NEW YORK CITY TAXI TRIP DURATION

I. DEFINITION

PROJECT OVERVIEW

The goal of this project is to predict trip durations for taxi rides. Predicting the duration of taxi rides is useful, because it can improve the efficiency of electronic taxi dispatching systems. VHF-radio dispatch systems have widely been replaced by electronic dispatch systems. Each taxi vehicle has a data terminal that records GPS and taximeter information as the trip progresses. A taxi driver receives ride requests from people hailing taxis on the street, as well as pickup requests from dispatchers. In addition to the change in dispatch system, the method in which dispatch messages are made has changed. Unicast-based messages, where a dispatcher communicates directly with a driver, rather than broadcast-based messages, where a dispatcher announces a pick up request to multiple drivers, are now the norm.

Since dispatchers are now communicating one-on-one with drivers, each dispatcher needs to decide which driver he/she should send to a pick up location. However, this is not straightforward because dispatchers are not informed of the drop off locations for a request, and the taxi driver does not input this information into the vehicle's data system. Thus, if the duration of a taxi's ride could be predicted, a dispatcher would be able to make better assignments of a pickup request to a taxi. This problem has been previously studied in this Kaggle competition, where the destinations of taxi trips are predicted based on their initial trajectories, and in this Kaggle competition, where the total travel times of taxi trips are predicted based on their initial partial trajectories.

PROBLEM STATEMENT

In this report, we look at a Kaggle competition with data from the NYC Taxi and Limousine Commission, which asks competitors to predict the total ride time (*trip_duration)* of taxi trips in New York City. The data provided by Kaggle is structured data provided as a CSV file. The data in the CSV file includes multiple formats: timestamps, text, and numerical data. This is a regression analysis, since the output, total ride time, is numerical. I will use several machine learning methods for the prediction task, which are linear regression, k-nearest neighbors regression, random forests, and XGBoost. The models will be evaluated using the root mean squared logarithmic error.

EVALUATION METRIC

The evaluation metric is the root mean squared logarithmic error:

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(\log(p_i + 1) - \log(a_i + 1))^2},$$

where n is the number of observations in the dataset, p_i is the predicted value for trip duration, and a_i is the actual value for trip duration. There are outliers in the trip duration, and the RMSLE is less affected by outliers.

II. Analysis

Datasets

The main dataset for this project is data from the NYC Taxi and Limousine Commission (TLC), that was published as 2016 NYC Yellow Cab trip record data on Google Cloud Platform. For the Kaggle competition, the data was cleaned and sampled, so that the training dataset contains 1,458,644 trip records, and the testing dataset contains 625,134 trip records. Each trip record contains the following attributes (descriptions taken from the Kaggle competition):

- *id* - a unique identifier for each trip
- *vendor_id* - a code indicating the provider associated with the trip record
- *pickup_datetime* - date and time when the meter was engaged
- *dropoff_datetime* - date and time when the meter was disengaged
- *passenger_count* - the number of passengers in the vehicle (driver entered value)
- *pickup_longitude* - the longitude where the meter was engaged
- *pickup_latitude* - the latitude where the meter was engaged
- *dropoff_longitude* - the longitude where the meter was disengaged
- *dropoff_latitude* - the latitude where the meter was disengaged
- *store_and_fwd_flag* - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip
- *trip_duration* - duration of the trip in seconds

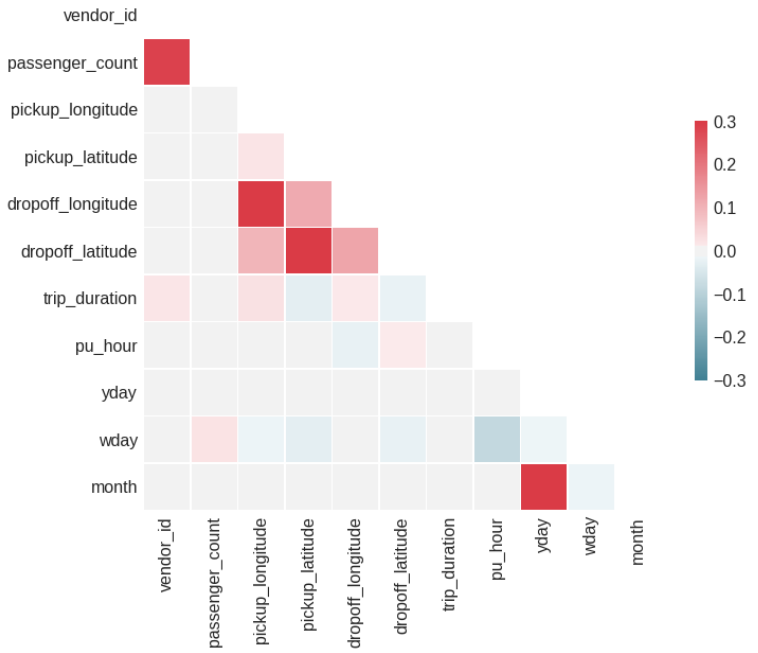
I will augment the provided dataset with data for the weather in New York City in 2016. This data was obtained by the National Weather Service, who recorded the weather at a station in Central Park, New York City for the first half of 2016. The features of the dataset are the daily low, high, and average temperatures, precipitation, snowfall, and accumulated snow depth.

Additionally, I used a dataset that provides the fastest routes for each trip, based on maps from the Open Source Routing Machine, OSRM. For each trip in the Kaggle NYC taxi dataset, this dataset provides the total distance between the trip start and end calculated from the fastest possible route (*total_distance*), the trip duration for this route

(*total_travel_time*) and the steps of the route (i.e. turns taken). I will only use the *total_distance* and *total_travel_time* features. The reasons for using this dataset and the weather dataset are discussed in the Exploratory Analysis section.

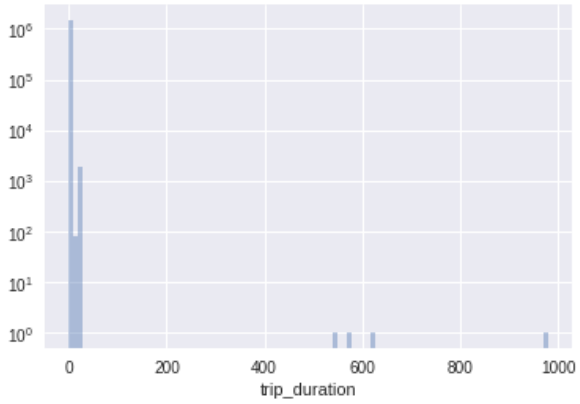
EXPLORATORY ANALYSIS

As a first step for exploratory analysis, I made a correlation plot for features in the Kaggle taxi dataset. Since the *datetime* features are not rational numbers, I extract the individual components of the *datetime*: hour (*pu_hour*), day of year (*yday*), day of week (*wday*), and month (*month*). There is no need to extract the year, since all of the data is from 2016. The correlation and trained models will use these extracted features of *pickup_datetime*.

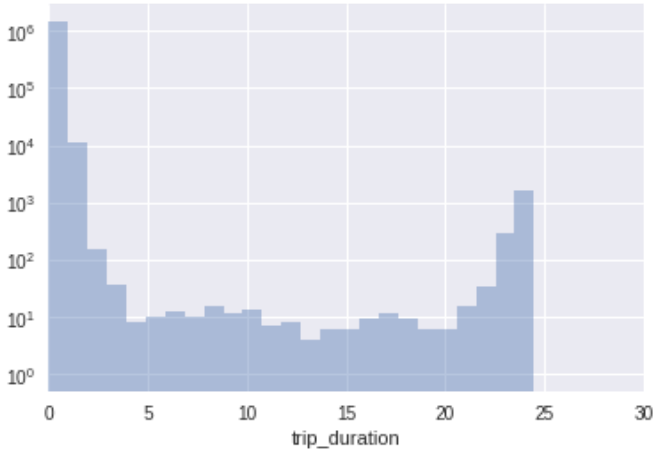


The correlation plot shows that the strongest correlations are between *passenger_count* and *vendor_id*, *dropoff_longitude* and *pickup_longitude*, *dropoff_latitude* and *pickup_latitude*, and *yday* and *month*. The only surprising correlation here is the correlation between *passenger_count* and *vendor_id*. This is surprising, as one would not expect a taxi company to cater to higher or lower numbers of customers, though perhaps one of the companies only has the capability to transport greater *passenger_counts* (i.e. they have SUVs). Most notably, none of the features in the dataset have a strong correlation with the feature I want to predict, *trip_duration*, which means that new features will need to be added.

The distribution of *trip_duration* in the training data set is shown below.



The plot has a log y-scale and *trip_duration* is measured in hours. There are some trips that last almost as long as 1000 hours! I will assume that the taxi driver forgot to turn off the meter for these very long trips. Restricting the range of *trip_duration* from 0 to 30 hours, results in the following distribution that has a log-scale y-axis.



There is a sharp cut-off at 24 hours (perhaps the taxi meter resets every day at midnight), and *trip_duration*s less than 2 hours are about 10^5 times more frequent than *trip_duration*s longer than 2 hours. *Trip_duration*s lasting multiple hours could be possible if someone took a taxi sightseeing around New York, so I will restrict *trip_duration* to be less than 22 hours. My reasoning is that there are ~24 hour trips because the taxi driver forgot to turn off the meter, so there is likely to be more of these trips, and *trip_duration* begins to spike in the plot around 22 hours.

New Features

Distance

The geodesic distance-the literal shortest route between two points-is calculated from the coordinates of the pickup and dropoff locations. This corresponds to the minimum travel distance possible because taxis (usually) do not take the straight-line distance between two points. I use the haversine formula to compute the great circle distance between the pickup and dropoff locations. The haversine formula is:

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

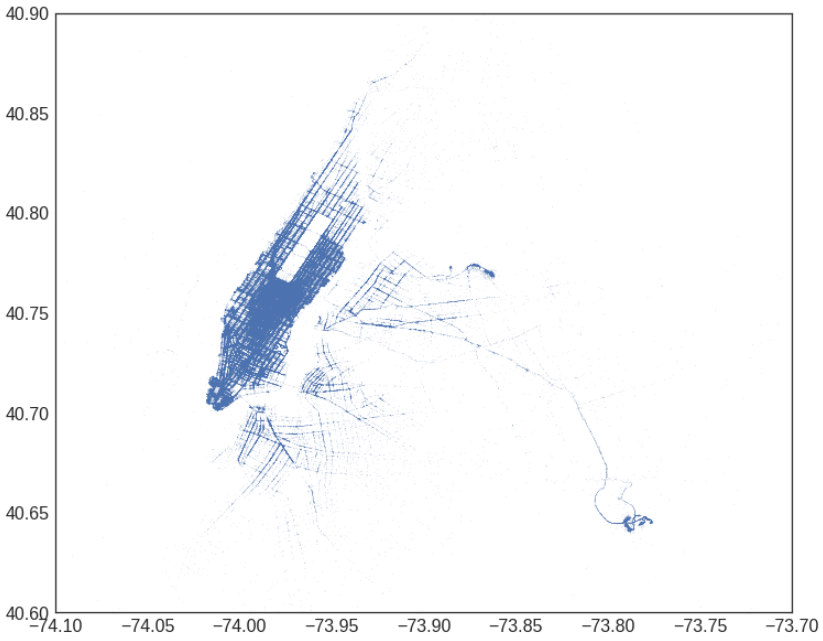
$$c = 2 \cdot \operatorname{atan}^2(\sqrt{a})$$

$$d = R \cdot c,$$

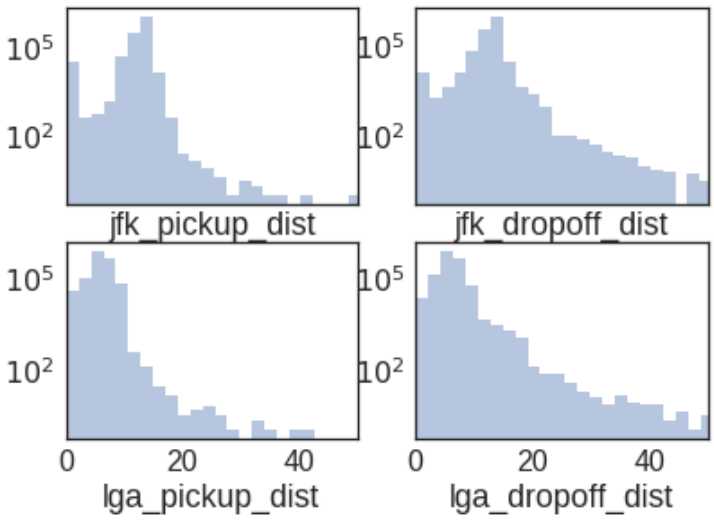
where φ is latitude, λ is longitude, and R is the Earth's radius in miles (3959 miles).

AIRPORTS: JFK AND LGA

Plotting the coordinates of the pickup locations reveals the outline of Manhattan in New York City, with fewer pickups in the outer boroughs:



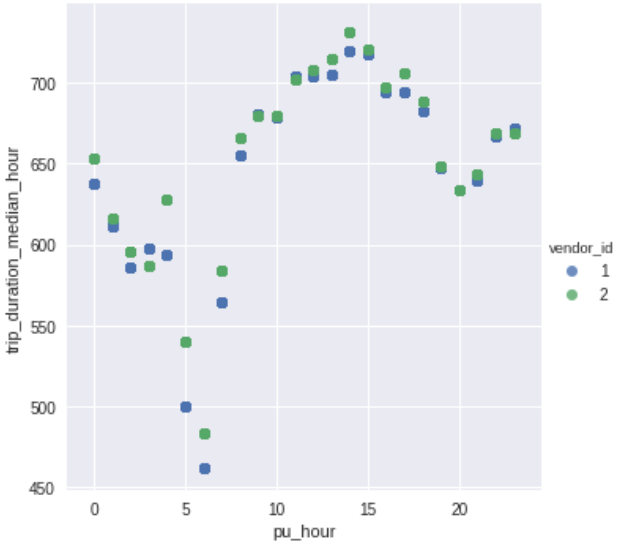
There are two clusters of pickups outside of Manhattan, and these happen to be the locations of two of the major airports in the New York area: John F. Kennedy International Airport (JFK) and LaGuardia Airport (LGA). For each trip in the dataset, I calculate the distance of each airport from the pickup location and the dropoff location. The coordinates of JFK International Airport are 40.639722° N, -73.778889° W, and the coordinates of LaGuardia Airport are 40.77725° N, -73.872611° W. The distributions of the distances are:



where I have used a log-scale to see the shape of the features better. In the JFK and LGA pickup and dropoff distance distributions, I see that there are two peaks in the data. There is one peak around 20 miles, and another peak around 2 miles. A pickup distance less than or equal to 2 miles from the airport seems reasonable for a trip that originates/terminates at the airport. Since there are clusters of trips originating/ending at these two airports, I will add two new Boolean features: *jfk* and *lga*, with the idea of them predicting longer trips. *jfk* and *lga* are true if a trips originates or ends less than 2 miles from the JFK and LGA airports. While I can't see a noticeable peak around this distance in the LGA distributions, I will still use 2 miles as the upper limit for an airport trip to/from LGA.

WORKDAY

I calculate the median *trip_duration* by hour for each *vendor_id* and plot it versus the pickup hour in the following plot:



The plot shows that the median *trip_duration* is longer for trips with pickup hours during hours of about 8 AM to 6 PM, approximately the hours of a typical work day. Therefore, I will add a new feature *workday*, which will be true if a trip's pickup hour occurs between 8 AM and 6 PM, and false otherwise.

ALGORITHMS AND TECHNIQUES

LINEAR REGRESSION

The first model that I will use is linear regression, as it is a simple model. Linear regression models the dependent variable, Y, in terms of the explanatory/independent variables, x_i, as

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n,$$

where β_0 is the intercept. In sklearn, this is accomplished by using `sklearn.linear_model.LinearRegression`, which fits the regression line with the method of least-squares. Least-squares linear regression finds the best-fit line for the data by minimizing the sum of squares of vertical deviations. The vertical deviations measure how far each data point is from the line. The positive and negative deviations have equal contributions because the deviations are first squared. Linear regression is sensitive to outliers, which are points which lie far from the line. Outliers can have a significant impact on the slope if they are *influential observations*, which are far away from the majority of the data points in the horizontal direction.

K-NEAREST NEIGHBORS REGRESSION

K-nearest neighbors is also a simple algorithm like linear regression. This algorithm stores all training data and then predicts the numerical target based on the k-nearest neighbors as calculated from some distance function. Examples of distance functions are the Manhattan distance, $\sum_{i=1}^{k} |x_i - y_i|$ and the Euclidean distance, $\sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$. In sklearn, this is accomplished by using `sklearn.neighbors.KNeighborsRegressor`, with the default parameters of a Euclidean distance function, and uniform weights (all data points are weighted equally).

RANDOM FORESTS

A decision tree is a supervised learning method that is non-parametric. The target variable is predicted by the model learning decision rules from the data features. Decision trees are simple to interpret, but they can overfit and be unstable as a small variation in the input data could cause a different tree to be generated. A solution to this problem is to use

an ensemble of trees, such as in a random forest. A random forest fits many decision tree classifiers on sub-samples of the dataset, and over-fitting is controlled with averaging.

XGBoost

XGBoost stands for eXtreme Gradient Boosting, and implements the gradient boosting decision tree algorithm. Boosting is an ensemble method in which the errors made by existing models are corrected by new models. New models keep being added until the improvement stops. These new models predict the errors of the prior models, and are put together for the final prediction. The loss is minimized when new models are added by using gradient descent, hence the name gradient boosting.

Benchmark Model

As the benchmark model, I will use the dataset for the fastest routes for each trip from the OSRM, to simply predict that each trip takes the fastest possible time. The benchmark model for the testing data set achieves a RMSLE of 0.990.

III. Methodology

Data Preprocessing

From the Exploratory Analysis section, we saw that there is a sharp cut-off in *trip_duration* at 24 hours, and since *trip_duration* begins to spike around 22 hours, I decided to remove data points with *trip_duration*s longer than 22 hours. Additionally, we saw that none of the features in the dataset have a strong correlation with the target, *trip_duration*, which means that new features need to be added. Therefore, I added the following features:

- *distance*: the geodesic distance between the pickup and dropoff locations in miles.
- *jfk* and *lga*: Boolean variables which are true if a trips originates or ends less than 2 miles from the JFK and LGA airports.
- Weather data as described in the Datasets section. The weather data includes the features *snow fall, precipitation,* and *snow depth*, which describe respectively the snow fall in inches, the precipitation in inches, and the amount of snow on the ground for each day. Some of these features have a 'T' to indicate trace amounts, but this cannot be used since I am doing a regression analysis. I have replaced all values of 'T' for these features with the value of 0.05, as this value is smaller than the smallest numerical value across all the features.

- *total_travel_time:* The time required for the fastest route possible between the pickup and dropoff locations, as described in the Datasets section.
- *total_distance:* the distance of the fastest route possible between the pickup and dropoff locations, as described in the Datasets section.

See the Exploratory Analysis section for a discussion about these new features. I have removed all datapoints with a *trip_duration* less than 1 minute, as a 1 minute taxi trip seems implausible. I have also removed all data points with a *trip_duration* greater than 1 minute, but a *distance* less than 0.05 miles, as taxi trips where a taxi did not move seem implausible. Additionally, I remove all data points that have a *passenger_count* of zero, as these points are probably mistakes since a taxi needs to have a passenger. Finally, I remove all data points with a *distance* (in miles) divided by *trip_duration* (in hours) greater than 60 miles per hour. I remove these points because data that have an average speed greater than 60 miles per hour seem suspicious, since New York is a large metropolitan city.

In summary, the models are trained on the following features: *vendor_id, passenger_count, pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude, pu_hour, wday, month, workday, precipitation, snowfall, snowdepth, total_distance, total_travel_time, jfk,* and *lga*.

IMPLEMENTATION

This project uses the following programming language and libraries:

- Python 3
- Scikit-learn: Python's open source machine learning library
- XGBoost: Python package for XGBoost model,

Where the particular machine learning algorithms are implemented as follows:

- Linear Regression: `sklearn.linear_model.LinearRegression` with the default arguments.
- K-nearest neighbors regression: `sklearn.neighbors.KNeighborsRegressor` with the default arguments other than *n_neighbors*=10
- Random forest: `sklearn.ensemble.RandomForestRegressor` with the default arguments
- XGBoost: `xgboost.XGBRegressor` with the following arguments *n_estimators*=100, *learning_rate*=0.08, *gamma*=0, *subsample*=0.75, *colsample_bytree*=1, and *max_depth*=7

The implementation is organized by initializing each of the models described above, and performing `cross_val_score` with the cross-validation splitting strategy `ShuffleSplit` with four splits on a test size of 10%. Then with the fitted model, I make the predictions on the test set, and submit these predictions to Kaggle, who provides me with the RMSLE score. I did not find any complications with the coding process.

Refinement

To help prevent the model from overfitting, I will use cross-validation. Cross-validation is learning the model parameters on a train set of data, and then testing the model on a different test set of data. Cross-validation is implemented in this project by using `sklearn.model_selection.ShuffleSplit`, with the arguments *n_splits*=4, *test_size*=0.1, *random_state*=0. `ShuffleSplit` gives *n_splits* independent train/test splits of the dataset. This is accomplished by shuffling the samples, and then splitting the samples into train and test sets in the proportion of *test_size*. `ShuffleSplit` is implementing *k*-fold cross-validation.

IV. Results

Model Evaluation & Validation

Kaggle provides a training set of data to competitors who can test their trained models on a hidden testing set. Kaggle will then provide the RMSLE. Since the RMSLE is undefined for negative values, I have replaced every *trip_duration* that is predicted to be negative with zero. Specifically, the number of negative *trip_duration* values are:

- Linear Regression:
  - cross-validation with 4 splits of training data: 14, 9, 17, 12 (out of 1,458,644 predictions)
  - testing data: 490 (out of 625,134 predictions)
- KNN:
  - cross-validation with 4 splits of training data: 0, 0, 0, 0
  - testing data: 0
- Random forest:
  - cross-validation with 4 splits of training data: 0, 0, 0, 0
  - testing data: 0
- XGBoost:
  - cross-validation with 4 splits of training data: 1, 0, 0, 0
  - testing data: 1

The results for the four machine learning models and the benchmark are as follows:

- Benchmark: The RMSLE on the test dataset is 0.990.
- Linear Regression: the four cross-validation scores are: 0.439, 0.440, 0.438, and 0.440 (average=0.440). The RMSLE on the test dataset is 0.535.
- K-Nearest Neighbors Regression: the four cross-validation scores are: 0.420, 0.419, 0.419, and 0.419 (average=0.420). The RMSLE on the test dataset is 0.505.
- Random Forest: the four cross-validation scores are: 0.350, 0.351, 0.349, and 0.350 (average=0.351). The RMSLE on the test dataset is 0.473.
- XGBoost: the four cross-validation scores are 0.343, 0.343, 0.341, 0.342 (average=0.342). The RMSLE on the test dataset is 0.463.

For all models, the RMSLE on the test set is higher than the average cross-validation RMSLE on the training set, indicating that there is some overfitting occurring.

Since XGBoost has the best performance, I have tried to improve the performance with `GridSearchCV` with four cross-validation splits and RMSLE scoring. First, I searched over the parameters:

- `max_depth`: 3, 5, 7
- `min_child_weight`: 1, 3, 5,

while keeping `learning_rate`=0.08, `n_estimators`=100, `seed`=0, `subsample`=0.75, `colsample_bytree`=1. The model with the lowest RMSLE had an average cross-validated RMSLE of 0.34564, with `max_depth`=7 and `min_child_weight`=1. (This is shown in xgb.ipynb.) With these values of `max_depth` and `min_child_weight`, I used `GridSearchCV` again, but this time I searched over the parameters:

- `learning_rate`: 0.1, 0.01
- `subsample`: 0.7, 0.8. 0.9

The model with the lowest RMSLE had an average cross-validated RMSLE of `0.34296`, with `learning_rate=0.1` and `subsample=0.8`. (This is shown in xgb2.ipynb.) With these parameters, (`learning_rate`=0.1, `subsample`=0.8, `max_depth`=7 and `min_child_weight`=1), the RMSLE on the test set is 0.45743. (This is shown in xgb3.ipynb.) This model doesn't predict any negative values of *trip_duration* on the train set, but has 21 negative predicted values on the test set.

JUSTIFICATION

The RMSLEs of all machine learning models on the test dataset are better than the RMSLE of the benchmark model. The model with the best RMSLE is XGBoost, which halves the RMSLE of the benchmark model. The cross-validation scores of each of the models are relatively stable, indicating that each of the trained models is robust.

V. Conclusion

Visualization

In the attached visualization, I have displayed one of the trained XGBoost models. The top node is *total_travel_time* < 603.05. If not, the next node is *total_travel_time* < 1178.55. If the top node is true, the next node is *total_distance* < 2688.85. Intuitively, it makes sense that the two top levels are *total_travel_time* and *total_distance*, since one would expect that at first order the travel time is determined by the distance of the fastest possible route, and the time of that route.

Reflection

To predict the travel time from pick up to dropoff of taxis in New York City, I have first performed an exploratory analysis. Using the features in the provided Kaggle competition dataset, I made a correlation plot, and saw that there was no strong correlation between *trip_duration*, the target variable, and any other features. This shows that additional features would need to be used to predict the *trip_duration*. For me, the most interesting and difficult parts of this project were thinking of external data and additional features to use. To add new features, I used two external datasets: the weather in New York during which the data for the Kaggle competition was collected, a dataset that provides the fastest routes for each trip. I also derived some features from the original dataset: the distance from the pickup location to the dropoff location, whether or not the pickup or dropoff locations where close to the JFK or LGA airports, and whether or not the pickup occurred during workday hours.

With these new features, I trained four different machine learning models on the Kaggle provided train dataset: linear regression, k-nearest neighbors, random forest, and XGBoost. The trained models are tested on a hidden test dataset, and evaluated by the root mean squared logarithmic error (RMSLE). I compare the RMSLE of these models to the RMSLE of the benchmark model evaluated on the test dataset. For the benchmark model, I predict that each trip takes the fastest possible time, as given in the external dataset from OSRM. The four machine learning models reduce the RMSLE of the benchmark model by about half. The model with the best RMSLE on the test set is XGBoost. I have visualized an XGBoost decision tree in the Visualization section. The nodes on the top two levels are either *total_travel_time* or *total_distance*, which fits my expectations for the problem. Since

the model improves upon the benchmark model, which just predicts that the travel time is the fastest time possible, I would say that this model should be used to predict taxi travel times in NYC. (The best performing model, XGBoost, predicts one out of 625,134 predictions on the training set has a negative *trip_duration*. Ideally, I would have the model say that it doesn't have a valid prediction, however, the RMSLE is undefined for values less than zero, so I need to predict a non-negative value for Kaggle to score my solution.)

Improvement

I think that the RMSLE on the test dataset could be improved by adding additional features to the dataset. For instance, the OSRM dataset of fastest travel times also includes information on the steps taken along the route. Since cars in the United States travel on the right hand side of the road, I would think that a contribution to the travel time from pickup to dropoff would be how many left turns a taxi takes along the route. I suspect that this would be important to the total travel time, since a taxi would need to wait for oncoming traffic to pass if it wanted to make a left-hand turn.

VI. References

1. http://www.ecmlpkdd2015.org/discovery-challenge/learning-taxi-gps-traces
2. https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i
3. https://www.kaggle.com/c/pkdd-15-taxi-trip-time-prediction-ii
4. https://www.kaggle.com/c/nyc-taxi-trip-duration
5. https://en.wikipedia.org/wiki/Earth_radius
6. http://www.movable-type.co.uk/scripts/latlong.html
7. https://tools.wmflabs.org/geohack/geohack.php?pagename=John_F._Kennedy_International_Airport&params=40_38_23_N_073_46_44_W_region:US-NY_type:airport
8. https://tools.wmflabs.org/geohack/geohack.php?pagename=LaGuardia_Airport&params=40_46_38.1_N_73_52_21.4_W_region:US-NY_type:airport