New York City Taxi Trip Duration

Executive Summary

In the Kaggle competition New York City Taxi Trip Duration, the challenge is to predict the duration of each taxi trips in the test dataset using the model built on features related to taxi trip. The dataset of this competition comes from the 2016 New York City Yellow Cab trip record data published by the New York City Taxi and Limousine Commission (TLC).

The train dataset contains almost 1.5 million trip records (1458644 to be exact) and has 10 different attributes. The attributes can be divided into three categories. First, basic description of every trip, including who is the provider of that trip, the number of passengers in the taxi, whether the taxi had a connection to the server or not, and duration of every trip, measured in seconds. Second, date and time of pickup and dropoff. Third, location information, including pickup longitude and latitude, dropoff longitude and latitude.

The test dataset is smaller, contains 625134 trip records. The test dataset is removed two attributes from the train dataset, date and time of drop-off, and duration of the trip, for us to use the trained data to predict the duration, because the duration of every trip is calculated by dropoff time minus pickup time.

The method of accuracy evaluation is Root Mean Squared Logarithmic Error (RMSLE), because we want our prediction to be as close as the actual duration as possible, so for the final result, the smaller the score is, the better its performance.

After benchmark other solutions, we find that for feature approach, the common ways include generate new features and remove outliers based on the distribution of variables. Because the dependent variable is trip duration, from the given latitude and longitude information, we can generate new features such as distance and direction, from the given date and time of pickup and dropoff, we can calculate trip duration and hence the corresponding speed. And then use correlations and feature importances to decide those features that can help improve the performance of our model. After further analysis, the results show that new created features “distance” and “direction” are very helpful in improving the accuracy of our predicting models. On the other hand, the feature “store_and_fwd_flag” can be ignored, as it did not improve our models.

For our problem, the goal is to predict trip duration, so we should build a regression model. Linear regression, random forest, gradient boosting, XGBoost and LightGBM are all advanced algorithms solving regression problems. In the modeling section of the paper, I compared the performance of three different models: random forest, gradient boosting and LightGBM. The results show that overall, LightGBM performs best in predicting the trip duration in testing dataset.

Benchmarking of Other Solutions

Kernel Name	Feature Approach	Model Approach	Performance (Score)
①EDaongam	1.Delete outliers and duplicated values	Random Forest	0.40302

	2.Drop the feature indicates whether the trip was held in vehicle memory 3.Calculate distances from pickup to dropoff 4.Convert string to datetime for pickup and dropoff date time		
②NYTaxis_ComeMS: Machine Learning	1.Calculate distances between pickup and dropoff 2.Create a new feature called 'zone' indicates where the pickups are 3.Filter data, remove small distances and fast trips 4.Convert datetime	XGBoost	0.39939
③ML Workflow	1.Remove outliers and duplicated values 2.Make a log-transformation of trip duration's data 3.One-hot encoding binary categorical features 4.Create date features and direction 5.Calculate distances between pickup and dropoff, remove distance outliers 6.Create speed features, remove speed outliers 7.Correlations and dimensionality reductions	LightGBM	0.37697

The three solutions above use three different model, as we mention before, lower score means better performance, and we can see that solution three which uses LightGBM model is the most successful one. Apart from model approach, the three solutions also approach differently in feature engineering part.

- Solution One

1. Visualize all the numerical features with histograms and box plots.

2. Based on the observation, only keep the values of trip duration between 0 and 5000.

3. Draw the scatter plots of pickup latitude and longitude, dropoff latitude and longitude, based on the observation, keep the values of pickup latitude between 40 and 41, pickup longitude between -75 and -73, dropoff latitude between 40 and 41.5, dropoff longitude between -75 and -72.5.

4. Only keep the values of passengers between 0 and 6.

5. Checking for missing values and duplicated values.

6. Drop the feature "store_and_fwd_flag".

7. Generate features "distance", "month", "week_day", "day_month" and "pickup_minutes". Distance is calculated using Haversine fuction.

- Solution Two

  1. Generate two new features, one is “distance” calculated distance between pickup and dropoff, using Manhattan distance formula. The other is “zone”, indicating where the pickups are.

  2. Generate features “pickup_month”, “pickup_hour”, “pickup_weekdays” and “pickup_weekend”.

  3. Remove trip distance less than 0.01km, trip duration less than 60 seconds and longer than 2 hours. Only keep those with average speed between 1km/h and 200km/h.

- Solution Three

  1. Checking for missing values and duplicated values.

  2. Visualize all the numerical features with box plots. Only keep trip duration less than 5900 seconds, and trips with passengers.

  3. Draw the scatter plots of pickup latitude and longitude, dropoff latitude and longitude, based on the observation, keep the values of pickup latitude smaller than 50, pickup longitude larger than -100, dropoff latitude smaller than 50, dropoff longitude between -80 and -70.

  4. Make a log transformation of trip duration values.

  5. Generate new feature “distance” using Haversine fuction. Create “month”, “week”, “weekday”, “hour”, “minute_oftheday” and “direction”.

  6. Analyze the relevance of variables using correlations and dimensionality reductions.

First, solution one and solution three both use visualization to remove outliers, they both focus on trip duration, longitude and latitude of pickup and dropoff. Solution one also takes number of passengers into account and remove outliers of more than 6 passengers. Based on the results of visualization, solution one only retains the most centric while solution three treat values with wider tolerance and try to delete as little values as possible.

Second, only solution one drops the feature indicates whether the trip was held in vehicle memory.

Third, solution two and solution three both generate similar features indicating where the pickup took place, as well as the speed associated with each trip. And based on the results to remove outliers.

Overall, all three solutions deal with outliers based on their own criteria. We can see that for the two solutions with scores lower than 40, new features are created and used to deal with data. And choose different criteria to deal with features may affect the performance of the model. Of course, in the modeling part, choosing different parameter will also affect the score of models, but because the three solution chose different algorithms, we are unable to compare that. It is worth noting that only solution three analyze the relevance of independent variables, and it is very important in feature selection, so that may be part of the reason why this solution yields the best score.

Data Description and Initial Processing

- Basic description of all data

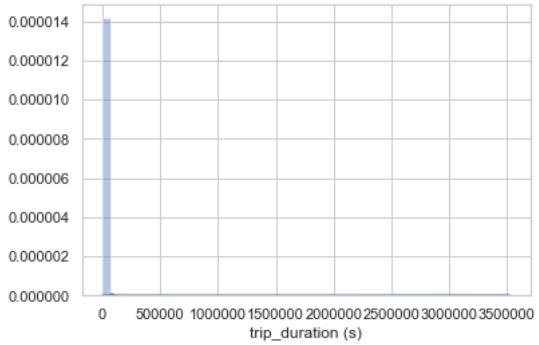
We can see that “vendor_id”, “passenger_count”, “pickup_latitude”, “pickup_longitude”, “dropoff_longitude”, “dropoff_latitude” and “trip_duration” are numeric data types, which means we can perform calculation directly. Except for the feature “vendor_id”, although the data type is integer, “vendor_id” is a categorical feature, there are only two values: 1 and 2, indicating that there are two different providers recorded in the dataset.

field	Data Type
id	object
vendor_id	int64
pickup_datetime	object
dropoff_datetime	object
passenger_count	int64
pickup_longitude	float64
pickup_latitude	float64
dropoff_longitude	float64
dropoff_latitude	float64
store_and_fwd_flag	object
trip_duration	int64

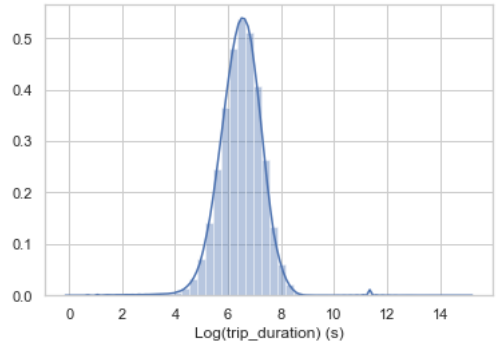
Then, we check if our dataset contains null or duplicated values, the corresponding result shows that there are not null or duplicated values in both the train and test dataset.

- Trip duration visualization

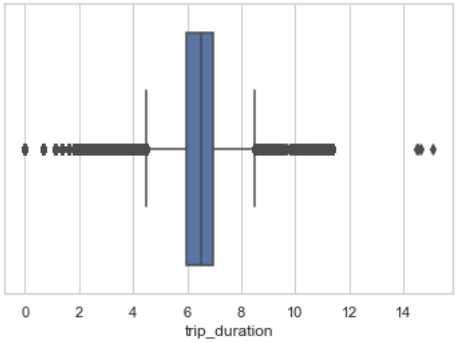
Now, we visualize numerical features for further data exploration. Because the goal is to predict trip duration, we first visualize the dependent variable trip duration with histogram.



The dataset records trip duration in seconds, and some of the very long trip duration (lager than 100 hours) are making all another trip duration invisible in the graph, as we can see that the distribution is skewed right, so we make a log-transformation of trip duration’s data. It is clear that most of the values are between 4 and 8.5, which means trip duration are between 55 seconds to 5000 seconds.

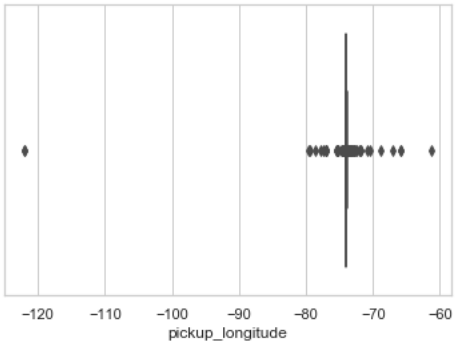


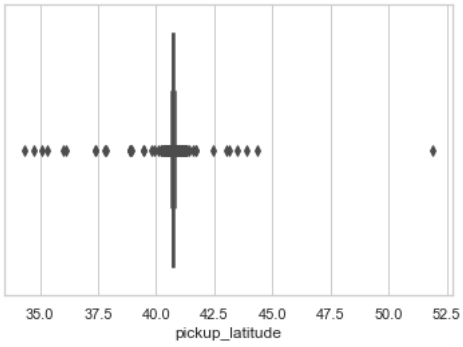
We can also draw the box plot to observe the outliers.

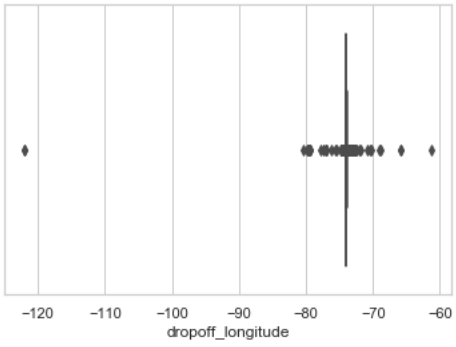


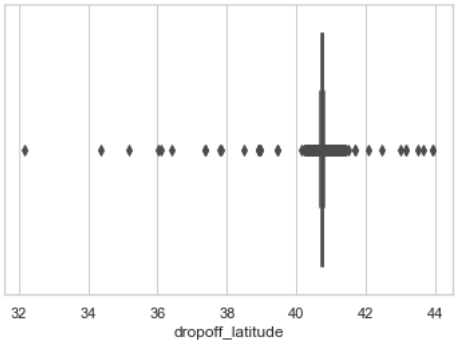
- Latitude and longitude visualization

We combine longitude and latitude to visualize data of pickup and dropoff location. We first visualize each of the four features with box plot, then combining them using scatter plot.



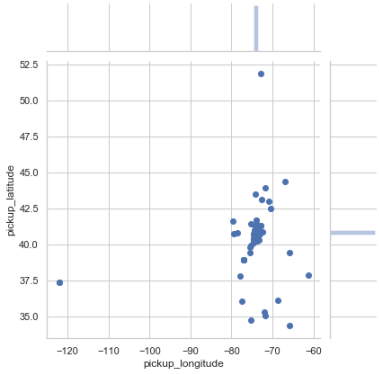


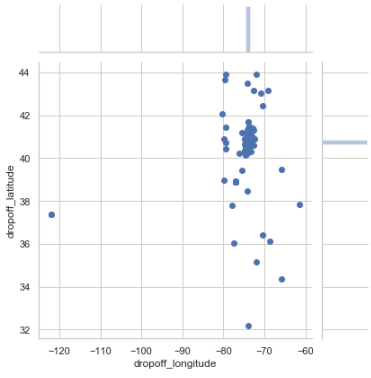




It is clear that most of the pickup latitude is between 40 to 42.5 and longitude is between -80 to -70. We can get similar result from dropoff's data as most of the dropoff happened between

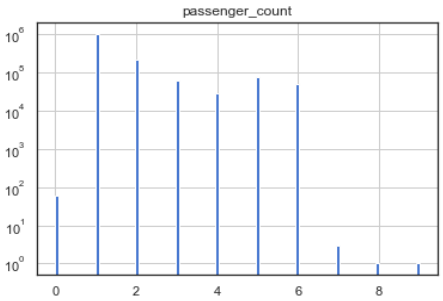
latitude 40 to 42 and longitude -80 to -70. It is not very surprising since most of the taxi tend to gather in certain area.





- Passengers count visualization

We visualize the number of passengers associated with each trip. It is clear that the number of passengers range from 0 to 6 for most of the trips.



For the outliers observed in the analysis above, we can simply remove them, because there is no proper explanation and these values contribute little to out model building. For example, taxi trip duration less than one minute makes no sense, and because the dataset records taxi trip in New York city, longitude and latitude locate too far can also be ignored. Another reason is that our dataset is very large, after removing outliers, the training set still contains 1446999 records.

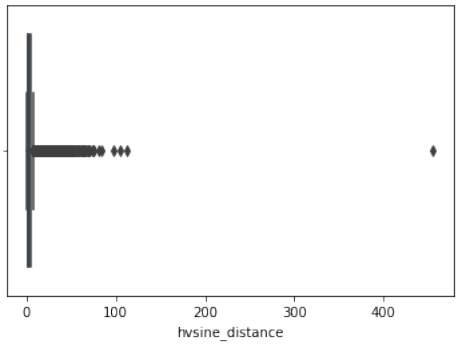
- Deal with categorical data

After getting some basic information of the numerical features, we move to the textual data. Clearly, we need to drop the field “dropoff_datetime” as it contains the information of actual taxi duration, and our test dataset does not include this feature. We can ignore the field “id”, it is unique to each trip, so it is also non-informative. For the categorical features “vendor_id” and “store_and_fwd_flag”, we create dummy variables. For the features “pickup_datetime”, we convert data type from string to datetime and create separate time features. We first analyze its interval, the result shows that the record taxi trip time of the train and test dataset range from January 2016 to June 2016, so we can ignore Year. The new features created include: “month”, “week_of_year”, “weekday”, “day_of_year”, “minute_of_day”.

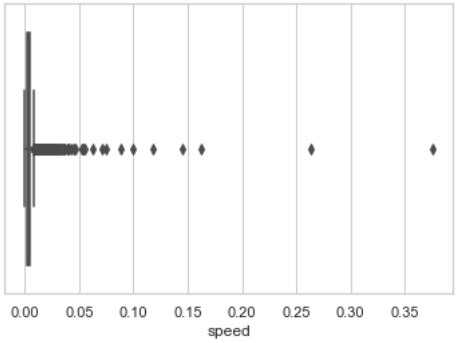
- Generate new features

Based on the given information, we can generate new features “direction”, “distance” and “speed”, as these features might help analyze. From the results of the pair plots of numerical features, we find that there are some relationships between pickup and dropoff latitude and latitude, so we create the new feature “direction” indicating where every trip is heading.

One of the most common methods of calculating distance is Haversine function, Haversine function calculates the shortest distance between two points on a spherical object. We can see that most of the distance is less than 200 km, so we remove outliers.



Then we calculate the speed using distance divided by trip duration. The unit of our speed is km/s, given the box plot result, most of the speed is less than 0.05 km/s (180km/h), so we can remove those with speed greater than 0.05 km/s.



Analysis of Relevance of Independent Variables

The independent variables in our dataset now include:

-vendor_id

-store_and_fwd_flag

-passenger_count

-pickup_longitude

-pickup_latitude

-dropoff_longitude

-dropoff_latitude

-distance

-direction

-month

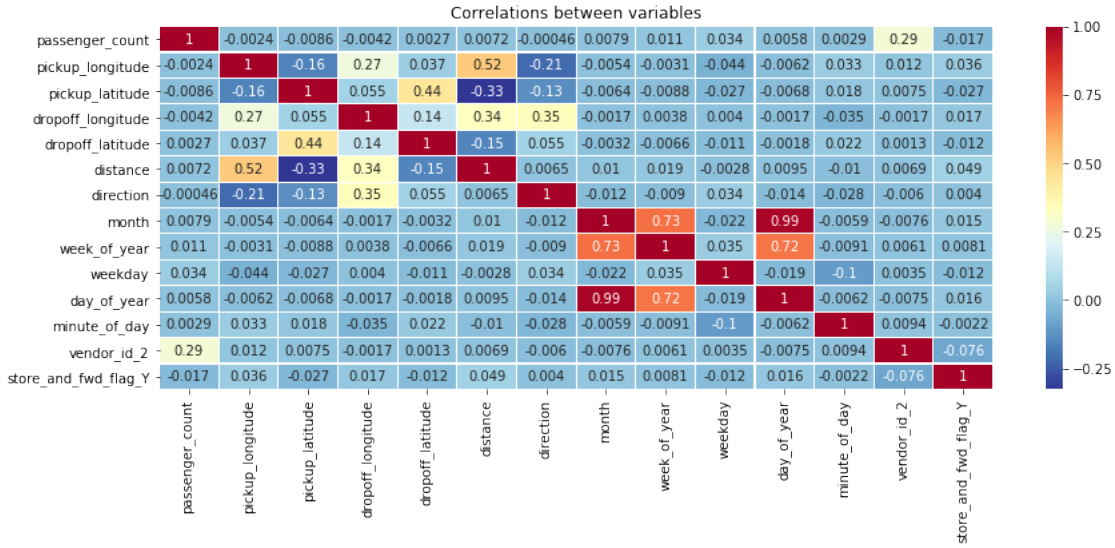
-week_of_year

-weekday

-day_of_year

-minute_of_day

We first use heatmap indicating correlations between variables, the graph below shows that "month" is highly correlated with "day_of_year" and "week_of_year", "day_of_year" and highly correlated with "week_of_year". In order to avoid multicollinearity, we remove "month" and "week_of_year".



Then, we calculate feature importances from random forest model, the higher, the more important the feature. The results show that he feature "store_and_fwd_flag" is the least important, and "passenger_count" and "vendor_id" are also considered less important, but in order to make sure that it can be ignored in our model analysis, I created three feature sets, feature set 1 with all the features mentioned above, feature set 2 drops the feature "store_and_fwd_flag", feature set 3 removes "store_and_fwd_flag", "vendor_id" and "passenger_count", and compared these three sets in modeling performance.

Rank	Features	Importances
1	distance	0.664265
2	minute_of_day	0.091556
3	direction	0.044042
4	dropoff_latitude	0.039960
5	dropoff_longitude	0.033780
6	pickup_longitude	0.033205
7	day_of_year	0.028955
8	weekday	0.027996
9	pickup_latitude	0.027978
10	passenger_count	0.005301
11	vendor_id	0.002560
12	store_and_fwd_flag	0.000402

Analysis of Performance of Different Model Types

In this section, I compared three different model types, random forest, gradient boosting and LightGBM.

Random forest is a good algorithm dealing with non-linear regression, it generates multiple prediction models at the same time, and summarizes the results of those models to improve the accuracy of the prediction. One of the advantages of random forest is that it can always find a good solution even before tuning hyperparameter, because it is not very sensitive to hyperparameter. The parameters that can be tuned include the number of generators, the number of subsamples, the number of leaves.

The basic idea of Gradient Boosting is to make every round of basic learner pay more attention to the samples of the last round of learning errors in the training process. The negative gradient is used as a measure of the errors of the previous round of basic learning, and the errors of the previous round are corrected by fitting the negative gradient in the next round of learning. By tuning tree-specific parameters, boosting parameters and miscellaneous parameters, the performance of gradient boosting can be greatly improved.

LightGBM is an improved version of gradient boosting, the negative gradient of the loss function is used as the residual approximation of the current decision tree to fit the new decision tree. It is faster and more accurate. Num_leaves, min_data_in_leaf and max_depth are three very important parameters, and there are other parameters that help prevent over-fitting and get better result.

The table below summarize performance of different models before and after hyperparameter tuning, using different feature sets. Overall, there are not very big difference of the performance of the three feature sets, removing features “store_and_fwd_flag”, “vendor_id” and “passenger_count” can slightly improves models.

Given the description of three models above, we can guess that LightGBM might be the best one, the results show that LightGBM and random forest get almost the same results before hyperparameter tuning, but after hyperparameter tuning, the performance of LightGBM is greatly improved, and LightGBM gets the best score of three models.

Model		Feature Set 1	Feature Set 2	Feature Set 3
Random Forest	Before Tuning	0.3472	0.3475	0.3469
	After Tuning	0.3447	0.3452	0.3446
LightGBM	Before Tuning	0.3432	0.3432	0.3423
	After Tuning	0.3101	0.3098	0.3092
Gradient Boosting	Before Tuning	0.4050	0.4050	0.4050
	After Tuning	0.3880	0.3880	0.3880