New York City
Taxi Fare Prediction

Zhibin Huang
hzhibin@bu.edu

Zhizhou Qiu
qzhizhou@bu.edu

Tianyi Tang
tty8128@bu.edu

Abstract

It is to predict the fare amount (inclusive of tolls) for a taxi ride in New York City given the pick-up and drop-off locations. A basic estimate can be gained based on the distance between the two points, but this will result in an RMSE of $5-8$, depending on the model used. By implementing Gradiant Boost, Random Forest and Neural Network, results of around 3 are retrieved.

1 Data Preprocessing and Features Engineering

1.1 Dataset Overview

Github: `https://github.com/qzhizhou/New-York-Taxi-Fare-Prediction`

1.1.1 Source:

`https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/data`

1.1.2 Features:

pickup datetime - timestamp value indicating when the taxi ride started.

pickup longitude - float for longitude coordinate of where the taxi ride started.

pickup latitude - float for latitude coordinate of where the taxi ride started.

dropoff longitude - float for longitude coordinate of where the taxi ride ended.

dropoff latitude - float for latitude coordinate of where the taxi ride ended.

passenger count - integer indicating the number of passengers in the taxi ride.

1.1.3 Target:

fare amount: float dollar amount of the cost of the taxi ride.

1.1.4 Dataset size:

55,000,000 (55M) rows.

Before preprocessing, we use visualization tools to explore the dataset.

1.2 Data Cleaning:

Considering the computational limitation of our resources, we only retrieve 10,000,000 (10M) records in the data set. We store all records in a new .csv file as our dataet.

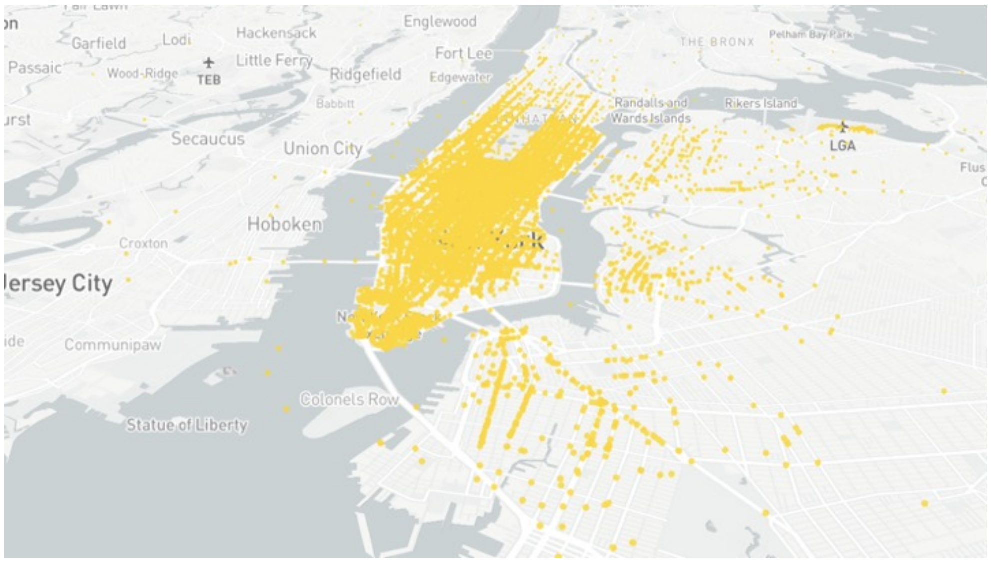

Figure 1: Pick Up Locations in NYC

The first step is data cleaning. According to our observation, the raw dataset contains some data that is not suitable for model training. We need to drop rows with:

- NaN rows
- Negative fare amount
- Coordinates far away from New York City
- Coordinates located in water
- Passenger count greater than six(not reasonable in a normal ride)

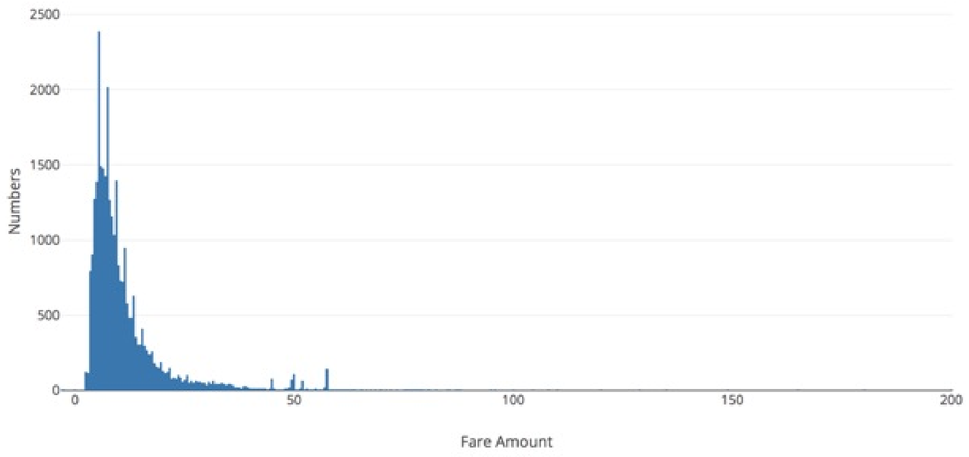

Figure 2: Fare Value Distribution

So, we have to clean the dataset before we use it.

We drop all data with any kind of the unreasonable types mentioned above. The original data size is 10M rows, after cleaning it becomes 9,750,645 rows. This new dataset is our training set.

1.3 Features Engineering:

As mentioned in 1.1, there are six features in training data set. However, these features cannot be used directly since they do not include enough information. What we care about in this problem is features like the distance the taxi has traveled. Moreover, we need to split the timestamp value to extract useful information of the taxi ride.

We compute absolute latitude and longitude difference respectively and also compute Euclidean distance and Manhattan distance of abs-diff-latitude and abs-diff-longitude. Then we extract year, month, day, hour and week-day from original feature pickup-datetime. After that, we delete those useless features including pickup-datetime, pickup-latitude, pickup-longitude, dropoff-latitude and dropoff-longitude. Finally, we keep all useful features as input features for our ML models. The input features include:

- abs-diff-longitude: absolute value of difference between pick up longitude and drop off longitude.
- abs-diff-latitude: absolute value of difference between pick up latitude and drop off latitude.
- euclidean: euclidean distance of the taxi ride.
- manhattan: manhattan distance of the taxi ride.
- year: year of the taxi ride started time.
- month: month of the taxi ride started time.
- day: day of the taxi ride started time.
- hour: hour of the taxi ride started time.
- weekday: weekday of the taxi ride started time.
- passenger-count: the number of passengers in the taxi ride.

2 Model Selections

Since we have no idea about which ML algorithm is most suitable to solve this problem, we decide to choose four different kind of models and then compare results to find the best method.

2.1 Linear Regression

As the simplest model for regression problem, we first try linear regression for this problem. Although it is not the precise model, it can get the result very quickly and we can also take it as a baseline model for comparison.

2.2 XGBoost

XGBoost is one of the gradient boost method called eXtreme Gradient Boost used for supervised Machine Learning. The weak learner used in XGBoost is generally the decision tree. The implementation of the model supports the features of the scikit-learn and R implementations, with new additions like regularization. The implementation of the algorithm was engineered for efficiency of compute time and memory resources. A design goal was to make the best use of available resources to train the model.

2.3 LightGBM

We started with applying the random forest on the targeted dataset. Specifically, the optimal estimator of this classier is found by Grid Search. In a random forest, every tree is bagged together which means it can only reduce the variance of this problem. In order to reduce the bias of this problem, we decided to try the Gradient Descent Boosting tree (GDBT) model which named LightGBM.
LightGBM is a new GBDT algorithm with GOSS and EFB[1]. Gradient-based One-Side Sampling (GOSS) can achieve a good balance between reducing the number of data instances and keeping the accuracy for learned decision trees. Exclusive Feature Bundling (EFB) effectively reduces the number of features. The combination of two methods enable LightGBM with fast training speed

and high efficiency, lower memory usage, compatibility with large datasets and parallel learning supported.

2.4 Neural Network

An artificial neural network is a model of computation inspired by the structure of neural networks in the brain.[2] Considering the property of this project, we use a relatively simple neural network for predicting fare amount.

There are six layers in our neural network, including one input layer, four hidden layers and one output layer. The activation function of hidden layer is ReLU function, and the activation function of output layer is linear function since it is a regression problem. The number of neurons in each hidden layer is decreasing because the training process will be faster. The structure of our neural network is in Figure 3.

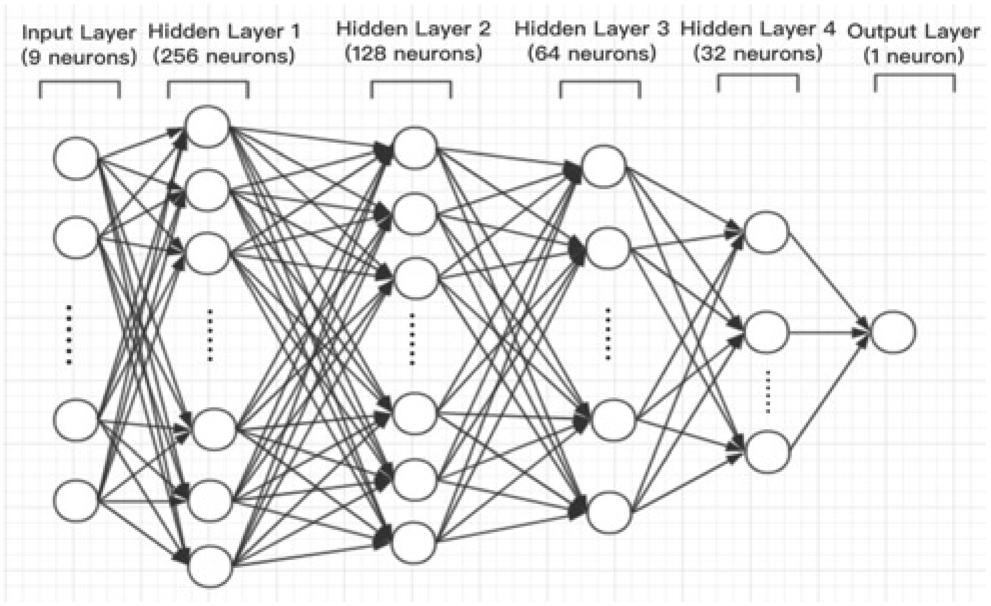
Input Layer (9 neurons)
Hidden Layer 1 (256 neurons)
Hidden Layer 2 (128 neurons)
Hidden Layer 3 (64 neurons)
Hidden Layer 4 (32 neurons)
Output Layer (1 neuron)

Figure 3: Structure of NN

The number of hidden layers of neural network is slightly different between two input features and all input features, but the overall structure is the same.

3 Model Training

Initially, we use only two features - abs_diff_longitude and abs_diff_latitude to train our model to make sure that our implementation is correct. After doing this, we use all features to train each model.

3.1 Linear Regression

As mentioned before, we firstly take only two features. By calling numpy.linalg.lstsq function, we get the least-squares solution to a linear matrix equation. Afterwards, we take all features as input and train the model.

3.2 XGBoost

In training XGBoost model, we separate the data into 99% training and 1% validation to get the validation loss. First we use two features and then we take all features. Parameters we set of XGBoost are: learning_rate=0.3, max_depth=6, n_jobs=-1, silent=False.

3.3 LightGBM

The main job of the LightGBM is adjusting the parameters for the model. For this purpose, we use Grid Search validation to find optimal parameters based on the validation dataset which is 1% of the whole training set.

We adjust parameters of LightGBM by 4-step modification. First, we choose a higher learning rate, around 0.1, in order to speed up the convergence. Then, we do a basic parameter adjustment for the decision tree. We import the GridSearch function from Scikit-Learn and part of the result of adjustment is in Table 1.

Table 1: Result of adjustment

Max_depth	Num_leaves	Means & Std
3	31	mean: -1.88629, std: 0.13750
5	31	mean: -1.86917, std: 0.12590
7	**31**	**mean: -1.86024, std: 0.11364**

From Table 1, we can easily draw the conclusion that depth = 7 and num_leaves = 31 is the optimal combination of the model. After that, we do regularization parameter adjustment with λ_{l1} and λ_{l2} to confront the overfitting. Finally, we reduce the learning rate gradually to improve accuracy. The final result of the parameters is included in Table 2.

Table 2: Parameters

objective	regression	**scale_pos_weight**	1
metric	rmse	**num_threads**	4
num_boost_round	1000	**boosting_type**	gbdt
learning_rate	0.034	**zero_as_missing**	True
num_leaves	31	**seed**	0
max_depth	-1	**eval_freq**	50
subsample	0.8	**min_child_samples**	10
colsample_bytree	0.6	**min_child_weight**	1
min_split_gain	0.5		

The score of two different models with different input features is shown in Table 3.

Table 3: Result

Input Features	**Score**
Only abs_diff	3.65
All features	**3.17**

3.4 Neural Network

In training neural network, we set batch size to 256 in the beginning. However, we find that this batch size is too small that the validation error fluctuate greatly. Hence, we increase batch size to 3,000. We also use Adam optimizer in our training process.

Adam[3] is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal re-scaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients.

Algorithm Adam pseudocode.

```
Require: $\alpha$: Stepsize
Require: $\beta_1 \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
        $m_0 \leftarrow 0$(Initialize $1^{st}$ moment vector)
        $v_0 \leftarrow 0$(Initialize $2^{nd}$ moment vector)
        $t \leftarrow 0$(Initialize timestep)
    while $\theta_t$ not converged do
        $t \leftarrow t + 1$
        $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$(Get gradients w.r.t. stochastic objective at timestep t)
        $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$(Update biased first moment estimate)
        $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$(Update biased second raw moment estimate)
        $\widehat{m_t} \leftarrow m_t/(1 - \beta_1^t)$(Compute bias-corrected first moment estimate)
        $\widehat{v_t} \leftarrow v_t/(1 - \beta_2^t)$(Compute bias-corrected second raw moment estimate)
        $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m_t}/(\sqrt{\widehat{v_t}} + \epsilon)$(Update parameters)
    end while
    return $\theta_t$t(Resulting parameters)
```

We change three parameters of model to find the best result: 1) proportion of validation set; 2) learning rate; 3) epoch. First, we set the proportion of validation set to 10%, 5%, 1% and fix learning rate to 0.001, epoch to 20.

Table 4: Result

Proportion of Validation Set	**Score**
10%	3.69
5%	3.64
1%	**3.61**

It shows that it is better for us to set the proportion of validation set to 1%. The performance of neural network gets better with more training data in each epoch.

Then, we fix proportion of validation set to 1%, set learning rate to 0.01 and 0.001, epoch to 20, 30, 50.

Table 5: Result

Learning Rate	**Epoch**	**Score**
0.01	20	4.07
0.01	30	3.60
0.01	50	3.78
0.001	20	3.61
0.001	30	3.63
0.001	**50**	**3.56**

We choose the neural network with proportion of validation set to 1%, learning rate to 0.001, epoch to 50 as our final model for comparison.

4 Test Result

After we finish training model, we use test set to evaluate the performance of ML algorithm. The score is defined by Kaggle, which equals to the Root Mean Square Error (RMSE) of model. RMSE represents how concentrated the data is around the line of best fit.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y_j})^2}$$

Therefore, the lower score means that the algorithm is better. The score of each model is shown in Table 6.

Table 6: Result

Model	**Score with only abs-diff**	**Score with all features**
Linear Regression	5.75	5.63
XGBoost	3.78	3.29
LightGBM	3.65	3.17
Neural Network	4.32	3.56

References

[1] Ke G., J.A. Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W. & Liu, T. Y. (2017) Lightgbm: *A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems.* pp. 3146-3154
[2] Shalev-Shwartz, S., & Ben-David, S. (2017) *Understanding machine learning: From theory to algorithms. Cambridge: Cambridge University Press.*
[3] Kingma, D. and Ba, J. (2019) *Adam: A Method for Stochastic Optimization. [online] arXiv.org. Available at: https://arxiv.org/abs/1412.6980* [Accessed 2 May 2019]