\hat{\theta}_m$ and $\hat{\beta}_m$ we update our model:

$$\hat{F}_m(\mathbf{x}) = \hat{F}_{m-1}(\mathbf{x}) + \hat{\beta}_m f(\mathbf{x};\hat{\theta}_m)$$

The same author has suggested an improvement on the algorithm named Stochastic GBM [4], in which boostrapping-like subsampling procedure is added to add randomization and make the model more robust, although here we sample without replacement. Also, to allow faster processing large-scale datasets, a model named XGBoost [1] brings in approximate algorithms to parallelize the GBRT model and allow for GPU optimization.

3 Dataset

3.1 Superboroughs

We began by dividing the city into groups of boroughs, which we termed "Superboroughs." Each of these superboroughs consists of boroughs which are well-connected to each other, and separated from the others by bodies of water.

In the end, our superboroughs were the following:

Superborough 1	Manhattan, Bronx, & EWR
Superborough 2	Brooklyn & Queens
Superborough 3	Staten Island

Figure 1: The groupings of boroughs to form superboroughs

Of note is the presence of EWR in the first superborough. This is remarkable because EWR is not a borough (being the airport code for Newark Airport), and is separated by the Hudson river from Manhattan and the Bronx. Newark airport is unusual because it is New Jersey, but is still served by New York City taxis. As such, it had to be artificially added to the existing boroughs.

Though Newark is closer to Staten Island than it is to either Manhattan or the Bronx, we decided to include it with the latter group because it made reasonable the assumption that a trip would never include more than two super boroughs, i.e. that superboroughs are crossed between only once. This is because Manhattan provides the most direct route to Newark for most of the city. As an example, were we to have grouped Newark with Staten Island, the most reasonable route from Queens to Newark, which passes through Manhattan, would have crossed first from superborough 2 into superborough 1, and then into superborough 3, requiring two crossings. Instead, that same trip now crosses between superboroughs only once, from superborough 2 into superborough 1.

3.2 Feature Engineering

We supplemented our dataset with several pieces of information. Through NYC Open Data [7], we found information about which borough each taxi zone was in, as well as GPS coordinates describing the boundaries of each taxi zone. We converted the GPS coordinates to a single representative point by drawing a bounding box around each taxi zone and taking its center. We did this rather than taking the mean of the points because a zone with one straight side and one "squiggly" side would have more points on the non-straight side, as it requires more points to describe. A simple mean would therefore be biased away from straight edges, which is undesirable behaviour.

3.3 Data Processing

The size of the dataset we used to train our models proved to be a significant challenge. Our dataset was over 5GB, and our approach of training smaller models for each borough required the ability to efficiently retrieve specific subsets of it according to flexible criteria. To this end, we wrote code to load our data into a SQL database.

However, we found that we still needed to write our queries with care to avoid extremely expensive operations. As an example, we had to design our models consuming our data to accept batch size of arbitrary size, as returning a fixed number of rows involved running joins over the entire dataset, and then returning a subset of it with the LIMIT keyword. This proved to be intractably slow. Instead, we switched to selecting a limited number of rows with the LIMIT keyword, and then running joins on this smaller amount of data, unfortunately returning an unpredictable number of rows. Furthermore, we had to take care with operations such as ORDER BY RANDOM which would operate over the entire table if used incautiously.

3.3.1 Skewness of Data

Another challenge posed by our dataset was the fact that our rides were not evenly distributed across the boroughs. The vast majority of rides either start or end in Manhattan, as can be seen in Figure 2. Furthermore, the majority of our rides never cross between superboroughs, as can be seen in Figure 3. This meant that care had to be taken to ensure that we performed well on all types of data rather than just Manhattan.

Borough	Number of Rides
Manhattan	63,809,748
Bronx	440,921
EWR	128,081
Brooklyn	3,497,591
Queens	6,318,806
Staten Island	16,047

Superborough	Number of Rides
Manhattan, Bronx, & EWR	63,929,565
Brooklyn & Queens	9,209,372
Staten Island	16,047

Figure 2: The number of rides starting and ending in each borough, and in each superborough. Note that the sum of rides in this table is greater than the total number of rides, as rides starting and ending in different boroughs are double counted

3.3.2 Data Issues

Manually examining our data revealed many strange artifacts. There are some rides which are far longer than seem reasonable, with some lasting for more than 12 hours. Even more concerningly,

Rides between Superboroughs	Rides within Superboroughs
6,947,045	59,293,762

Figure 3: The number of rides starting and ending in the same superborough vs those that cross superboroughs

while there are no rides that last for more than 24 hours, there is a large spike of rides that lasted just a few minutes less than 24 hours (see Figure 5). However, these accounted for only a small portion of our data, and the rest appear to be reasonably distributed (see Figure 4).

We were unable to get good testing accuracy when we trained our model on the entire dataset, as these very long rides seem random and our models could not effectively predict them. To compensate for this, we removed from our dataset all points falling outside of 1 sample standard deviation. This retained a very large proportion of our data (99.46%), and allowed for the results we present here.

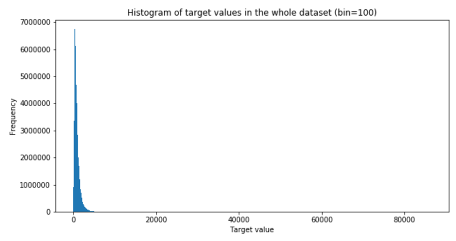

Figure 4: Overall distribution of data

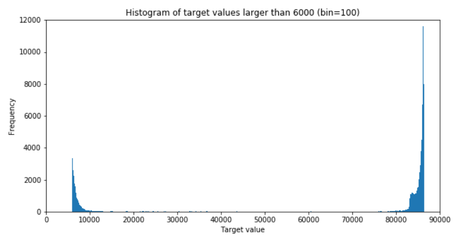

Figure 5: The distribution of the data near the right tail

4 Methods

We present two approaches to solving the problem of estimating travel time of a NYC taxi cab, which we compare against simple baselines. Both of our approaches follow the same high level logic:

1. Split a ride across superboroughs into constituent parts
2. Use a relatively simple learner to predict the amount of time taken by each part
3. Somehow combine our estimates of each part into a coherent total prediction.

4.1 Baseline

We began with three baselines, the first of which was simply the average time it takes to complete a trip, completely disregarding any information about the specific trip being predicted. This model performed poorly, but was useful in gauging our degree of improvement. The second baseline model was a multiple linear regression model. Both baseline models were trained on both the whole dataset as well as on each superborough. Finally, we trained an XGBoost model on the whole dataset to better highlight the effect of our fine-grained approach.

4.2 XGBoost

We train an XGBoost model per superborough, the details of which are given in Section 5. Then, to predict the durations of cross-superborough trips, we first iterate through each bridge connecting the pick-up (PU) and drop-off (DO) superboroughs. We compute the predicted time taken for a trip via each bridge by adding the duration of the trip from the PU location to the PU-superborough side of the bridge and the duration from the DO-superborough side of the bridge to the DO location.

We assume that a NYC taxi driver will have knowledge of how long a trip via each bridge will take, and that they will want to take the fastest route. To approximate this, we predict that the time taken will be the amount of time taken to get to the destination via the bridge which minimizes travel time.

4.3 Neural Networks

4.3.1 Motivation

While the performance of XGBoost was considerably better than that of baseline approaches, this approach still had some limitations.

1. The learning was done only for each superborough model and heuristic measures were used to get predictions for the bridge to be used for the cross-superborough travels.
2. This approach does not allow us to predict the time taken to travel over a given bridge, forcing us to effectively ignore this time.
3. The model was unable to learn from cross-superborough data points. This was because we do not know what portion of a trip was spent in each superborough, so we cannot update each individual model with these travel times.

The aforementioned limitations can be overcome with the use of neural networks. We used a new approach, in which we train separate networks to predict travel time in each superborough, analogous to the previously-used individual regression trees. The flexibility of neural networks allowed for these models to be combined in an ensemble with sub-networks to predict the bridge used for the trip and how long was spent on this bridge. This allows us to combine the predictions of each borough network for the cross-borough trips, and backpropagate the combined loss through all the branches of the network. Thus, our ensemble model can make use of the cross-superborough data points, and is not limited to relying on heuristic measures for predictions.

4.3.2 Model Overview

Our ensemble consists of three different types of network:

1. Superborough Network: Predicts the travel time between two zones within a superborough. We have 3 superborough models, one for each superborough.
2. Selector Network: Predicts the probability of each bridge having been used to cross between superboroughs. We have a total of 6 selector networks, one for each ordered pair of superboroughs.
3. Bridge Network: Predicts the amount of time taken to cross the bridge.

For any cross-superborough data point, the feature vector consists of three distinct sections: pick up data information (**PU**), drop off data information (**DO**) and the date, time, and weekday information (**DT**). We begin by choosing the selector and superborough networks corresponding to our pickup and dropoff superboroughs.

The feature vector is first fed to the selector network, which predicts which bridge the trip was likely to have crossed. The final dense layer of the selector network uses softmax activation, and thus predicts a vector containing the probability of having taken each bridge. This probability vector is multiplied with constant matrices containing information about the ends of each bridge.

This output is in turn used create the inputs to each of the superborough models, which are the original vector modified to have new pickup and dropoff information corresponding to the ends of the selected bridge, which we denote **BR1** and **BR2**. The output of the selector network is also used to create the bridge feature vector **BR**, which contains information about the endpoints of the bridge (**BR1** and **BR2**), and the date and time (**DT**).

Thus, the selector network separates the trip into three separate chunks - (1) from pickup location to the first end of bridge, (2) across the bridge and (3) from the second end of bridge to the drop off location.

Next, we predict the travel time across all three parts of the journey. The superborough models predict the travel time within each superborough, and the bridge network predicts the time taken to cross the bridge. The sum of these components is our prediction for the duration of the entire trip.

This ensemble architecture can be seen in Figure 6.

Our model calculates the loss between our prediction and the true travel time. This loss is then backpropagated through the bridge network and relevant superborough networks.

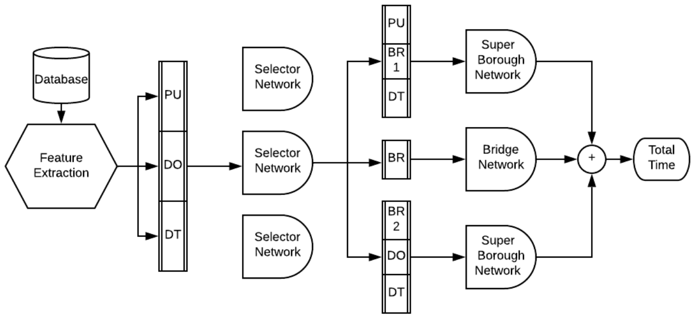

Figure 6: Network architecture for the ensemble neural net model

5 Experiments and Results

5.1 Feature Engineering

Multiple feature representations were extracted and tried, differing in the aspects listed below:

1. whether the date-time and weekday representations were one-hot for retaining independence or numerical for retaining a sense of continuity
2. whether the actual one-hot encoded location IDs in the feature vector were included, or if we relied only on the GPS coordinates for the starting and ending zones
3. in what order our features were concatenated to form our feature vector

5.2 Implementation Details

The implementation can be found in the following open-source Github repository: `https://github.com/nilesc/NYCETA`

5.2.1 Baselines

We computed sample means and fitted linear regression models using Scikit-Learn package [8] for the whole dataset as well as for each superborough, and computed loss across the respective datasets.

5.2.2 XGBoost

All models have been trained with row subsampling rate of 0.5 and learning rate of 0.1. We trained them until validation error stopped decreasing for 10 consecutive iterations to ensure it was fully trained. Different maximum depths were applied based on the size of the dataset to avoid overfitting. The baseline and Manhattan+ superborough models had maximum depth 10, the Queens+ superborough model had maximum depth 6, and the Staten Island model had maximum depth 3. We validated the superborough models on a separate holdout set with size 20% to choose the best ones to compose the ensemble model. Once we obtained the ensemble model, we tested its performance on a fraction of cross-superborough datapoints since we already have chosen each superborough model based on intra-superborough performance. Models have been implemented using XGBoost [1], Scikit-Learn [8], and Scipy [9] packages.

5.2.3 Neural Network

Model Architecture The architecture details for each of the smaller networks in the ensemble are as follows:

1. Superborough model: Each superborough model consists of two dense hidden layers of sizes 200 and 50, and final dense output layer. Each of the layers have ReLU activation.
2. Selector Network: Each selector network consists of two dense hidden layers, the first with size 100, and the second with size equal to the number of bridge to select from. The first layer uses ReLU activation and the second uses softmax activation.

3. Bridge Network: The bridge network consists of two dense hidden layers of sizes 100 and 50, and a final dense output layer. Each of the layers have ReLU activation.

The model was implemented using `tensorflow 2.0` [6].

Training individual superborough models Each superborough model was trained on trips that start and end in that superborough. We used the Adam optimizer with a learning rate 10^{-3} for performing the updates.

Training ensemble Each superborough model in the ensemble was initialised with the pre-trained weights. Other networks were initialized randomly. We used the Adam optimizer for performing the updates. Superborough models had a smaller learning rate of 5×10^{-4}, while the other models had a learning rate of 10^{-3}.

5.3 Results

The metric used for the results of the approaches is the RMSE (root mean squared error) between the predicted and true values of the travel times.

	Sample Mean	Linear Regression	XGBoost
Testing RMSE	638.14	505.99	269.25

Figure 7: Testing RMSE of sample mean, linear regression and XGBoost on entire dataset

	Sample Mean	Linear Regression	Baseline XGBoost	Specific XGBoost	Neural Networks
Manhattan+	462.43	427.27	248.12	245.95	263.68
Queens+	857.94	600.26	310.61	335.50	346.61
Staten Island	603.77	564.13	367.90	538.99	570.02

Figure 8: Comparison of RMSE for various approaches trained on data points within a superborough

	Sample Mean Baseline	Ensemble XGBoost	Esemble Neural Net (Train)	Ensemble Neural Net (Test)
Manhattan+ $\leftrightarrow$ Queens+	846.59	473.96	385.07	419.60
Queens+ $\leftrightarrow$ Staten Island	671.64	482.13	666.01	685.55
Staten Island $\leftrightarrow$ Manhattan+	761.27	1509.71	516.81	570.03
Overall	846.88	480.26	385.19	419.97

Figure 9: Comparison of RMSE for various approaches trained on data points across superboroughs

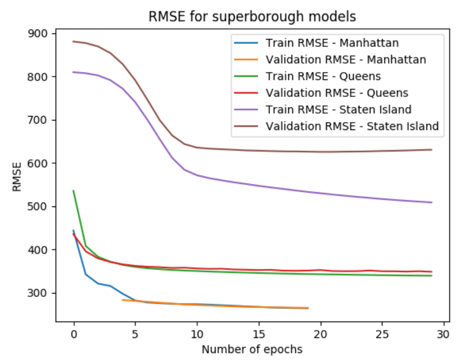

Figure 10: RMSE for superborough neural network trainings

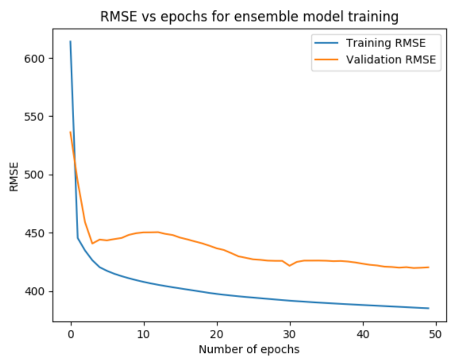

Figure 11: RMSE for ensemble neural network training

5.4 Discussion

There are some noteworthy points to discuss about the results:

- Both XGBoost and neural networks outperform the baselines by a large margin, in both the intra-borough as well as the inter-borough case. This suggests the validity of our approach.
- Performance is across the board good for Manhattan and poor for Staten Island. We hypothesize that this is related to the glut of data involving Manhattan, and the severe lack of training data for Staten Island as can be seen in Figures 2 and 3.
- XGBoost has better performance for each borough when taken individually. However, the neural network approach outperforms it on trips between boroughs. We suspect that this is due to its ability to train using trips that cross boroughs, which our XGBoost ensemble notably lacks.

6 Conclusion and Future Work

In this work, we propose two methods of creating ensemble models to predict travel times for taxis in New York City by leveraging information about the geography of the city to create simple learners for subsets of the task. We show that these models have considerably better performance than our baselines. Though our approaches have some limitations and could be improved as discussed below, we have made significant headway and feel that this is a promising avenue both for predicting travel times and for demonstrating the value of domain knowledge.

For the XGBoost ensemble model, we need a better heuristic for creating an ensemble. Currently the model not only has to consider every single bridge when predicting durations of cross-superborough trips (which is highly expensive), but is also ignoring the amount of time taken when crossing bridges when that could be a significant factor in certain cases. Simply adding GBRTs to predict bridge crossing times given superborough models would not suffice, since the errors from superborough models would propagate, not to mention aggravating the inefficiency issue. We may leverage graph algorithms, extra information on maps and traffic, etc., and it would be better if we could exploit cross-superborough datapoints when doing so.

For the neural network model, we believe that our model could be improved by providing it with more domain knowledge of our task. As an example, there are a number of freeways in NYC along which traffic tends to travel much more quickly. While our model could theoretically learn to account for these simply by seeing travel times in zones starting and ending near them, we believe we would see better results by explicitly encoding which zones contain freeways.

Additionally, our model uses identical architectures for predicting travel times within each borough. This disregards some information that could leveraged. For example, roads in Manhattan almost always form a grid. This could be leveraged to calculate the Manhattan distance between start and end locations, which might improve our performance. Additionally, using the anecdotal insight that cross-town and up/down-town traffic tends to behave differently, we could provide each of these components separately.

References

[1] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *CoRR* abs/1603.02754 (2016). arXiv: 1603.02754. URL: http://arxiv.org/abs/1603.02754.

[2] Yoav Freund and Robert E. Schapire. *A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting*. 1996.

[3] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine." In: *Ann. Statist.* 29.5 (Oct. 2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451. URL: https://doi.org/10.1214/aos/1013203451.

[4] Jerome H. Friedman. "Stochastic gradient boosting". In: *Computational Statistics Data Analysis* 38.4 (2002). Nonlinear Methods and Data Mining, pp. 367–378. ISSN: 0167-9473. DOI: https://doi.org/10.1016/S0167-9473(01)00065-2. URL: http://www.sciencedirect.com/science/article/pii/S0167947301000652.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001, pp. 305–308.

[6] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[7] *NYC Open Data*. `https://opendata.cityofnewyork.us/`.

[8] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[9] Pauli Virtanen et al. "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python". In: *arXiv e-prints*, arXiv:1907.10121 (July 2019), arXiv:1907.10121. arXiv: `1907.10121 [cs.MS]`.