Taxi Fare Prediction Based On Machine Learning

Alissa de Bruijn
Shuhao Chang
Yin Deng
Taehyeong Noh

November 29, 2018

1 Introduction

Recently, the revolution of the vehicle industry is becoming true. More and more people prefer to choose taxi-hailing services including DiDi and Uber than to buy their own cars due to convenience. In a big city like New York, especially, there is a significantly increasing number of Uber cars in contrast to private-owned cars, gradually becoming the mainstream option of transportation for people. This phenomenon is and will continue affecting people's lives and elevate our productivity.

However, we notice that the estimated fare for Uber often has a large variation for the same distance. This sometimes causes overcharged problem for many people and therefore hurts the company's credibility. We think that taking taxis for transportation can save us the hassle of high fuel costs, losing value on the cars, and most importantly, greener environment for the society.

Hence, our project is to build a predictive model such that our taxi fare prediction is as close to the real world cost as possible. By implementing several machine learning models such as Neural Network with Keras, Regression, Random Forest and eXtreme Gradient Boosting with sklearn, we aim to produce a baseline model that predicts the fare amount given the pickup and dropoff locations. Our ultimate goal is to minimize the deviation of fare prediction from the Uber app.

The report is divided into 5 sections. Section 2 describes the data alongside with the data analysis. Section 3 introduces the algorithms and models. The results are given in Section 4. Lastly, the conclusion is given in Section 5.

2 Exploratory Data Analysis

In this section, we answer some basic questions about the collection of our datasets: How was the data collected? Meanings? How big? Qualities? Then, we use tables to demonstrate detailed discoverings that we found in these datasets. Last, we plot various graphs to present the correlations and diversities of our datasets.

2.1 Data Collection

The data of this project was collected from Kaggle NYC Taxi fare competition. It contains information such as fare amount, pickup and dropoff longitudes, passenger counts, etc. This training dataset itself is 5.7GB, containing over 55 millions rows of taxi activities. Most of the datasets are accurate, they do not contain missing values or other strange values for the different features.

However, some examples are bad and we remove them while preprocessing and loading them in Python.

2.2 Data Cleansing

In this project, we are dealing with a large training dataset that the size is over 5GB. Using Pandas DataFrame directly will simply cause the computer to be out of memory space.

When using the training.csv file, we split it into chunks of 5000000 lines and then use the split function in Linux in the terminal to give the new files names, so after the command finish, we will have multiple files that have fewer rows and thus will be easier for Pandas to read. The command is rather long so we omit it here. The next step is to check each 'csv' file and do some data cleaning so that we can have a better prediction later in our project.

Just like the example in NYC Taxi fare, we notice that the minimum fare is negative, so we remove these rows from our datasets. Also, there are zeros or missing data in the files, we remove those as well.

```
df_train01 = df_train01[df_train01.fare_amount>=0]
df_train01 = df_train01.dropna(how = 'any', axis = 'rows')
df_train01 = df_train01[(df_train01 != 0).all(1)]
```

Now, we can see the size of remaining data in this file:

```
print("New size: %d" % len(df_train01))
# New size: 4882711
```

We run df.describe() and df.corr() to provide a quick summary of our datasets, as the table shown below.

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	4.882711e+06	4.882711e+06	4.882711e+06	4.882711e+06	4.882711e+06	4.882711e+06	4.882711e+06
mean	2.499806e+06	1.133382e+01	-7.391522e+01	4.069223e+01	-7.391187e+01	4.069003e+01	1.690800e+00
std	1.443433e+06	9.727828e+00	7.766811e+00	6.737989e+00	7.915352e+00	7.491393e+00	1.314368e+00
min	0.000000e+00	1.000000e-02	-3.426609e+03	-3.488080e+03	-3.412653e+03	-3.488080e+03	1.000000e+00
25%	1.249656e+06	6.000000e+00	-7.399227e+01	4.073650e+01	-7.399158e+01	4.073553e+01	1.000000e+00
50%	2.499651e+06	8.500000e+00	-7.398209e+01	4.075332e+01	-7.398060e+01	4.075383e+01	1.000000e+00
75%	3.749972e+06	1.250000e+01	-7.396829e+01	4.076753e+01	-7.396529e+01	4.076840e+01	2.000000e+00
max	4.999999e+06	9.520000e+02	3.439426e+03	2.977031e+03	3.457622e+03	3.345917e+03	2.080000e+02

Figure 1: df.describe()

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	distance	hour_density	month_density	pickup_density	dropoff_density
pickup_longitude	1.000000	0.698986	0.394195	0.335040	-0.052649	0.038886	0.011473	-0.115395	-0.018611
pickup_latitude	0.698986	1.000000	0.320550	0.471193	-0.067127	0.069558	0.013423	-0.013035	-0.002790
dropoff_longitude	0.394195	0.320550	1.000000	0.663165	0.031776	-0.014018	0.009970	-0.028490	-0.120456
dropoff_latitude	0.335040	0.471193	0.663165	1.000000	-0.009843	0.029398	0.012057	-0.011519	-0.006325
distance	-0.052649	-0.067127	0.031776	-0.009843	1.000000	-0.049231	-0.007067	-0.056844	-0.092569
hour_density	0.038886	0.069558	-0.014018	0.029398	-0.049231	1.000000	0.003254	-0.003983	0.001972
month_density	0.011473	0.013423	0.009970	0.012057	-0.007067	0.003254	1.000000	-0.001772	-0.006643
pickup_density	-0.115395	-0.013035	-0.028490	-0.011519	-0.056844	-0.003983	-0.001772	1.000000	0.040906
dropoff_density	-0.018611	-0.002790	-0.120456	-0.006325	-0.092569	0.001972	-0.006643	0.040906	1.000000

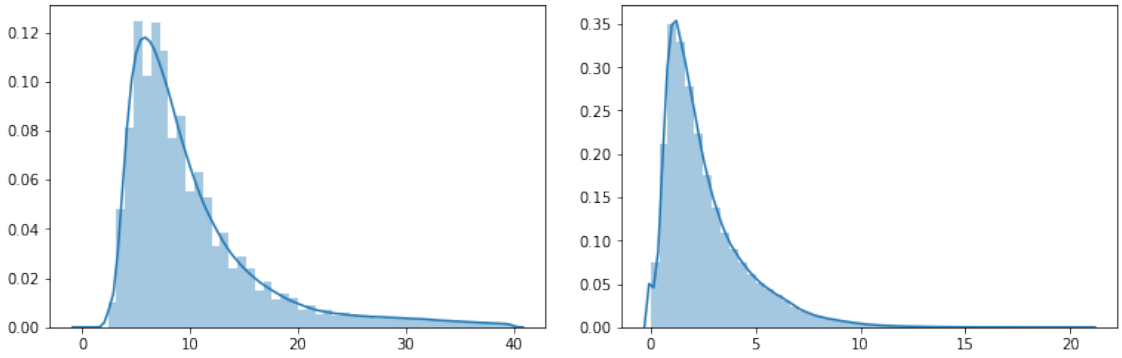
Figure 2: df.corr()

What we notice here is that the average fare amount is $11.3 and the standard deviation of fare amount is $9.73 which is too high. In fact, our goal is to reduce this amount after we use our

prediction model. As of right now, we are not able to see any correlation between these features.

Finally, we can overwrite our previous training set with the modified one. Next, we will visualize the NYC map and refine our datasets even better by removing outliers.

2.3 Data Analysis

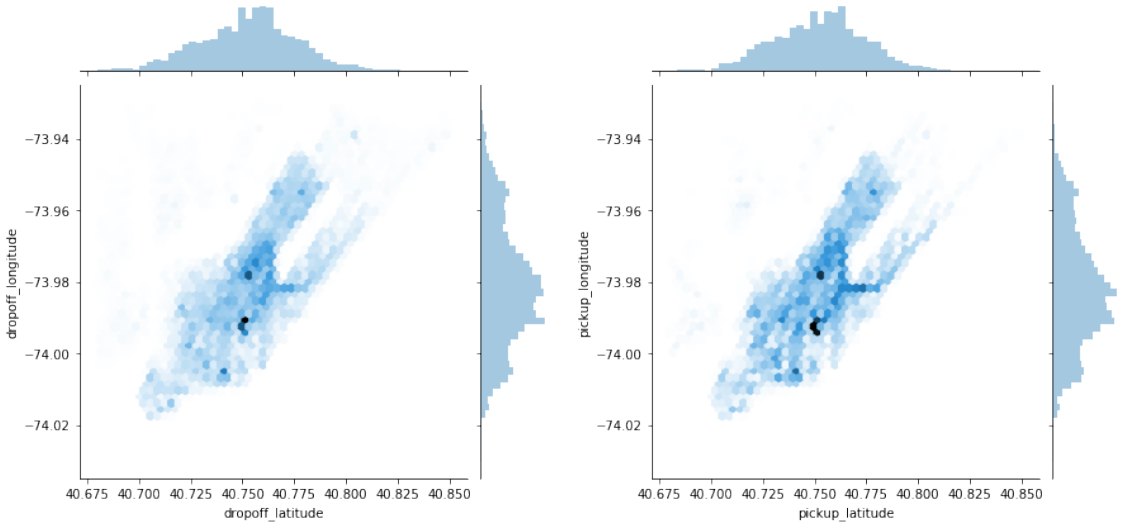


(a) Fare Amount (b) Distance

Figure 3: Distributions

On the left, Figure 3a shows that the distribution of the fare amount is centered around $7 dollars, and on the right, Figure 3b shows the distribution of the distance which is centered around 4.

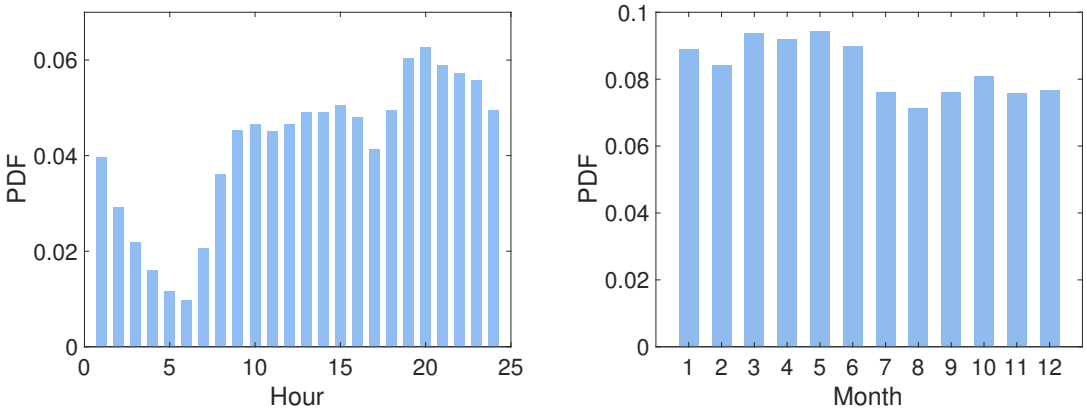
Using the Seaborn jointplot, Figure 4 below show not only the distribution of both the drop-off and the pickup location, but also their density distribution in the city. These graphs give us a sense where the popular places are that people tend to take Uber. As stated in the introduction, Uber is very popular in the New York City.



(a) Drop-off (b) Pickup

Figure 4: Location Distribution

Figure 5 shows the time distribution hourly and monthly. From Figure 5a we can derive that the busiest hours are after 20:00. Furthermore, Figure 5b shows that the demand for a taxi is a little bit higher from January to June due to the weather in New York.



(a) Hour PDF

(b) Month PDF

Figure 5: Time Distribution

In Figure 6, most of the taxi rides contain one customer, so we suspect that the passenger count is highly related to fare amount prediction.

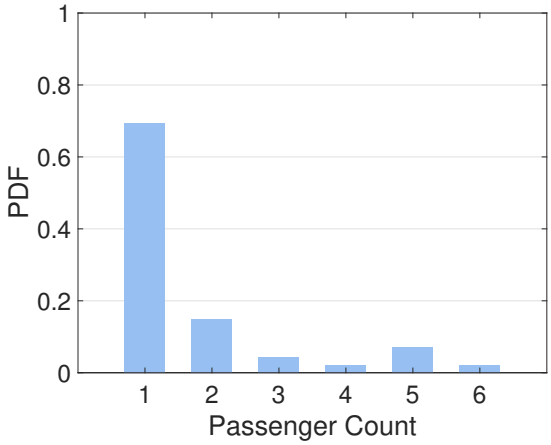


Figure 6: Passenger Count

Figure 7 shows the correlation between the variables. From this Heatmap, we can derive that there is a relatively high correlation between distance and average fare amount, pickup longitude and pickup latitude, dropoff longitude, and dropoff latitude and a negative correlation between passenger count and average fare amount.

Figure 7: Correlation

Figure 8 specify pick-up points of each ride. Longitude range of map is from -72.8 to -74.8. Latitude range of map is from 40.4 to 41.8. Most of riding points are concentrated at Manhattan, Brooklyn, and Queens area. For the other concentrated spots, there are two airports, JFK international airport and LaGuardia airport. On the sea and river there are also several red points are observed. These are data error from the Uber longitude and latitude recording system.

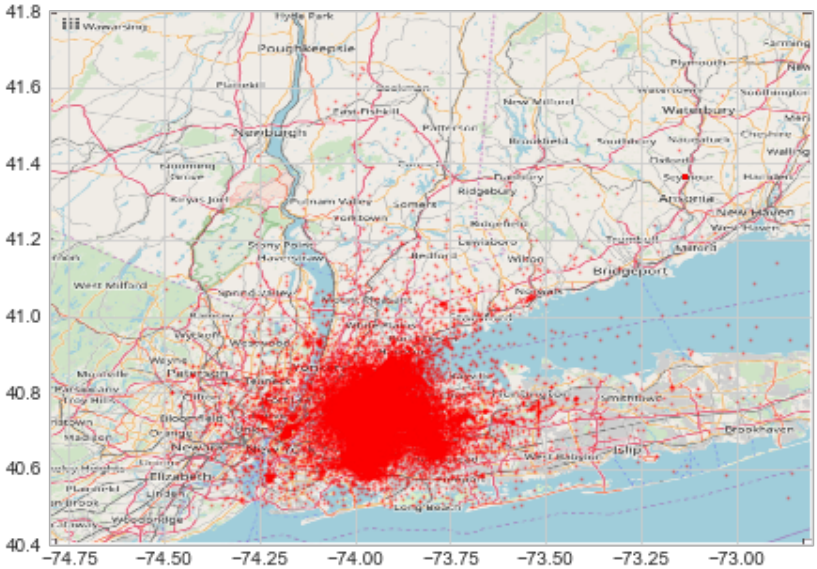


Figure 8: Pick-Up Location Mapping

Figure 9 represents drop-off points of each ride. Longitude and latitude range of map are same as

above. drop-off points are more spread out. This result shows that people use Uber to go back to home which is located far from the city center. This map also contains several points on the sea and river according to the same reason mentioned above.

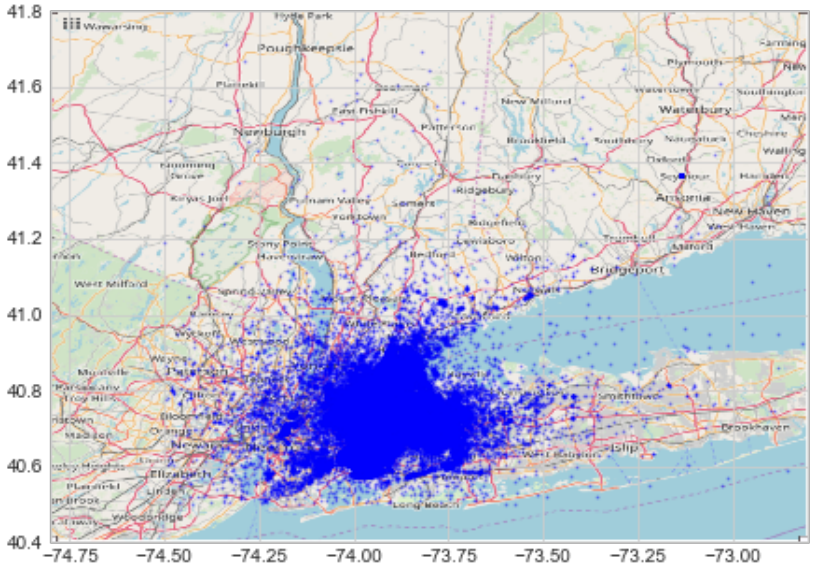


Figure 9: Drop-Off Location Mapping

2.4 Data Augmentation

In addition to the original dataset, we make use of the Uber API to expand the dimensions of our dataset features. Based on our daily experience, we also hypothesize that the density of the locations and the traffic hours may affect our predictions as well. So we add those information in our datasets. Details are in the Jupyter Notebook, here we show a sample of the result:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	distance	hour_density	month_density	pickup_density	dropoff_density
count	345617.000000	345617.000000	345617.000000	345617.000000	345617.000000	345617.000000	345617.000000	345617.000000	345617.000000
mean	-73.981723	40.753079	-73.980788	40.753670	2.395780	0.482349	0.840443	0.069473	0.053922
std	0.016227	0.021286	0.016970	0.022362	1.697335	0.121359	0.077204	0.063041	0.054454
min	-74.019978	40.700013	-74.019977	40.700002	0.000000	0.075644	0.709408	0.000231	0.000231
25%	-73.992790	40.738408	-73.992200	40.738636	1.183537	0.469096	0.759869	0.031017	0.023841
50%	-73.982680	40.753937	-73.981949	40.754672	1.930793	0.500414	0.858729	0.056479	0.043053
75%	-73.971145	40.766973	-73.970134	40.768012	3.148207	0.543907	0.919328	0.089810	0.066432
max	-73.930000	40.819980	-73.930000	40.819996	13.770677	0.650546	0.935323	0.489327	0.483540

Figure 10: df.describe()

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	distance	hour_density	month_density	pickup_density	dropoff_density
pickup_longitude	1.000000	0.698986	0.394195	0.335040	-0.052649	0.038886	0.011473	-0.115395	-0.018611
pickup_latitude	0.698986	1.000000	0.320550	0.471193	-0.067127	0.069558	0.013423	-0.013035	-0.002790
dropoff_longitude	0.394195	0.320550	1.000000	0.663165	0.031776	-0.014018	0.009970	-0.028490	-0.120456
dropoff_latitude	0.335040	0.471193	0.663165	1.000000	-0.009843	0.029398	0.012057	-0.011519	-0.006325
distance	-0.052649	-0.067127	0.031776	-0.009843	1.000000	-0.049231	-0.007067	-0.056844	-0.092569
hour_density	0.038886	0.069558	-0.014018	0.029398	-0.049231	1.000000	0.003254	-0.003983	0.001972
month_density	0.011473	0.013423	0.009970	0.012057	-0.007067	0.003254	1.000000	-0.001772	-0.006643
pickup_density	-0.115395	-0.013035	-0.028490	-0.011519	-0.056844	-0.003983	-0.001772	1.000000	0.040906
dropoff_density	-0.018611	-0.002790	-0.120456	-0.006325	-0.092569	0.001972	-0.006643	0.040906	1.000000

Figure 11: df.corr()

In summary, after completing prepossessing our datasets, we are ready to explore some algorithms

and determine their usefulness so that we can eventually accomplish our minimization goal.

3 Methodology

Core concept is to use Gradient Descent method to estimate the function to be as close to the real output y as possible. The GD is in the form below:

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

where $\mathbf{a}$ be a point in the neighborhood of the function $F(\mathbf{x})$, and $F(\mathbf{x})$ is a differentiable multivariable function and also known as the loss function. We minimize the gradient $\nabla F(\mathbf{a})$ so that point $\mathbf{a}$ can eventually reach the global minimum. γ is defined as a learning rate that controls the step size of gradient function.

We also introduce the Root Mean Square Error(RMSE) as our measuring metrics to differentiate performance between models where Y is the true label and $\hat{Y}$ is the prediction:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}.$$

3.1 Baseline – Gradient Boosting

Our first model is the Gradient boosting, which is a commonly used machine learning technique for regression and classification problems. A short description of the algorithm is as follows:

- Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, which in this case is RMSE, and number of iterations M.
- Initialize model with a constant value $F_0(x)$, then for $m = 1$ to M:

  1.Compute the pseudo-residuals which is defined:

  $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

  2.Fit a base learner $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.

  3.Compute multiplier γ_m by solving the following optimization problem:

  $$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

  4. Update the model:
  $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

- Output $F_M(x)$.

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

3.2 Random Forest

Random Forest is also a good candidate model for the regression problem. It is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The Random Forest is an

ensemble of Decision Trees. One of the decision trees from the Random Forest is shown in Figure 12.

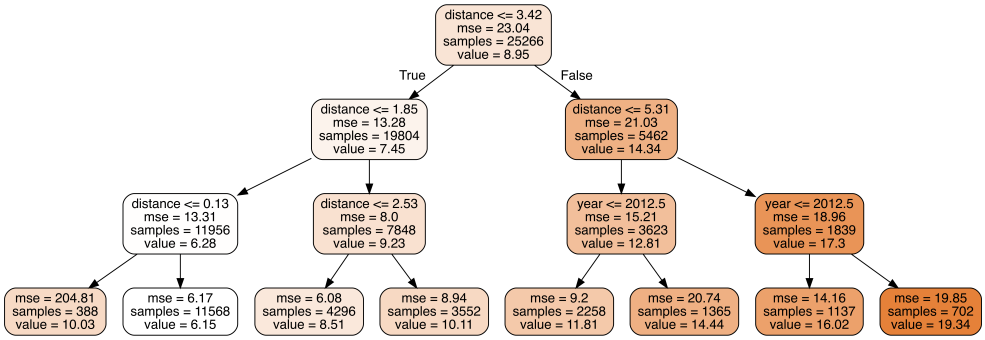


Figure 12: One of the decision trees from the random forest

3.3 Neural Network

An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Figure 13 is an example of such network that each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.

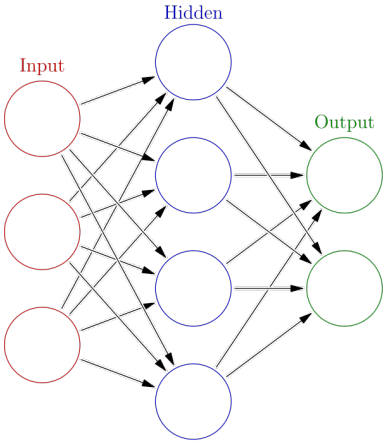
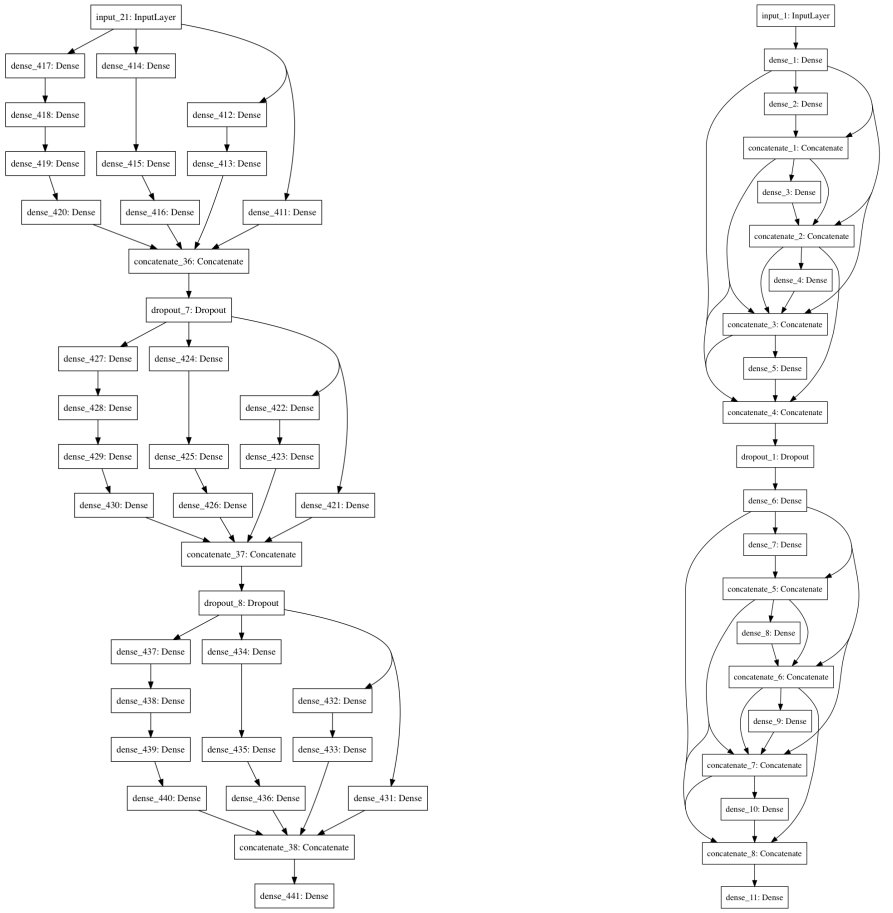


Figure 13: Neural Network

Notice that our task is essentially a regression problem, therefore, it is under the category – supervised machine learning problem, which requires our algorithm to fit a line to separate the data based on training example. Figure 14 shows are our actual models – Inception network and DenseNet.

In subfigure 14b the dense blocks are arranged in a feed forward manner. As we can see, each layer is directly connected to another layer, and the layers are applied to the input to generate features. Furthermore, the generated features are then concatenated in certain parts of the model. Subfigure 14a shows the Inception network.



(a) Inception Model (b) DenseNet Model

Figure 14: Our Neural Network Models

4 Results

The results are surprising as we originally thought that with the the sophisticated architecture and implementation, the Neural Network model supposed to outperform the other models. However, the Random Forest is actually the most well-performed model.

4.1 Gradient Boosting

We run the gradient boosting algorithm first, using the scikit-learn package in Python:

```
clf = ensemble.GradientBoostingRegressor()
```

The best result Gradient Boosting can achieve is $2.91. What we find out is that increasing max_depth as well as increasing training datasets in Gradient Boosting has little to no positive effect on the accuracy. Possible explanation to this is the distribution of the locations are too dense that more datasets can mean duplication or redundant information. On the other hand, when we chose max_depth equals to 10, the result came out worse than the default setting max_depth equals to 3.

4.2 Random Forest

Model	Feature	Data Amount	RMSE ($)
Gradient Boosting	Pickup/Dropoff Location Pickup Year/Month/Hour Distance Time Crowdedness Area Crowdedness	400,000	2.92
		200,000	3.07
		100,000	3.09
		50,000	2.91
Random Forest	Pickup/Dropoff Location Passenger Count Pickup Time Distance	400,000	4.05
		200,000	4.65
		100,000	4.81
		50,000	4.93
	Pickup/Dropoff Location Pickup Year/Month/Hour Distance Time Crowdedness Area Crowdedness	400,000	2.87
		200,000	2.90
		100,000	2.94
		50,000	2.96
DenseNet Model	Pickup/Dropoff Location Pickup Year/Month/Hour Distance Time Crowdedness Area Crowdedness	400,000	3.22
		200,000	3.28
		100,000	3.56
		50,000	4.23
Inception Model	Pickup/Dropoff Location Pickup Year/Month/Hour Distance Time Crowdedness Area Crowdedness	400,000	3.08
		200,000	3.12
		100,000	3.44
		50,000	3.97

Figure 15: Results

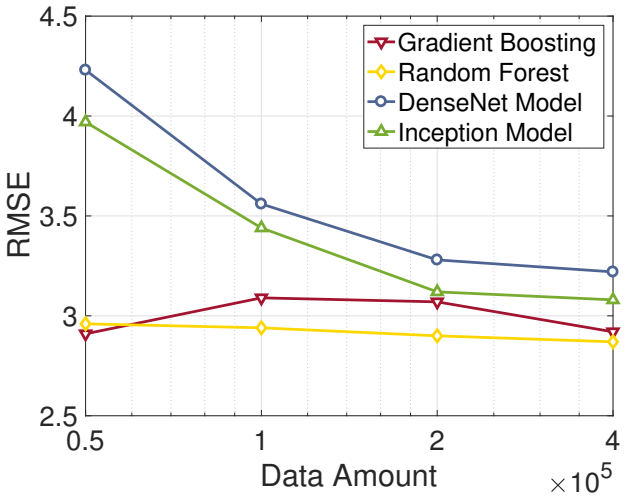
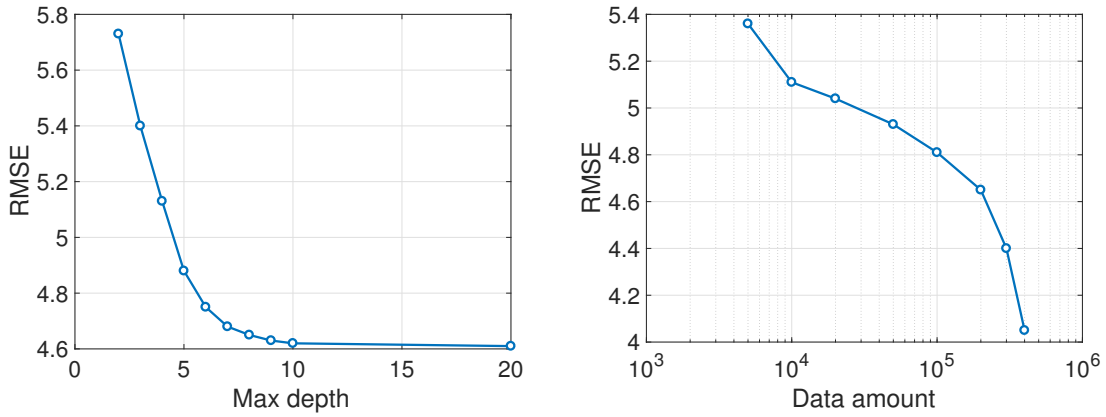


Figure 16: Results

```
regr = RandomForestRegressor(max_depth=None, random_state=0,
n_estimators=100)
```

We first use the basic features as the input for the random forest. Figure 17 shows the impact of max depth and data amount on the prediction accuracy of random forest. From Figure 17a, we notice that when the max depth of each tree in the forest is less than 5 the root-mean-squared error of prediction decreased rapidly. When the max depth reaches 10, the prediction accuracy does not change a lot. Therefore, a max tree depth of 10 is a relatively good choice for our random

forest model. Figure 17b reveals that as the data used in the training of the forest the accuracy increased continuously. We can observe similar tendency from Figure 4.

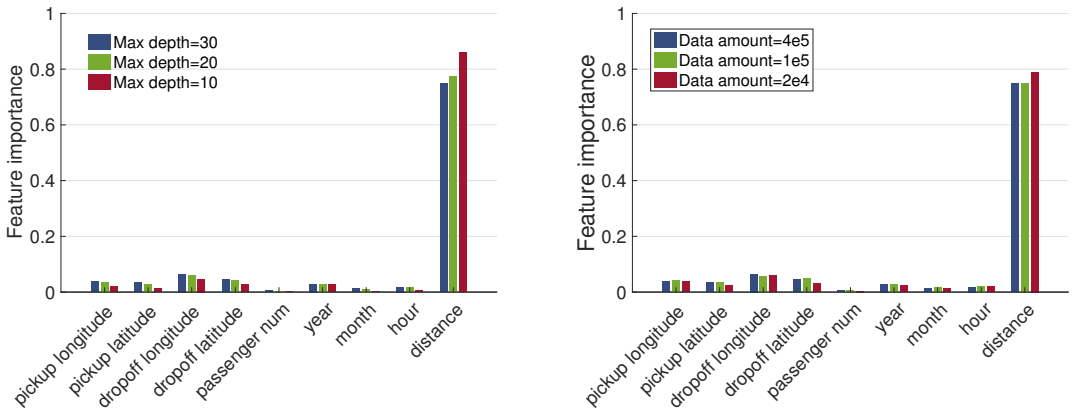


(a) Max Depth (b) Data Amount

Figure 17: Parameters

We then analyze the importance of different features in the random forest method. We compare the feature importance from different angle from Figure 18. It is obvious that distance is the most important feature while other features seem to have relatively low importance. There is an interesting finding: as the trees grow deeper the distance feature becomes less important while other features become more important. However, change in data amount does not change the feature importance consistently. Such phenomenon enlightens us that deepening the decision trees in the forest is beneficial to make use of relatively poor features.

Based on these analysis, we extract four new features from the basic dataset and delete the passenger count feature which is the least important. Figure 5 enlightens us that the traffic situation varies with hours and months. Similarly, traffic situation may also correlated to different part of the city. Therefore, we calculate the congestion degree of different time periods and locations. In conclusion, we add *hour_density, month_density, pickup_density, dropoff_density* and delete the *passenger_count* feature.



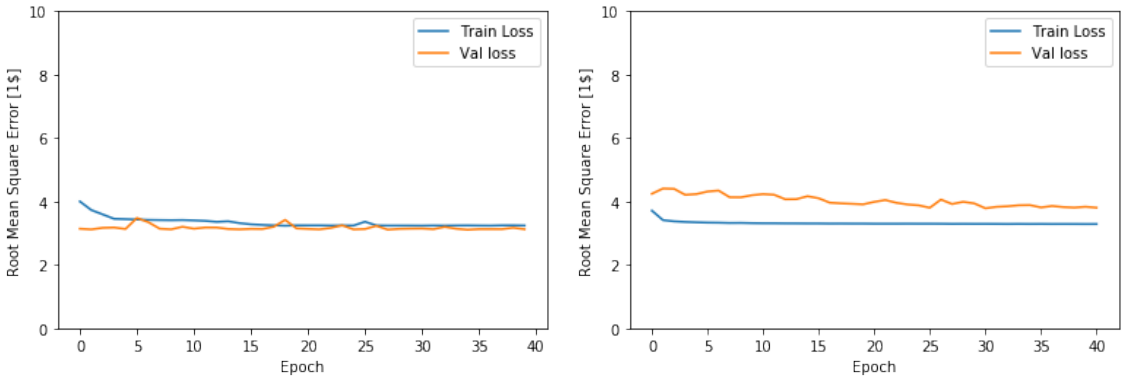
(a) Max Depth (b) Data Amount

Figure 18: Feature Importance

Random Forest achieves a RMSE of 2.87, rank the best of all models.

4.3 Neural Network

The training loss and validation loss of our two neural networks are shown below in Fig 19a and Fig 19b. From these two figure, it is easy to conclude that both Inception model and DenseNet model perform badly on reducing the training and validation loss. In other words, the loss functions do not change a lot during the whole training process. But instead, they just decrease a little bit for the first several epochs, which means that the learning effect of neural networks is not so satisfying. Therefore, we are able to explain why simple models like gradient boosting and random forest has even better prediction results. Actually, the prediction accuracy of our two complex models is merely a little bit higher than a simple network with fully-connected layers, which is proved to be more efficient. Overall, the Neural Network achieves just over $3, rank the third of all models.



(a) Inception Model

(b) DenseNet Model

Figure 19: Different performances

4.4 Discussion

As mentioned above, we are able to get four more features in the datasets after our data augmentation. As a result, our datasets are sufficient for training. All the algorithms are working as their predictions are far better than the standard deviation of the original datasets 9.73 shown in Figure 1. In fact, in the Kaggle competition, the best result from the winner is 1.38, so we are not quite far from the best solution.

The algorithm is fast enough for this task, and we find that the lack of training features is still the main factor that hold us back. We improve our result from more than $6 to under $3. For the GB, the time to execute the program is around 3 second..(need time for random forest and NN as well.)

Bottlenecks could be after this many max_depth, the result does not improve a lot any more. The deeper our model uses for max depth, the less error we can achieve, which is a good thing. However, it takes way too long to compute the prediction, and the advantage for using deep model decreases significantly. For example, initially it decreases 0.73, at depth 20, it only decreases 0.01.

As shown from Table 4.4, random forest is the best performing model the second best performing model is the gradient boosting model and the worst performing model is the neural network.

Models	Accuracy	time
Gradient Boosting	$2.91	3.61s
Random Forest	$2.87	37.9s
DenseNet	$3.22	2681s
Inception	$3.08	3214s

5 Conclusion

In this project we predict the taxi fare based on several machine learning models. As the vehicle industry is undergoing a drastic change, we believe that it will lead to a more efficient society. What we have accomplished can encourage people to use Uber for transportation more often.

Based on our discovery and simulation on the taxi fare process, we successfully turn the taxi fare problem into a optimization problem, which we fit our algorithm to minimize the RMSE so that our predictions are as close to the actual fare amount as possible. We have applied three different machine learning models to predict the taxi fare, Gradient Boosting, Random Forest and Neural Network. The Random Forest model was the best performing model with a RMSE of $2.87.

We welcome future industry leaders and developers who are interested in this project to make use of the trajectory information and continue to improve the existing web and phone taxi ride-sharing applications.

References

[1] Speech and Language Processing
Jurafsky, D., J. Martin, and A. Kehler 2000 Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition: MIT Press.

[2] Gradient boosting
https://en.wikipedia.org/wiki/Gradient_boosting

[3] Artificial neural network
https://en.wikipedia.org/wiki/Artificial_neural_network

[4] Random Forest
https://en.wikipedia.org/wiki/Random_forest

[5] Gradient descent
https://en.wikipedia.org/wiki/Gradient_descent

[6] Densely Connected Convolutional Networks
Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. arXiv preprint arXiv:1608.06993, 2016a.

[7] Going Deeper with Convolutions
C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In CVPR, 2015