

GB656_FinalProject: New York Taxi Fare Prediction

Chieh-Yin (Jenny), Liao

Introduction

Ober is a Taxi services company based in California and now seeking to expand its market to New York City. As analytics team members in Ober, we need to predict Ober's revenue that might be earned in New York City in order to evaluate whether it's appropriate to expand. Therefore, we collect a bunch of historical taxi data. We would like to predict the fare amount for a taxi ride in New York City given the pickup and dropoff locations and times. In the end, we can use predicted fare to forecast our revenue.

Methodology

1. Load the historical taxi data

First, we load and explore our data:

taxi_train.head()

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	5.0	2014-07-16 10:57:00+00:00	-73.996147	40.741890	-73.992203	40.739425	6.0
1	3.7	2010-01-31 10:53:00+00:00	-74.001633	40.730766	-73.997108	40.737533	1.0
2	7.7	2010-12-04 14:26:13+00:00	-73.996597	40.736568	-73.982155	40.744322	1.0
3	5.7	2010-08-19 16:33:00+00:00	-73.973831	40.763718	-73.989418	40.771522	1.0
4	12.5	2011-08-31 08:21:47+00:00	-73.917397	40.746487	-73.973755	40.763836	1.0

taxi_train.describe()

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	5545831.00	5545831.00	5545831.00	5545792.00	5545792.00	5545831.00
mean	11.33	-72.51	39.91	-72.51	39.92	1.69
std	9.78	12.76	9.98	12.99	9.91	1.33
min	-70.00	-3426.61	-3492.26	-3408.43	-3429.36	0.00
25%	6.00	-73.99	40.73	-73.99	40.73	1.00
50%	8.50	-73.98	40.75	-73.98	40.75	1.00
75%	12.50	-73.97	40.77	-73.96	40.77	2.00
max	1273.31	3454.85	3352.85	3211.58	3351.47	208.00

taxi_train.isnull().sum()

fare_amount	0
pickup_datetime	0
pickup_longitude	0
pickup_latitude	0
dropoff_longitude	39
dropoff_latitude	39
passenger_count	0
dtype: int64	

We found that there are some outliers and invalid data in this dataset. For example, there are some weird values such as negative fare amount, negative pickup longitude / latitude which is not possible in the case of NYC, and a maximum passenger of 208.

2. Data Cleaning

Since we found some outliers and missing data in this dataset, we plan to remove them. We set the criteria range as below:

- $1 \leq \text{fare_amount} \leq 500$
- $-75 \leq \text{pickup / dropoff longitudes} \leq -72$
- $40 \leq \text{pickup / dropoff latitudes} \leq 42$
- $1 \leq \text{passenger_count} \leq 6$

taxi_train.shape	taxi_train.shape
(5545831, 7)	(5409799, 7)

We drop approximately 136,000 rows to make our data more precise.

3. Feature Engineering

There are only few features within the dataset, and it is essential to convert these into useful information to model them effectively. Feature engineering is the process of extracting information from existing data in order to improve the performance of the model. Therefore, we create these new features:

a. Calculate the Haversine distance

When considering taxi fare, **distance** should be the most predictive feature that affects the fare. We use Haversine distance to calculate the distance between pickup and dropoff points. Haversine distance is used to calculate the **great-circle distance** between two points on a sphere given their longitudes and latitudes.

$$d = 2r \arcsin \left(\sqrt{\sin^2 \left(\frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

- ϕ_1, ϕ_2 are the latitude of point 1 and latitude of point 2,
- λ_1, λ_2 are the longitude of point 1 and longitude of point 2.

b. Extract parts of dates

We believe that **time** can be another important feature. For example, it is common to have traffic jams in rush hour that will increase total ride time and the taxi fare. Therefore, we extract several columns [pickup_year, pickup_month, pickup_day, pickup_hour, pickup_weekday] from pickup_datetime.

c. Create Base Fare

Wikipedia illustrates that as of June 2006, fares begin at \$ 2.50, 3.00 after 8:00 p.m., and \$3.50 during the peak weekday hours of 4:00 - 8:00 pm. Base fare is estimated based on these slabs of the time range.

d. Add distance from popular landmarks

We calculate the difference between the popular landmarks (listed below) and dropoff points by using Haversine distance. Since we believe traffic jams usually happen nearby those popular landmarks which might increase the taxi fare.

- Times Square: (40.7580° N, 73.9855° W)
- JFK Airport: (40.6413° N, 73.7781° W)
- Statue of liberty: (40.6892° N, 74.0445° W)

Here are the features that we created from the original dataset:

```
taxi_train.head()
```

	passenger_count	distance	pickup_year	pickup_month	pickup_day	pickup_hour	pickup_weekday	basic_fare	time_square_distance	jfk_distance	statue_distance
0	6.0	2.342486	2013	11	6	11	Sunday	2.5	0.709867	21.226512	9.564284
1	1.0	0.946365	2011	3	4	18	Friday	3.5	2.551654	20.391082	7.079799
2	6.0	0.430317	2014	7	2	10	Wednesday	2.5	2.139922	21.079769	7.110075
3	1.0	0.842717	2010	1	6	10	Sunday	2.5	2.475470	21.328510	6.691976
4	1.0	1.490212	2010	12	5	14	Saturday	2.5	1.545802	20.655997	8.068041

4. Modeling

I leverage a Linear regression model, Regression tree, random forest, boosted tree, and neural network as below to forecast the taxi fare in New York City. We compute a **Root Mean Squared Error (RMSE)** to evaluate our model's performance, which RMSE is also the way that Kaggle scores.

First, I split 25% of random samples (1,352,450) into test datasets and the remaining into train datasets in order to evaluate in-sample and out-sample RMSE for not overfitting. Fare_amount as the dependent variable (y) and the other 14 features as the independent variables (X).

a. Linear Regression

Starting with a simple one, I fit the training data into a linear regression model. Then, I used both train and test samples that I split to check model performance, the RMSE is as below:

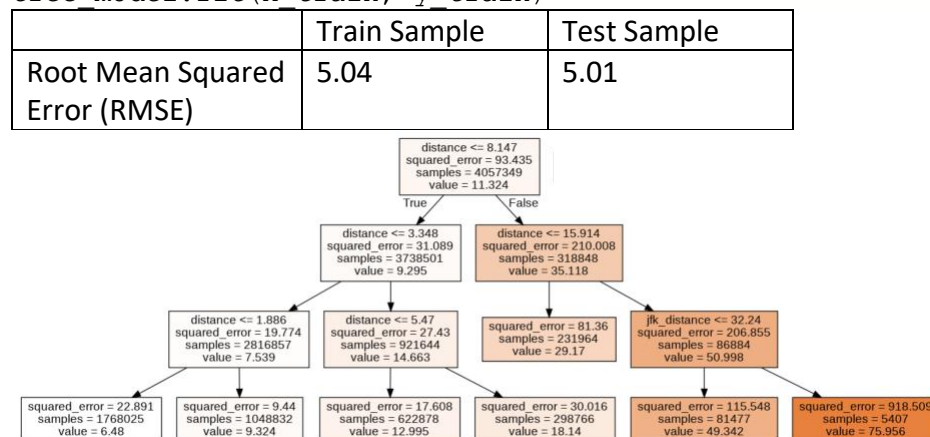
```
linear_model = LinearRegression(fit_intercept=True)
linear_model.fit(X_train, y_train)
```

	Train Sample	Test Sample
Root Mean Squared Error (RMSE)	5.37	5.36

b. Regression Tree

I fit the data into a Regression tree model. To reduce the complexity and improve predictive accuracy by reducing overfitting, I pruned the regression tree by setting maximum 7 leaf nodes. Then, I used RMSE to check the model performance below:

```
tree_model = DecisionTreeRegressor(max_leaf_nodes=7)
tree_model.fit(X_train, y_train)
```

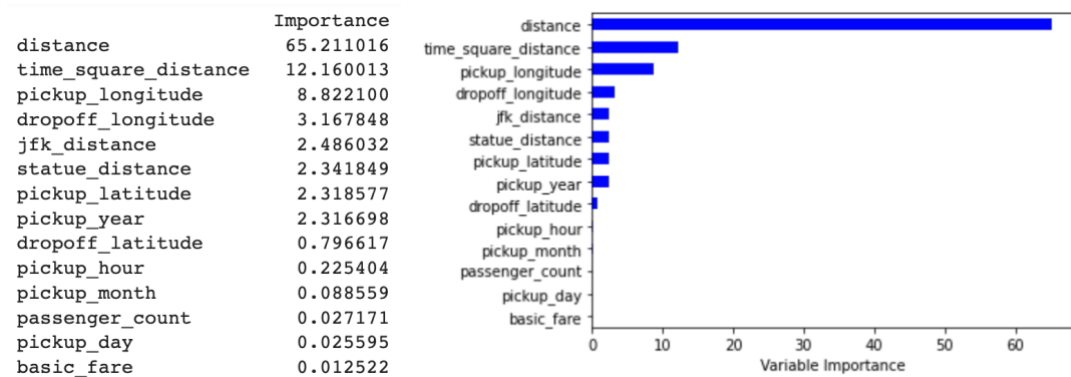


c. Random Forest

I fit in a random forest model by using 50 numbers of trees, half variables at each split, and a maximum of 10 depths to prevent overfitting. Then, I used RMSE to check the model performance below:

```
rf_model = RandomForestRegressor(n_estimators=50, random_state=1,
max_depth=10 max_features=0.5, min_samples_leaf=3)
rf_model.fit(X_train, y_train)
```

	Train Sample	Test Sample
Root Mean Squared Error (RMSE)	3.78	3.84



I plot the importance of each variable in the Random Forest Model, verifying the hypothesis we made previously that distance should be the most predictive feature that affects the fare.

d. Boosting

I fit in a boosted tree model by using 100 times and learning rate 0.05 of boosting. Then, I used RMSE to check the model performance below:

```
boost_model = GradientBoostingRegressor(n_estimators=100,
learning_rate=0.05,random_state=1)
boost_model.fit(X_train, y_train)
```

	Train Sample	Test Sample
Root Mean Squared Error (RMSE)	4.17	4.14

e. Neural Network

I fit in a 2 hidden layer neural network model by using 10 neurons in the first layer and 5 neurons in the second layer. Both hidden layers use the 'Relu' activation function. Then, I used RMSE to check the model performance below:

Model: "sequential_1"

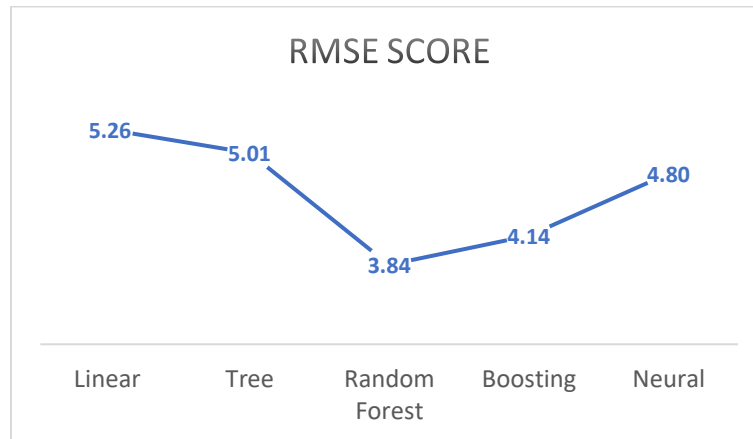
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 10)	110
dense_2 (Dense)	(None, 5)	55
dense_output (Dense)	(None, 1)	6
Total params: 171		
Trainable params: 171		
Non-trainable params: 0		

```
nn_model = Sequential()
nn_model.add(Dense(10, input_shape=(14, ), activation='relu',
name='dense_1'))
nn_model.add(Dense(5, activation='relu', name='dense_2'))
nn_model.add(Dense(1, activation='linear', name='dense_output'))
```

	Train Sample	Test Sample
Root Mean Squared Error (RMSE)	4.80	4.79

Results

To sum up, in order to evaluate a model's performance, I apply the RSME score in test sample to compare these models (Linear regression model, Regression tree, random forest, boosted tree, and neural network). RSME is the standard deviation of the residuals, lower values of RMSE indicate a better fit. As the graph shows as below, RMSE Score in Random Forest (3.84) is smaller than others, which means that the Random Forest model has a better prediction in this taxi fare case.




Therefore, we predict the fare_amount from test sample by using our Random Forest model.

submit

	key	fare_amount
0	2015-01-27 13:08:24.0000002	10.140592
1	2015-01-27 13:08:24.0000003	10.180844
2	2011-10-08 11:53:44.0000002	5.467028
3	2012-12-01 21:12:12.0000002	8.445412
4	2012-12-01 21:12:12.0000003	14.201637

Ultimately, we ended up with a 3.31 RMSE score, which is a relatively good score compared to other submissions.

Submission and Description		Private Score ①	Public Score ①
 rf_submission.csv	Complete (after deadline) · 1s ago	3.30671	3.30671