Predicting MTA Bus Arrival Times in New York City

Man Geen Harold Li, CS 229 Final Project

Abstract

This final project sought to outperform the Metropolitan Transportation Authority's estimated time of bus arrivals in NYC. The best model, a cross-validated random forest algorithm, achieved MAPE and RMSE at around 20% higher than what the MTA had achieved. Next steps include experimenting with more relevant features, estimating with support vector regression and collecting more accurate data.

I. Introduction

Background

New York City is the most populated metropolitan in the United States; offering reliable and efficient transportation to 8 million people is no easy task. To improve the experience of bus riders, the Metropolitan Transportation Authority (MTA) released an API in January 2011 that tracks all buses and provides estimated time of arrival (ETA) of all future stops. This has helped over 2 million daily riders plan their commute more efficiently.

Goal

The goal of this project is to apply machine learning methods to further improve the accuracy of ETAs in the New York City bus network. More explicitly, the goal is to use information about buses (location, time, weather, etc) to predict the **time in seconds for a bus to reach a future destination**. We will be using **Mean Absolute Prediction Error (MAPE)** as the primary measure, followed by **Root Mean Squared Error (RMSE)** as the secondary measure. MAPE will help normalize error by time to destination, as certain stops are further away than others. That said, RMSE will provide a more intuitive understanding of forecast error, as it is measured in time (seconds), and it will further penalize inaccurate predictions. Providing more reliable forecasts will cause less confusion for bus riders and will help the MTA plan and deploy buses should there be issues with supply and demand.

Previous Work

While the MTA Bus API has been released for nearly 6 years, there has been little research the performance of these ETAs or even the proposal of better ETA methodologies. This may be a result of difficulty in gathering the data, as it was streamed and updated in real time and never archived.

Jeong (2004) used multiple linear regression models and artificial neural networks to forecast arrival times for 1 bus route in Houston, Texas. While Jeong made valuable contributions to the methods that can be used for bus time predictions, I hope to increase the scope of these methods to all bus routes in New York. I also plan to experiment with ensemble methods to see if they achieve superior results. In addition, I intend to experiment with features from different data sources in hope of further prediction improvements.

II. Data Capture and Processing

The MTA Bus API

The MTA Bus Data API publishes details of bus locations in 30-second increments. For each bus, it includes the route it is on, its distance from every future stop, and the estimated ETA for each step. There is no flag that designates whether a bus has arrived at a stop, so we denote all buses as having "arrived" when they are within 50 meters of their destination.

Collecting Data and Extracting Features

To collect observed and predicted arrivals, I captured all bus information in 30-second increments from October 18th, 2017 12:00 AM to October 20th 2017 11:59 PM. This data was streamed and requested in real-time, requiring three days of continuous runtime to gather. While this provided plenty of observations, the only meaningful features that could be extracted were: distance to stop, NYC borough (Manhattan, Queens, Brooklyn, Bronx and Staten Island), and hour of day.

To get more useful features that could affect arrival times, I collected weather data from the Dark Sky API. Given budget constraints, I was only able to collect hourly weather data at a 5-by-5 latitude-longitude grid, and attributed each bus' weather information to the nearest location with weather data. Features from this data source include: cloud cover, temperature, wind speed, visibility, UV index, pressure and dew point.

Furthermore, I collected population data using census tract information from the U.S. Census Bureau. The hypothesis is that higher population may lead to more traffic, thereby extending arrival times. By associating each bus' geographic location to census tract, we were able to capture population density and population for every prediction instance.

Removing Illogical Observations

While 27.2 million prediction instances were initially captured, 2.7 million of them had predicted arrivals as the same as the time of prediction, 3.6 million had predictions that were over 40 minutes off, and 11.2 million had predictions that implied a bus speed of over 60 km/h or less than 10 km/h. We decided on the 40 minute cutoff as there was a significant distribution of forecasts that were beyond that

threshold. We deemed those forecasts as invalid and removed them from the dataset.

Splitting into Train, Validation and Test Sets

Overall, with these data sources, we were able to collect 9.7 million prediction instances. The predictive variable is the time (in seconds) needed for a bus to reach a particular destination.

We divide the data into the training set, which contains prediction instances during October 18-19, 2017, and the test set, which will contain bus instances on October 20, 2017. Within the training set, we divide the data into a model training set (October 18 to October 19 11:59 AM), and a dev set (October 19 12:00 PM - 11:59 PM). The number of instances in each set are 4.7 million (48%), 2.1 million (22%), and 2.9 million (30%) respectively.

Exploratory Analysis of the Data

Initial analysis of published real-time bus time predictions suggest that the MTA prediction errors are largely intuitive. The RMSE graph on the top right of Figure 1 shows that ETA error increases on an absolute basis as distance increase. The MAPE graph on the top left, however, suggests that predictions at longer distances are in fact better on an absolute percentage basis.

We also found that the prediction errors are most significant during rush hour - between 8-11 AM and 5-8 PM, as shown in the bottom two graphs.

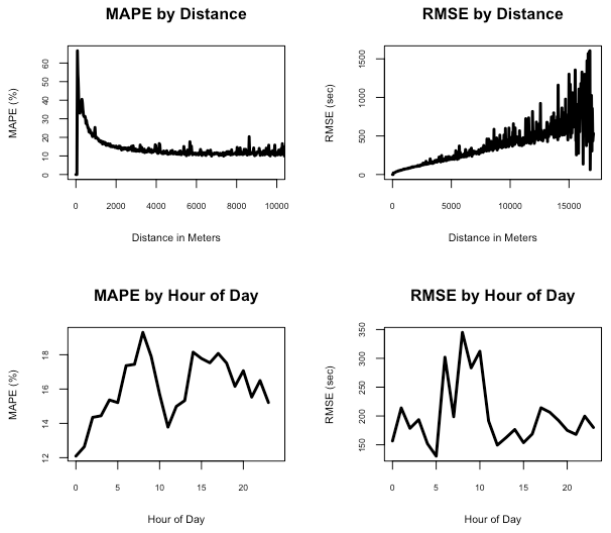

Fig. 1. MTA Forecast Error by Distance to Destination and Hour of Day

III. Models and Methodology

Two Baseline Metrics

Given that this project represents a regression problem, where we use a set of features to predict a continuous variable (time in seconds to bus stop), the first baseline model we use is to run a simple linear regression, with **distance from bus stop** as the only feature. We find that the MAPE is 28.99 % while the RMSE is 311.1 seconds on the test set. We want to use both metrics, as MAPE normalizes for all different times of arrival, while RMSE assess heavy penalties for highly inaccurate ETAs.

The second baseline to track comes from the MTA Bus Time ETA predictions themselves. The MAPE and RMSE for their estimates on the test set are 14.48 % and 203.3 seconds respectively.

Linear Regression with Feature Engineering

To improve on the first baseline model, we incorporate additional features to the linear regression. However, some of the variables were categorical with many unique values. For instance, there were over 150 bus lines captured, and there are 24 distinct hours of the day that had arrivals. Including these categorical variables expanded the dataset to many more columns, which made running the machine learning models on a local machine with millions of observations very slow. For that reason, I discarded the bus route features and bucketed the hours of day into distinct time periods: early hours (12 - 7AM), early rush hour (8 - 11AM), afternoon rush hours (5 - 8 PM), and other hours. Some other features had heavy tail distributions; for example, population density and population was severely right-tailed, so we transformed these features by taking the log.

We also added an interaction variable separating the distance feature from those that were under 300 meters and over 300 meters away. This is because the relationship between distance and time to arrival appear to be drastically different under and above the 300 meter threshold. Separating these as two features could significantly improve our error metrics.

TABLE I
List of Features for the ML Algorithms

Distance in Meters	Bus Direction	Cloud Cover	Temperature
Wind Speed	Visibility	UV Index	Pressure
Dew Point	Log Population	Log Density	Distance > 300 Meters
Morning Rush Hour	Evening Rush Hour	Early Morning	Manhattan

With these transformations, the new dataset now has 16 features to choose from (see Table I for list of features). We conduct a forward feature selection procedure, where the next best feature is added according to the F statistic, then run linear regressions on the selected features. We run this procedure 16 times with a limit on the number of features selected, and found that the best dev set error was achieved when only the top 2 features were selected. Not surprisingly, they were **Distance to Stop** and **Distance to Stop (over 300m)**. Figure 2 shows the cross-validation errors as a function of the number of features selected.

Linear Regression with Regularization

Another way to feature select and minimize overfitting in a linear regression environment is through regularization.

# Features	1	2	3	4	5	6	7
Dev Set RMSE	27.27	**26.34**	38.38	48.33	49.66	41.78	42.15
Dev Set MAPE	307.9	**307.8**	300.4	301.9	303.6	304.4	299.5

Fig. 2. Dev Set Errors by No. Features Selected

We decide to run lasso and ridge regression on the 16 transformed features, and find the best regularization term λ with cross-validation. Lasso and ridge regression, like linear regression, seek to minimize the squared errors between predicted and observed values, but add on a regularization term to penalize large coefficients in θ:

$$J(\theta) = \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \mid \theta_j \mid \quad (1)$$

$$J(\theta) = \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \quad (2)$$

We found that the optimal λ for Lasso and Ridge regression to be 10^3 and 10^{10} respectively (see Figures 3 and 4). In particular, all except **Distance to Stop** and **Distance to Stop (over 300m)** had non-zero coefficients for the Lasso regression.

λ	$10^{0.5}$	10^{1}	$10^{1.5}$	10^{2}	$10^{2.5}$	10^{3}	$10^{3.5}$	10^{4}
Dev Set RMSE	51.56	34.72	42.63	40.84	30.45	**27.38**	27.68	28.64
Dev Set MAPE	340.6	309.43	304.4	302.2	305.9	**307.9**	308.1	308.6

Fig. 3. Dev Set Errors by Lasso Regularization Parameter

λ	10^{6}	10^{7}	10^{8}	10^{9}	10^{10}	10^{11}	10^{12}	10^{13}
Dev Set RMSE	39.88	31.41	34.41	28.77	**26.67**	26.70	29.78	64.09
Dev Set MAPE	338.6	308.8	304.5	306.2	**307.6**	308.0	309.9	334.1

Fig. 4. Dev Set Errors by Ridge Regularization Parameter

Ensemble Methods: Boosting

In addition to linear regression, we attempted to experiment with ensemble methods, the set of machine learning algorithms that integrates many weak learners into strong learners. Boosting is an ensemble method that sequentially generates learnings of the same form, but weighs the training example such that subsequent learners focus on the examples that have large errors. Since this is a regression problem, we utilize Drucker's Adaboost.R2 algorithm (which is implemented in scikit-learn), where the weights of each example ($w_t^{(i)}$) were readjusted after each iteration as follows:

$$e_t^{(i)} = \frac{\mid y^{(i)} - h_t(x^{(i)}) \mid}{\max_j \mid y^{(j)} - h_t(x^{(j)}) \mid} \quad (3)$$

$$\epsilon_t = \sum_{i=1}^{m} e_t^{(i)} w_t^{(i)} \quad (4)$$

$$w_{t+1}^{(i)} = w_t^{(i)} (\frac{\epsilon_t}{1-\epsilon_t})^{1-e_t^{(i)}} / Z_t \quad (5)$$

Here, $e_t^{(i)}$ represents the error of the t^{th} learner on the i^{th} example, ϵ_t is the weighted learner error, and Z_t is a normalizer term.

Using a regression decision tree as the weak learner, we tuned the Adaboost algorithm by cross-validating the optimal maximum number of iterations with our dev set. We found that the best results were achieved when we capped our iterations at 50 (see Figure 5)

# Max Estimators	25	50	100	200
Dev Set RMSE	17.48	**17.07**	19.05	212.66
Dev Set MAPE	250.1	**250.6**	249.1	309.6

Fig. 5. Dev Set Errors by No. Max Iterations from Adaboost

Ensemble Methods: Random Forests

Another ensemble method we sought to experiment with is random forests. Random forests is an extension of the bagging algorithm, in which each weak learner subsamples a subset of the training data, and averages the predictions of its weak learners. Bagging has shown to increase bias only marginally while drastically reducing variance. Random forests is similar to bagging, but adds the constraint that each split in the decision tree can only be drawn from a random subset of the features provided. Doing so reduces the correlation the regression trees can potentially have when certain features are strong predictors, potentially yielding better results. Again, we decide to use a regression decision tree as the weak learner.

We found the best random forest algorithm by cross-validating across three parameters: the number of regression trees the algorithm will use, the percent of features to subsample from, and the depth of each of the regression trees. Iterating through combinations of all three parameters would take a lot of time. So we optimized for each parameter in the order above by holding the other two parameters constant. Figures 6-8 shows how we got to the optimal random forest, which has 80 trees, uses 70% of features, and has regression trees with depth of 13.

# Trees	20	40	60	**80**	100	120	150
Dev Set RMSE	17.82	17.20	17.38	**16.85**	17.43	17.31	16.87
Dev Set MAPE	235.7	231.8	231.9	**231.7**	232.9	232.9	231.1

Fig. 6. Random Forest Dev Set Errors by No. Trees (at 11 Tree Depth and 50% Features used

% Features Used	20%	30%	40%	50%	60%	**70%**	80%
Dev Set RMSE	60.76	36.77	21.15	**17.07**	16.46	**16.18**	16.31
Dev Set MAPE	251.9	239.1	233.5	**232.1**	230.7	**233.5**	234.6

Fig. 7. Random Forest Dev Set Errors by % Features Used (at 11 Tree Depth and 80 Trees

Ensemble Methods: Adaboost and Random Forests

Previously, we used a regression tree as the weak learner in the Adaboost algorithm. Here, we used our fine-tuned random forest algorithm as the weak learner, with the hope that a stronger weak learner can lead to even better results.

Again, we cross-validate the Adaboost algorithm across the max iteration parameter and found that 30 iterations yielded the best dev set MAPE. Figure 9 shows the cross-validation results.

Expanding Feature Space with Neural Networks

Since Jeong had yielded satisfactory results on Houston bus arrivals, we decided to experiment with neural networks to see if we can further reduce MAPE and RMSE. We experimented with neural networks with one hidden layer, with ReLU as the activation function, given that our predictive variable (time to arrival) is continuous and non-negative. As neural networks are prone to overfitting, we found the optimal number of hidden nodes and L2 regularization term with cross-validation. We decided to fix the learning rate schedule, where the learning rate decreases at a rate of $1/\sqrt{t}$. (Should more computing resources exist, we would have spent time fine tuning the learning rate). Resulting dev set errors can be found in Figure 10-11; the best fit neural network was found to have 10 nodes in the hidden layer and L2 regularization parameter of 10.

Depth of Tree	8	9	10	11	12	**13**	14	15
Dev Set RMSE	17.87	16.50	16.22	16.15	16.02	**16.00**	16.03	16.06
Dev Set MAPE	243.2	238.7	234.8	233.8	231.5	**230.6**	230.3	230.8

Fig. 8. Random Forest Dev Set Errors by Tree Depth (at 70% Features Used and 80 Trees

# Max Estimators	**30**	40	50	60	70
Dev Set RMSE	**16.44**	16.64	16.60	16.49	16.57
Dev Set MAPE	**240.5**	244.3	244.5	242.2	244.6

Fig. 9. Adaboost with Optimized Random Forest Dev Set Errors by Max Iterations

# Nodes in Hidden Layer	6	8	**10**	12	14
Dev Set RMSE	57.86	34.37	**31.19**	31.83	38.89
Dev Set MAPE	278.1	271.4	**272.3**	273.7	288.9

Fig. 10. Neural Network Dev Set Errors by Number of Hidden Nodes

L2-Regularization Parameter	0.01	0.1	1	**10**	100
Dev Set RMSE	35.75	33.27	31.19	**27.22**	28.92
Dev Set MAPE	264.0	273.9	272.3	**269.2**	273.2

Fig. 11. Neural Network Dev Set Errors by L2 Parameter

IV. Results

Table II shows the MAPE and RMSE of the various machine learning methods we have utilized and optimized, and compares it to the two baseline metrics. We can see that while Random Forest performs the best out of all algorithms (18% MAPE, 245 seconds in RMSE), it is still inferior to the MTA forecasts (14.5% MAPE, 203 seconds in RMSE).

TABLE II
Test Errors resulting from various algorithms

ML Method	Validation Parameters	Test MAPE	Test RMSE
Baseline 1 (Regress w/ Distance only)	None	28.99%	311.1
Baseline 2 (MTA Predictions)	None	14.48%	203.3
Regression (w/ Forward Selection)	No. Features = 2	27.99%	311.0
Lasso Regression	$\lambda = 10^3$	29.10%	311.1
Ridge Regression	$\lambda = 10^{10}$	28.48%	310.8
Adaboost.R2	Max. Estimators = 50	19.51%	275.7
Random Forest	80 Trees; Depth = 13; 70% Features	18.14%	245.4
Adaboost.R2 with Random Forest	Max. Estimators = 30	18.36%	250.2
Single Layer Neural Network	No. Nodes = 10 $\lambda = 10$	23.95%	268.8

V. Discussion of Results

As seen in the table above, random forests yielded the best results. While this is an interesting finding, a closer look at

feature importance implied from the machine learning model in Figure 12, shows that the two distance features (Distance in Meters and Distance over 300 Meters) are still far away the most important features utilized. This suggests that weather-related and population-related features did not play a major role in predicting bus time arrivals accurately, and is in line with what we found in the Linear Regression with Forward Selecting Features and Lasso Regression.

Feature	Importance
Distance > 300 Meters	49.2% +/- 33.5%
Distance (in Meters)	45.6% +/- 33.6%
Log Population Density	1.1% +/- 0.2%
Early Morning Hours	0.9% +/- 0.1%
Temperature	0.6% +/- 0.1%
Log Population	0.6% +/- 0.1%

Fig. 12. Features with Highest Importance Score from Random Forest

When graphing the MAPE and RMSE of the optimized random forest by distance to stop, we found that the plots look similar to the MAPE and RMSE of the MTA predictions (Figure 13). MAPE in general decreases as distance increases, while RMSE increases monotonically as distance to destination increases. This may suggest that the feature spaces I collected for my model and the MTA model are similar and that both sources are missing key features or information that can improve predictions in a drastic manner.

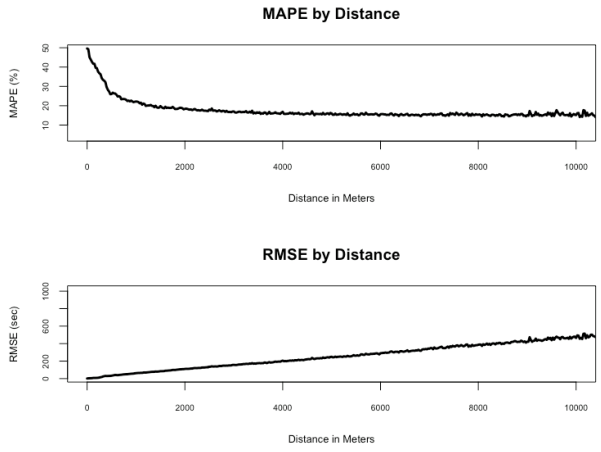

Fig. 13. MAPE and RMSE of Random Forest by Distance

One interesting finding is that ensemble methods performed significantly better than neural networks and linear regression. Since the weak learners of all the ensemble methods are decision trees, it is likely that dividing the feature space in a tree-like manner rather than utilizing a linear combination of features yielded better predictions.

One area of concern I had when collecting bus time arrival data was that the test set would not be a fair representation of the training and dev sets, leading to vast discrepancies between dev and test set error. This is because the training and dev sets captured bus arrivals on Wednesday and Thursday, while the test set captured bus arrivals on Friday, which may have drastically different traffic patterns. Fortunately, the final test set MAPE and RMSE for the optimized random forest was only 12% and 5% higher than the dev set MAPE and RMSE respectively.

Furthermore, when we re-trained the optimized random forest using the training and dev set to predict the actual test set, the training set MAPE was 18.05% and RMSE was 243.2, which is within one percent of the final test set MAPE and RMSE. This suggests that there is very little overfitting, and that a lot of the error we see are likely variances that our feature set may not have been able to capture.

This project is an example of how machine learning algorithms can only perform as well as the data we have at hand. After examining the compiled dataset, we had to discard over half of the dataset due to illogical results. We also only collected three days worth of bus arrival data, which may not be representative of bus movements on all days of the week. In addition, we were unable to incorporate features that truly capture traffic patterns of a specific bus route. It is likely the MTA possesses data that captures these trend, and as a result have estimated time of arrivals that have lower errors than what I have achieved.

In terms of further improvements I can make, I would like to experiment with support vector regression and various kernels to try to achieve lower MAPE and RMSE. However, given that this algorithm has quadratic time complexity as a function of the number of training samples, running SVRs on this particular dataset may be time consuming.

What I believe would generate larger improvements would be introducing more relevant features. Adding in bus routes as a dummy variable, I believe, can be helpful, although the 150 bus routes will dramatically increase the feature space of the dataset, leading to increased computational time. As eluded earlier, features describing traffic congestion of the associated bus route would help us differentiate the large variance that we have seen in ETAs. In addition, deriving a better way to capture bus movements and actual arrivals would also improve accuracy. As the MTA API only captures bus locations in 30-second increments, it is likely that this capturing mechanism introduces additional noise to the dataset.

REFERENCES

[1] MTA Bus Time API. Accessed Oct 18-20, 2017. http://bustime.mta.info/wiki/Developers/Index

[2] Dark Sky API. Accessed November 18, 2017. https://darksky.net/dev/docs

[3] Jeong, Ran Hee (2004). The prediction of bus arrival time using Automatic Vehicle Location Systems data. Doctoral dissertation, Texas A&M University. Texas A&M University.

[4] New York City Population by Census Tracts. https://catalog.data.gov/dataset/new-york-city-population-by-census-tracts-af944

[5] 2017 U.S. Gazetteer Files. https://www.census.gov/geo/maps-data/data/gazetteer2017.html

[6] Drucker, Harris. Improving regressors using boosting techniques. In Proceedings of the Fourteenth International Conference on Machine Learning, pp. 107-115, 1997.