Project Report

Cab Fare Prediction using Machine Learning Algorithms

Instructions to run the code file: -

- Extract the 'Cab_Fare_Prediction.zip' file.
- Open Jupyter notebook in the browser and upload all three files named 'Clean_train_data+EDA.ipynb' , 'Clean_test_data.ipynb' and 'Fare_Prediction.ipynb' using the upload button provided on the home screen of Jupyter Notebook.
- After uploading is done, the notebooks can be accessed from the list on the home screen.

Problem Statement: -

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected the historical data from your pilot project and now have a requirement to apply analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

Solution: -

This project was done in three parts, and has 3 notebook (.ipynb) files attached with it.
The first part deals with cleaning of the train data and exploratory data analysis (EDA).
The second part deals with cleaning of the testing data and third part deals with prediction of fares in the test data. Each part will be explained in detail in this report.

1.1 Cleaning of training data and Exploratory Data Analysis (Please refer to the file named "Clean_train_data+EDA.ipynb")

The training data contains 16062 observations and 7 variables which are:

- **pickup_datetime** - timestamp value indicating when the cab ride started.
- **pickup_longitude** - float for longitude coordinate of where the cab ride started.
- **pickup_latitude** - float for latitude coordinate of where the cab ride started.
- **dropoff_longitude** - float for longitude coordinate of where the cab ride ended.
- **dropoff_latitude** - float for latitude coordinate of where the cab ride ended.
- **passenger_count** - an integer indicating the number of passengers in the cab ride.

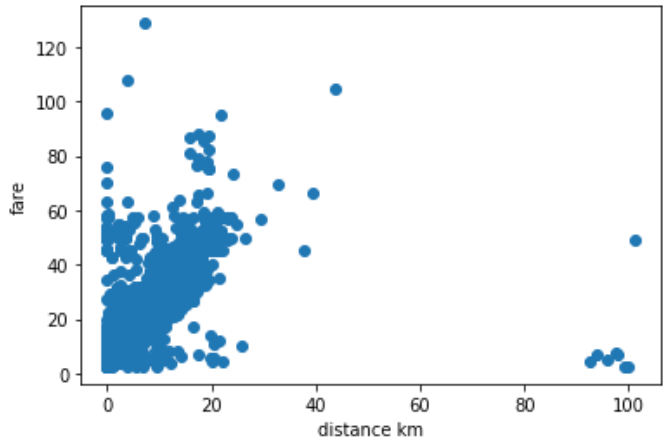
The entire procedure of cleaning the train data will be further explained step by step:

- First, open the file in MS Excel and observe the data. The train data had many missing values, some values were 0 for number of passengers and some values were more than

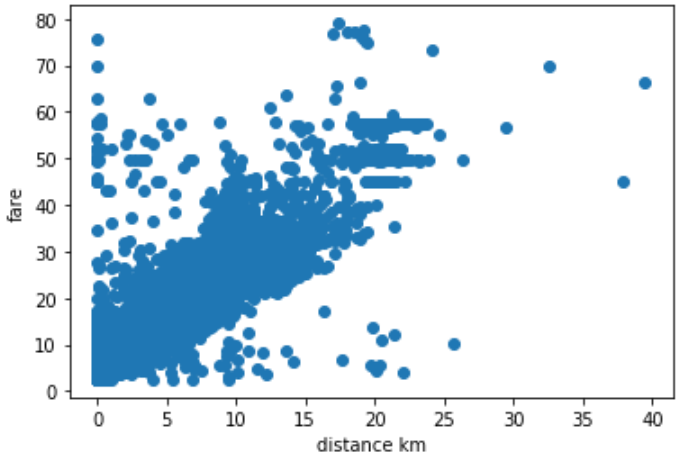
6. Also, it was observed that some fare amount had negative values. Here, the missing values were replaced with 0 using special paste function in excel. The file was saved and rest of the data cleaning was done in python.

- In a new jupyter notebook all the necessary libraries were imported. Libraries used in this section are:
    - os: to change the working directory
    - pandas: to create dataframes and perform dataframe related operations.
    - numpy: to carry out complex mathematical and matrix operations.
    - Fancyimpute: for imputing missing values.
    - Matplotlib: for visualizations.
    - Seaborn: for visualizations.

- The train data was imported and summary of the dataset was checked using train.describe(). Here, it was observed that maximum fare value was 54343 which is not possible and clearly an outlier. Then pickup longitude had max value 40.766 as the longitude must be in the range of -70s. The same applies for dropoff longitude. For pickup latitude max value was 401.08, which is again an outlier. Dropoff latitude seemed fine. Passenger count had max value of 5345 which is not possible.
- After getting overall idea of the data, all 0s were replaced with null value so that they can be imputed or dropped easily.
- We, check the number of missing values in each column.
- 'fare amount' is the dependent variable and it contains least number of missing values. Also, being dependent variable, errors in the imputation may affect our model and hence, we remove those observations with missing values.
- Now, we find all the observations with passenger count that are either less than 1 or greater than 6 (outliers) and drop such observations.
- Then, as discussed earlier, we remove the outlier values from the GPS co-ordinate (pickup and dropoff longitude and latitude) data based on minimum and maximum values of co-ordinate within the New York City.
- Then pickup datetime column was separated from the train data as KNN impute only allows numeric variables when we apply it on a dataset.
- Here, we used KNN for imputation because mean and median imputation will simply replace all NaNs with mean and median respectively which in turn will make our models biased in the modelling phase. Also, different values for k were used and best one that approximated the known values was found to be k=10 as mentioned in the notebook file.
- After that, pickup datetime was included again in the dataset using concat function.
- Also, it was observed that some fare values were very high and some were very low, almost $0. According to information found on web, the base rate of hiring cab in NYC is $2.5 so all values below this should be removed. Also, within NYC, no cab ride should cost more than $150. So we remove all such entries.
- Next, we calculate Haversine distance based on pickup and dropoff longitude and latitude values. Haversine distance basically gives the distance between two points on a spherical surface. This will give us approximate vector distance from point A to B.



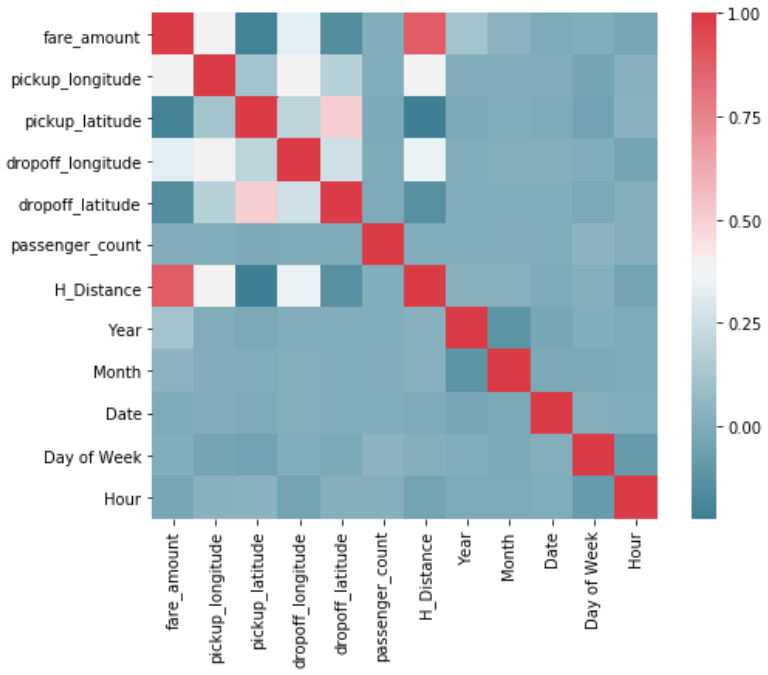
- Now, it is obvious that for a cab ride, the fare amount is directly proportional to the distance travelled. So, we plot a graph of distance vs. fare amount.



- As, we can see there are lot of outliers i.e. the distance is above 80km and fare is less than $20 which cannot be possible. Also, some entries have distance less than 20km but fare is more than $80 which is again not possible. Also, such entries were less and thus they were removed.
- From this graph we can see there is a linear relationship between the distance and fare amount.

1.2 Exploratory Data Analysis

After cleaning the data, we extracted features like Year, Month, Date, Day and Hour from pickup datetime feature. Then, we did some exploratory data analysis by plotting graphs to determine which variables other than distance had an effect on our fare amount.



We plotted a correlation plot using heatmap to understand the correlation among the variables. It was observed that distance was highly correlated with fare amount as compared to any other variable. Also, there was no such strong correlation among the independent variables and thus, no variable was removed as there was no redundancy.

Following were the questions that were aimed to be addressed by this data exploration:

i. How no. of passengers affects the fare?
ii. How year affect the fare?
iii. How month affect the fare?
iv. How day of the month affects the fare?
v. How day of the week affects the fare?
vi. How particular hour of day affects the fare?

Following are their answers that were obtained through visualizations:

i. Single or two passenger rides were ones with maximum fare amount. It could be that such rides covered longer distances.
ii. The fares seem consistent over the years.
iii. Fare amounts from July to December seems to be higher than other months.
iv. The prices seem more or less consistent throughout the month.
v. The highest frequency is on Friday and Saturday (weekends). While the prices aren't affected that much although there are some datapoints with high fares which maybe due to people travelling long distances during weekdays.
vi. Frequency during midnight hours is less and at 5am its the least and it is most between 6pm to 8pm. Now during midnights when frequency is the least, yet the prices are comparable to those during working hours. This maybe due to the fact that fares are higher after midnight and most cab hires are going to the airport which has fixed rates.

Finally, we dropped the pickup_datetime variable as all its features were extracted earlier and there was no use keeping it in our dataset.

2. Cleaning Test Data (Please refer to the file named "Clean_test_data.ipynb")

- Explore the text file in excel. It was observed that there were no missing values in the dataset. The dataset was further explored in the python notebook.
- Import important libraries just as in the previous section.
- Check the summary of numeric variables. It was observed that all the values were sensible and there were no outliers.
- Then we checked for missing values again and there were no missing values in the dataset.
- Next, features were extracted from pickup datetime variable and features like Year, Month, Date, Day and Hour were appended to the test dataset.
- Then, Haversine Distance was calculated for the given co-ordinates in the test dataset, just like we did in the train dataset.
- Next, we looked for 0 values in the H_Distance variable and found that 86 observations had distance 0. This may indicate a round trip, but we cannot impute it appropriately as we don't have fare amount to calculate the best estimate of distance. So, we drop these observations.
- Finally, we drop pickup_datetime variable.

3. Fare Prediction (Please refer to the file named "Fare_Prediction.ipynb")

For this section, we first created a baseline model using multiple linear regression model and then went on to use Lasso Regression, Decision Tree, Random Forest, LightGBM and XGBoost. Detailed explanation for the process is given below.

- Import the important libraries.
- Load the cleaned train and test data.
- Now, we need to split our training data into train and validation data. We used 'train_test_split' from sklearn's model selection. The train data was given 80% of the original training data and validation data was given 20% of the original training data.
- Independent variables and dependent variable was separated into different objects for both train and validation data.
- Next, for calculating error metrics for our models, a function was written wherein all the error metrics like mean squared log error (MSLE), r2, mean absolute error (MAE), root mean square error (RMSE), Akaike information criteria (AIC) etc.

Note: Even though many error metrics were calculated, special attention was given to RMSE values. The reason for choosing RMSE for determining error in this project is that RMSE penalizes highly varying predicted values from the original values. In other words, if the residue is higher, higher would be the RMSE. On the other hand, MAE will give us mean of all the residues i.e. it will give us exact mean of difference between original and predicted values without penalizing highly varying predicted values. Thus, if we use RMSE, we will get a good idea as how well our model is predicting the fares.

Implementation of Models: -

i. **Multiple Linear Regression: -**

Created a baseline model using MLR. The baseline RMSE we got is 4.2345 for train data and 4.1297 for validation data. Our aim is to build a model that has better RMSE score than this. Also, the variance between these two RMSEs matter and should be minimum, because if these are far apart, it would indicate overfitting of the model which we must avoid.

ii. **Lasso Regression: -**

A Lasso Regression model was implemented and was iterated for different values of alpha, but it was found that using lasso regression is not feasible as alpha=1e-5 gives us least RMSE of 4.2344 which is almost as good as multiple linear regression model's RMSE according to above analysis. But if we use such a small value of alpha for lasso, it will reduce lasso to linear regression model (as alpha=0 for lasso gives linear regression).

Thus, we won't use Lasso for our further analysis.

iii. **Decision Tree Model: -**

We also tried non-linear models like a Decision Tree and Random Forest. For Decision Tree, we observed that RMSE value for prediction on training data goes down to 0 when max depth of the decision tree is more than 30. This means that our model with max depth 30 and above will exactly predict the training data i.e. it will overfit. But, when we iterate our model on validation data, we see that it does not perform that well for max depth of 30. The RMSE is too large.

The minimum RMSE we obtain for prediction on validation data is for max depth 6 which is 4.3400 which is slightly more than previous two models. Thus, we will not use this model.

iv. Random Forest: -

We tried decision trees, as it is based on ensemble method and may avoid overfitting problem as seen in decision tree. Also, we will see whether it gives better result than the decision tree.

Although, the RMSE for train and validation data (1.417 and 4.0574) are the least of all above models. But, it is observed that RMSEs of train and validation data are way off. The variance is more than 2. This clearly indicates overfitting of the model and thus, we shall not use it.

v. Light Gradient Boosting: -

We tried LightGBM model to see whether it performs better. LGBM has trees of fixed length and are shorter than the trees in decision tree and random forest. It is also based on ensemble method like random forest, but the difference is that all the trees in LGBM do not have equal say on the output. The say (or vote) of each tree depends upon the error of previous tree and how that current tree reduces the error.

In the model, most of the default parameters were set and we implemented our model.

The results were good. Light gradient boosting model seems to perform better than all previous models with least RMSEs (2.5426 & 3.7489) in both training and validation and variance between train and validation is 1.2. Also, the AIC (Akaike Information Criteria) score was calculated for all above models and was found to be minimum for LGBM.

Note: AIC deals with goodness tradeoff between goodness of fit and simplicity of the model. In other words, it deals with the risk of both overfitting and underfitting of the model. AIC can be used as basis for comparing models to determine their goodness of fit. Model with a lower AIC (along with lower RMSE or MAE) should be selected.

But before we fix this model for our test data, lets try XGBoost algorithm to see if it performs better than this.

vi. Xtreme Gradient Boosting: -

This model is similar to LGBM. Infact LGBM improves upon XGBoost. On implementation, we get RMSEs (3.0852 & 4.2533) on training and validation data. As it is seen, it performed worse than previous models and thus, we won't use it.

Out of all the models we have implemented, LGBoost performed the best. LGBoost provides with RMSE of 3.7489 for validation data which is better than any other model implemented. Here RMSE of 3.7489 mean that our model will predict the fare with an error of $ 3.7, more or less than the actual fare amount.

One of the ways in which we can improve the accuracy of our model is by feeding it more data. The more the data we give it, more the patterns our model will get to train on.

Thus, we used Light Gradient Boost for prediction of our test data.

The fare amount in the test dataset was predicted and predicted values were appended to the test dataset. The complete file with the predicted fares was exported as 'Final Predictions.csv' and is included in the 'Python' folder in 'Cab_Fare_Prediction.zip'.