GEN 511 - Machine Learning
New York City Taxi Fare Prediction

Rahul Murali Shankar (IMT2017033)
Phani Tirthala (IMT2017031)
Brahma Kulkarni (IMT2017011)

International Institute of Information Technology, Bangalore

Abstract. New York City is a bustling place with millions of taxi cabs travelling across the city in a single day. Even amidst such a vast set of data, there must be some way to predict the amount that a cab will cost, given certain parameters like pickup and drop-off locations, date and time of the day and other relevant parameters. This particular dataset contains eight features, on which we must perform various operations to make the data more suitable to apply machine learning techniques.

1 Introduction

1.1 Dataset

In the given problem, we are tasked with predicting the Fare Amount for a taxi ride in New York City given the following features:

- Key
- Pickup Datetime
- Pickup Latitude
- Pickup Longitude
- Dropoff Latitude
- Dropoff Longitude
- Passenger Count
- Fare Amount - Target Variable

This is a regression problem since the target variable is non boolean and we have to predict a finite quantity.

1.2 Cost Function

As given in the problem description, the cost function that we have to optimize is the Root Mean Square Error. The aim of the problem is to minimize the RMSE on the test dataset as much as possible.

1.3 Process

The process of predicting fare amount will have the following steps:

1. Exploratory Data Analysis
2. Data Pre-Processing
3. Feature Engineering
4. Model Selection

The following sections will explain the process followed in greater detail.

2 Exploratory Data Analysis

This section involves plotting graphs between various features and exploring relationships between them, so as to be able to pre-process data in an efficient manner. The size of the training dataset is approximately 44.5 million rows, and plotting graphs for all the rows would be highly impractical, so the data has been divided into chunks of 500,000 rows for processing and only the first chunk has been used for plotting graphs. Also, the "Key" attribute is not an attribute that the output has a dependency on, so we can remove it for the purpose of training the model.

The following graphs were plotted to analyze the data.

2.1 Frequencies of each feature - Fig 1-3 Graphs of each feature against fare amount - Fig 4-6 Boxplot - Fig 7

We plotted the graphs of the frequency of each feature as a histogram. From these, we observed that some features had values that are not possible, like latitude and longitude being in the 1000-2000s range, the passenger count being 200 and so on.

We also plotted the graphs of each feature against the target variable i.e. fare amount as a scatter plot, to observe and derive some sort of dependency between them.

The last graph we plotted was the boxplot for fare amount, a graph that is used for outlier detection, in order to clean the data of any outliers.

2.2 Non-graph analysis

Upon checking the number of null values in the dataset, it has been found that the first 4-5 chunks have about 18 null values each out of 500,000. We have thus concluded that approximately the same pattern will exist throughout the dataset.

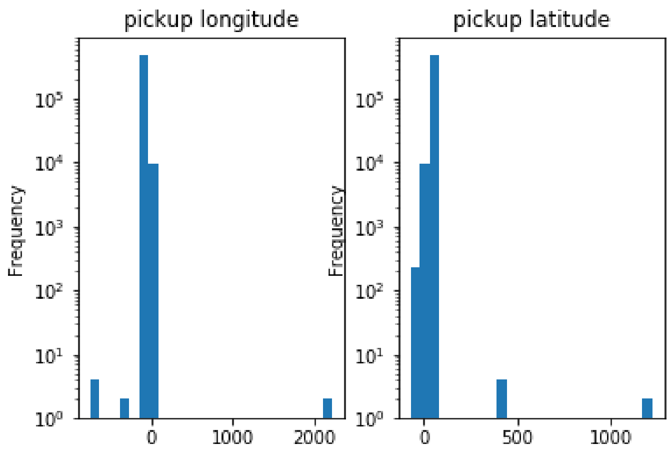

Fig. 1. Pickup Longitude and Latitude Frequency

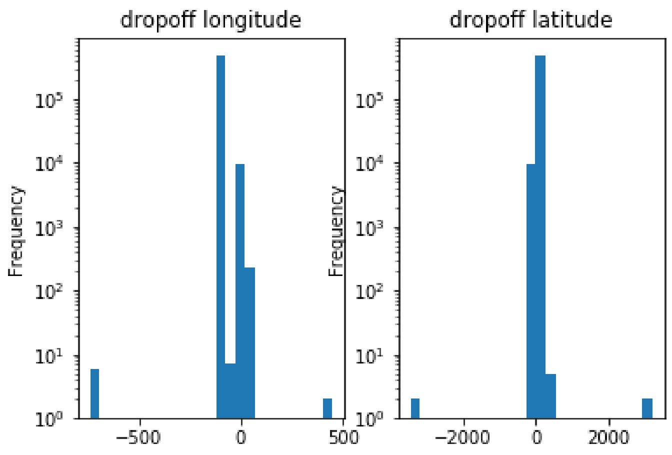

Fig. 2. Dropoff Longitude and Latitude Frequency

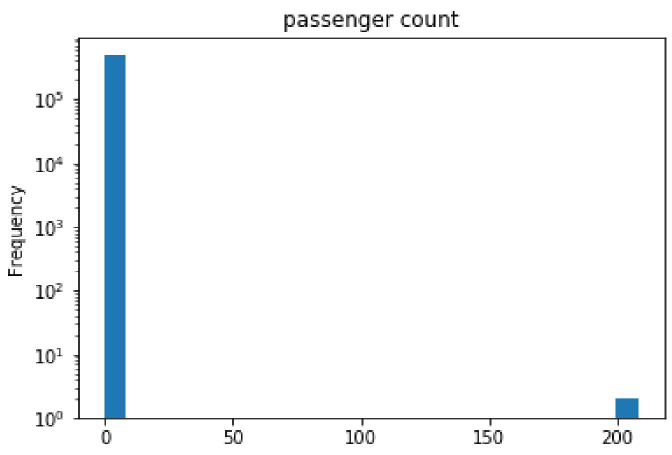

Fig. 3. Dropoff Longitude Frequency

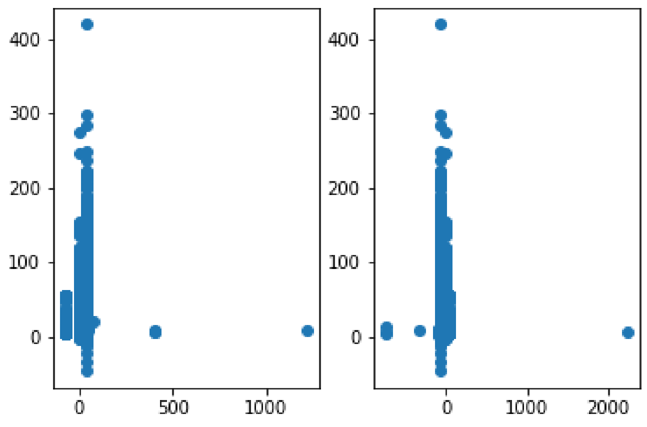

Fig. 4. Pickup Longitude and Latitude vs Fare Amount

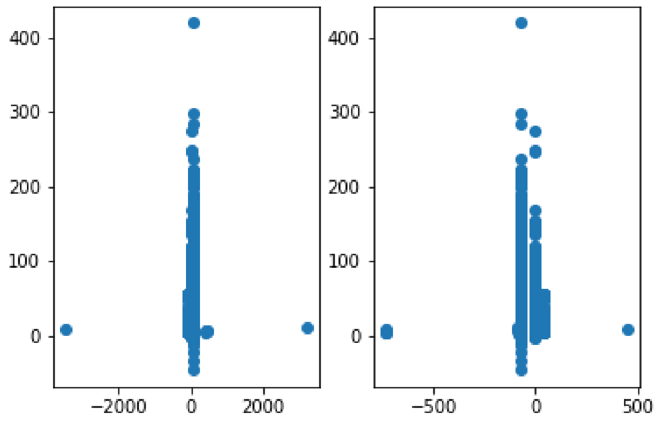

Fig. 5. Dropoff Longitude and Latitude vs Fare Amount

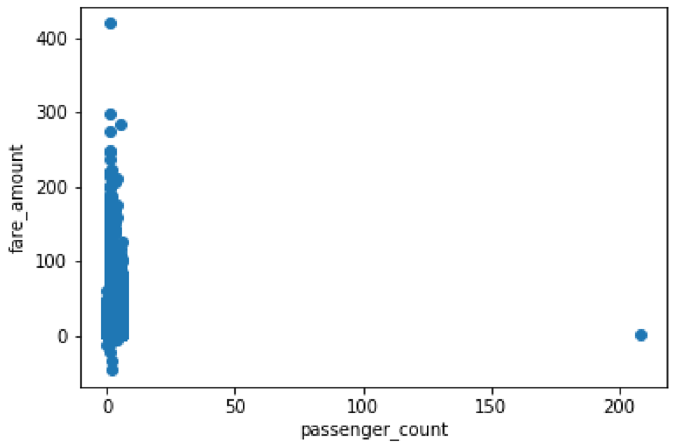

Fig. 6. Passenger Count vs Fare Amount

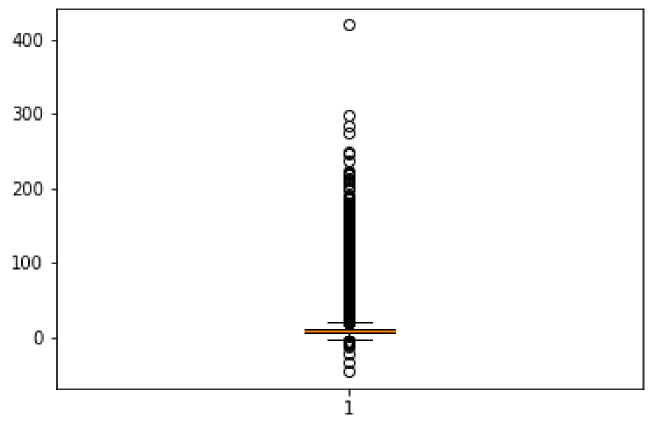

Fig. 7. Fare Amount Boxplot

Also, we are given the pickup datetime for each data point, which can be used to predict the fare amount after some pre-processing. The time and date have a cyclic dependence on fare amount, and the appropriate processing for it will be discussed in the next section.

3 Data Pre-Processing

The ”Key” attribute is not part of the feature list used to train the model, since it is just a feature to identify a particular row, and the fare amount will not have any dependency on it.

3.1 Null Values

As mentioned in Section 2.2, there are only 18 null values per chunk in the dataset, which gives us a total of about 800 null values out of 44.5 million. This means that about 0.0017% of the data is null values. Thus, we can safely remove the rows with null values since it will not affect the cardinality of the dataset as far as the learning process is concerned.

3.2 Invalid Values

The pickup latitude and longitude, and the drop-off latitude and longitude are the features that contain invalid values in the dataset.

As we know, latitude can take values between -90 and 90 while longitude can take values between -180 and 180. From Figures 1,2 and 4,5 we can see that some of the values lie as far as -3000 to 3000. In addition to this, the dataset is supposed to be for New York City, so we also have to restrict the co-ordinates to NYC. The co-ordinates of NYC are 40.495992 to 40.915568 latitude and -73.699215 to -74.257159 longitude.

Furthermore, passenger count lies mostly close to zero, but there are some data points which contain passenger count around 200, which is obviously impossible.

3.3 Outlier Detection

Upon running data.describe(), we noticed that the target variable, fare amount, contains some values that are impossible practically. The fare amount, in USD, has values in the range of \$200 to \$300, which cannot happen within New York City (we can see this from the graphs containing fare amount too).

This is a clear indication of outliers in the dataset which also qualify as invalid data. To deal with outliers, we plotted the boxplot for fare amount, which is a type of graph used for outlier detection. Figure 7 contains the boxplot. After this, we can calculate the z-score of each data point in the fare amount column, which is given by:

$$\text{z-score} = \frac{X-\mu}{\sigma}$$

where X is the data point, μ is the mean and σ is the standard deviation.

This basically tells us the number of standard deviations away from the mean a data point is. It is a standard metric that any data point with z-score > 3 or z-score < -3 can be considered an outlier.

3.4 Conclusion

We could remove these rows, since they constitute less than 3% of the dataset in total and would not affect the model training in any way. But there is a problem that we run into while following such an approach. The test data also contains such invalid values, and we cannot remove rows from the test dataset. Thus, if we train the model with the training dataset containing nothing but valid values, the model would not be able to predict the values for the test dataset accurately. Thus, we can come up with a workaround for this, that will be discussed in the next section.

4 Feature Engineering

This section details how to modify features in the dataset based on the exploratory data analysis done in order to create more useful features for the training model.

4.1 Cyclic Dependence of Datetime

As we know, the date and time is something that occurs repeatedly over the course of time. For example, the same combination of hour, minute and second will repeat once every day. We can thus split the ”datetime” attribute into it’s components i.e. year, month, day, hour, minute, second.

To capture the cyclic dependence of the combinations month + day and hour + minute + second, we created two attributes called ”seconds past midnight” and ”days past January 1”, and plotted it’s dependence against fare amount. A regular plot would not accurately capture the dependence, so we plotted the sin and cos of each of them, as shown in Fig 8 and 9.

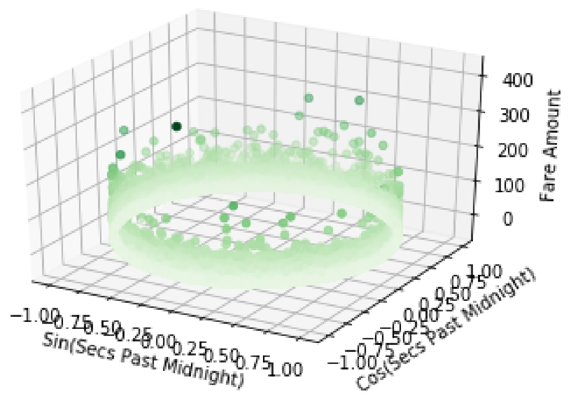

Fig. 8. Sin SPM and Cos SPM vs Fare Amount

4.2 Invalid Column

As mentioned in Section 3.2, there are some values in the training dataset that are invalid. Since they are also present in the test dataset, we cannot simply discard them from the train dataset, but rather we must learn from them.

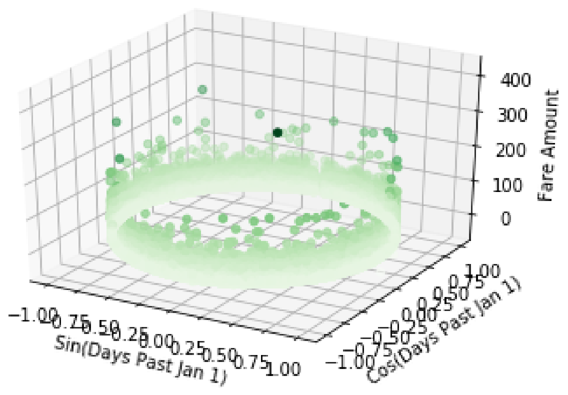

Fig. 9. Sin DPJ and Cos DPJ vs Fare Amount

Now, how do we learn from invalid values while still being able to distinguish them as invalid? For this, we created a new feature called 'Invalid', which would be marked 1 if any one feature in that row was invalid, and 0 if every feature in that row contained valid values.

With this new feature, we can make predictions on invalid values in the test dataset as well, so our problem is resolved.

5 Model Selection

We tested many models on Upon running the XGBoost Regressor on the test dataset, we got an RMSE of 6.03 on the leaderboard.

the dataset and picked the one that gave us the highest accuracy on the training dataset, starting from the basic Linear Regression to the one we are currently using.

For each model, we mention a quantity called average RMSE. This means that we are running the model on many chunks in the dataset, and for each chunk we get an RMSE, after which we take a sample mean. We know that the average RMSE is not simply the sample mean, but is a good approximation of it.

5.1 Linear Regression

We trained the dataset on a basic Linear Regression model, and got an average RMSE of 7.62 on the training dataset. This model was bound to give a relatively high RMSE, which is why we did not train it on the test dataset.

5.2 Random Forest Regressor

On a Random Forest model, we got an average RMSE of 4.08 on the train dataset. We briefly considered running a grid search on this model to find out the best hyperparameters, but once again, we got better results on other models.

5.3 Adaboost Regressor

We tried running this model on a dataset with a different set of features. After having performed random forest on the dataset, we tried to perform some more feature engineering such as:

- Adding a feature called ”Distance to JFK Airport”, the reason for which was that there is a standardized cab fare for JFK Airport so we needed to learn that as a feature.
- Adding a feature called ”Day of the Week”, to check for the correspondence with fare amount.

Running Adaboost Regressor on this feature set, we got an average RMSE of 8.88, far worse than other models, which is why we decided to scrap that feature set and go back to the old one.

5.4 XGBoost Regressor

The XGBoost Classifier gave us an average RMSE of 3.85, which is lower than any other model we used. After this, we tried a method to increase the accuracy of the model, called grid search.

Grid Search This is a method to find the hyperparameters like learning rates and number of epochs that give the highest accuracy on training. There is a special python package that does this, but the process took a lot of time. For only a single chunk in the dataset, consisting of 500,000 rows, the process took well over an hour and still hadn’t stopped, so we decided to not use it since it would be highly impractical for the entire dataset.

5.5 Conclusion

After looking up other models, we discovered a model called Light GBM (Light Gradient Boosting Machine) or LGBM, which would have performed extremely well on large datasets. Unfortunately, it was not a straightforward implementation in that we could simply call the package and run the model, instead we had to change a lot of parameters. We were unable to implement the required changes in time so we decided not to implement the model.

6 Result

Upon running the XGBoost Regressor on the test dataset, we got an RMSE of 6.03 on the leaderboard.

6.1 Link to Models (Pickle files)

The link to the model.sav files used while running the code is given below:

https://drive.google.com/drive/folders/17CwQqo2fJdL7YUP2ycm9G9taoEe2XkDG