MACHINE LEARNING NANODEGREE CAPSTONE PROJECT REPORT

PREDICTING NEW YORK CITY TAXI TRIP DURATION

AKSHAY UPPAL

1. DEFINITION

1.1 PROJECT OVERVIEW

With companies like Uber, Lyft and various other cab and cab sharing services, there has been a huge increase in the number of cab rides over the past few years. Cabs have always been a part of our lives in every major city where owning a car is more of a nuisance. As a part of my first project I wanted to touch upon something that impacts everyone.

All the mentioned taxi services must be using some electronic system for dispatching to cater to the growing demand of cabs. This brings to light the requirement of predicting accurately how long each ride takes, making it possible for the system to assign the pickup of next passenger to the right driver. The research paper that intrigued me and made me pursue this is "A Simple Baseline for Travel Time Estimation using Large-Scale Trip Data - Hongjian Wang , Zhenhui Li , Yu-Hsuan Kuo , Dan Kifer" and on some research I found a very relevant competition on Kaggle (A huge online platform for ML and data science related competitions), to predict the duration of a cab ride in New York, one of the busiest cities in the world and decided to use that as my capstone project.

The dataset used in the competition officially can be found here, however I have used an adaptation of the same dataset "New York City Taxi with OSRM" provided by Kaggle in the playground contest as my primary dataset. This dataset has additional files which has the information of the fastest route for each data point which is very useful feature to work with. Apart from the mentioned primary dataset I have used "NYC 2016 Holidays" to create a list of working days and rest days and the dataset "KNYC Metars 2016" to extract the visibility and temperature based features for 2016 in NY City.

1.2 PROBLEM STATEMENT

The dataset being used for the competition 'New York City Taxi Trip Duration' consist of the trip details for about 1.4 million taxi rides between the years 2009 and 2015 in New York city.

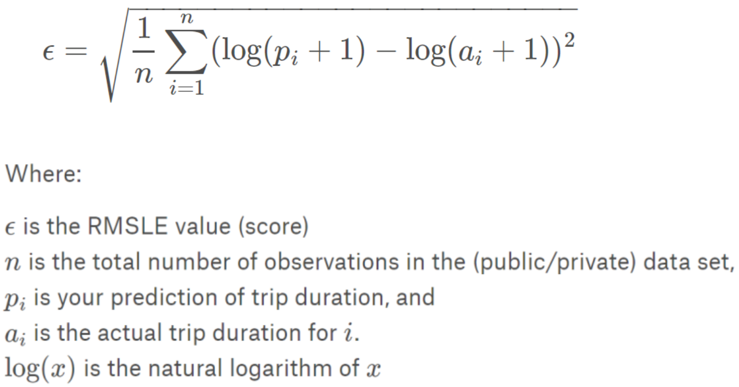
The main aim of the challenge is to predict the duration of a taxi ride, which makes it a supervised regression problem to begin with. I will try to achieve as much as I can to a comparable score on Kaggle leaderboard for this contest based on the Root Mean Squared Log Error, which is defined as the criteria.

"It is good to know that the competition dataset is based on the 2016 NYC Yellow Cab trip record data made available in Big Query on Google Cloud Platform. The data was originally published by the NYC Taxi and Limousine Commission (TLC)." The data contains information about the starting and ending positions (coordinates) of the ride, the date and time information, the passenger count which are to be used as inputs in order to predict the duration of the taxi trip. I perform a 80-20 split on the 1.45 million entries present in the training data to obtain training data and the validation data.

My approach to the problem is multifold in nature, the first step being exploratory data analysis (EDA) where I explore the data to get an insight of the features and inputs existing which will lead to some visualizations for more information. Post the EDA comes the step of feature engineering, this is the step where I use the secondary datasets and transform the data present in all datasets to create a relevant feature space for my model. This also involves transforming the features to make it compatible with the chosen model. Once the final feature space is obtained I will model a regressor (XGBoost here) to generate a score and a submission file for the competition. This first draft will now undergo fine tuning and hyper-parameter tuning in order to further reduce the error and create my final submission file.

1.3 EVALUATION METRICS

The evaluation metrics defined by Kaggle is the Root Mean Squared Log Error which is defined here as :



$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

Where:

ϵ is the RMSLE value (score)
n is the total number of observations in the (public/private) data set,
p_i is your prediction of trip duration, and
a_i is the actual trip duration for i.
$\log(x)$ is the natural logarithm of x

Being the official metric RMSLE was chosen by me as well, the RMSLE is very similar to the most widely used root mean squared error, only the actual values and predicted values pass through the log function first, the 1 is added to avoid log(0). The log punishes the error for smaller values way more than larger values, example log(100)- log(97) gives a way smaller value when compared to log(10)-log(7). This disparity between larger and smaller values makes more sense when we are trying to predict the trip durations as well, a delta of 3 min is way costlier on a short trip duration than a long one. Therefore, instead of a metric like RMSE where the delta has same priority irrespective of the magnitude of predicted and actual values RMSLE provides us with a relative error.

2. ANALYSIS

2.1 DATA EXPLORATION

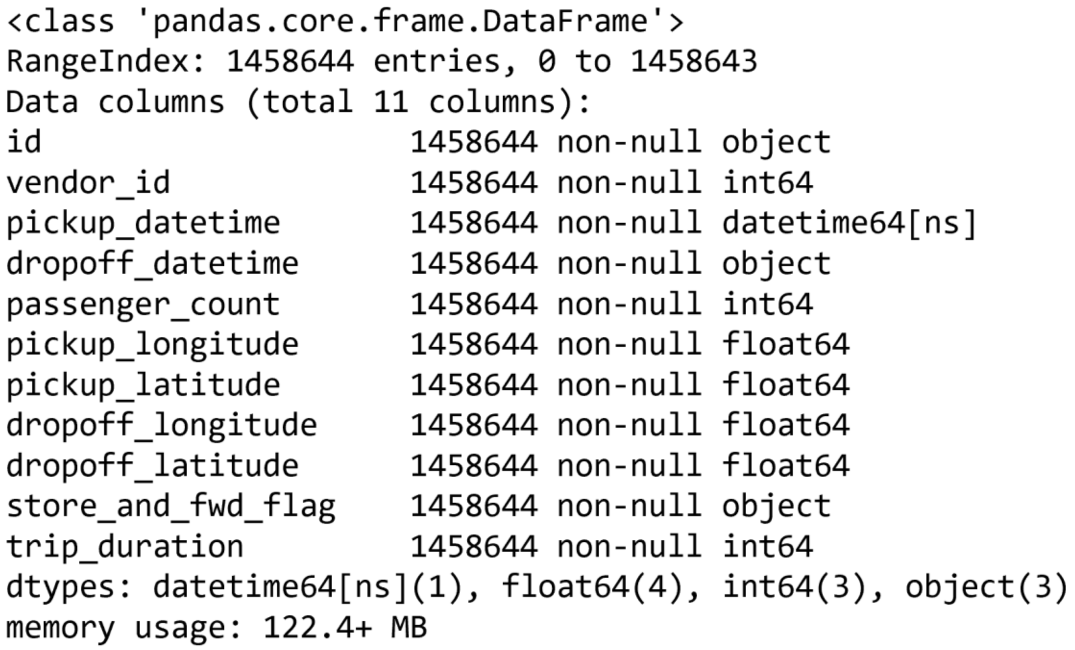
The primary dataset has a train.csv and test.csv which we will be working with, the resepectively have the training and testing data for the competition. Both the files have an identical structure and the train.csv contains 1,458,644 data entries whereas the test.csv has 625,134 entries. The features represented by the dataset are as follows:

- id - a unique identifier for each trip

- vendor_id - a code indicating the provider associated with the trip record
- pickup_datetime - date and time when the meter was engaged
- dropoff_datetime - date and time when the meter was disengaged
- passenger_count - the number of passengers in the vehicle (driver entered value)
- pickup_longitude - the longitude where the meter was engaged
- pickup_latitude - the latitude where the meter was engaged
- dropoff_longitude - the longitude where the meter was disengaged
- dropoff_latitude - the latitude where the meter was disengaged
- store_and_fwd_flag - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip
- trip_duration - duration of the trip in seconds (Target Value)

*above points directly from the competition page

The tes.csv is identical with the exception of the trip_duration and dropoff_datetime column as that is our target value to predict. The information of the data reveals its type.



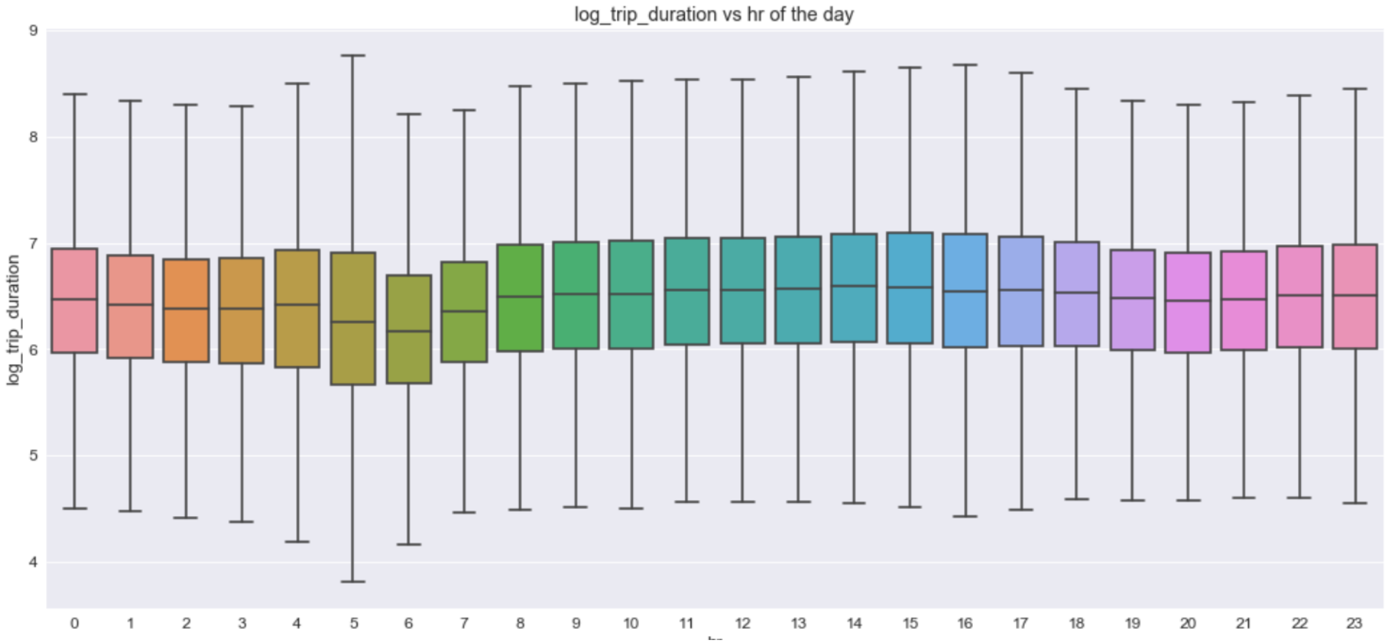
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1458644 entries, 0 to 1458643
Data columns (total 11 columns):
id                    1458644 non-null object
vendor_id             1458644 non-null int64
pickup_datetime       1458644 non-null datetime64[ns]
dropoff_datetime      1458644 non-null object
passenger_count       1458644 non-null int64
pickup_longitude      1458644 non-null float64
pickup_latitude       1458644 non-null float64
dropoff_longitude     1458644 non-null float64
dropoff_latitude      1458644 non-null float64
store_and_fwd_flag    1458644 non-null object
trip_duration         1458644 non-null int64
dtypes: datetime64[ns](1), float64(4), int64(3), object(3)
memory usage: 122.4+ MB
```

On further study of the data I found out that the value type of "vendor_id" contains only two different values and there is a 47-53% split among the data between the values '1' and '2'. There exists a boolean variable "store_and_fwd" whose value is true for 8045 values i.e. only 0.0056%. I also found that the longitude and latitude values have a very less standard deviation and lie around the mean values (-73.97, 40.75). While checking for potential outliers for the latitude range I found that there are 82 samples have lower values than -74(NYC latitude range), as the number of such samples are low I have ignored them as the XGBoost can take care of them.

2.2 DATA VISUALIZATION

The first thing I was curious was the trend of the duration of trip vs the hour of the day, I wanted to observe the reasoning behind any existing pattern and establish a dependency of trip duration and hour of the day.

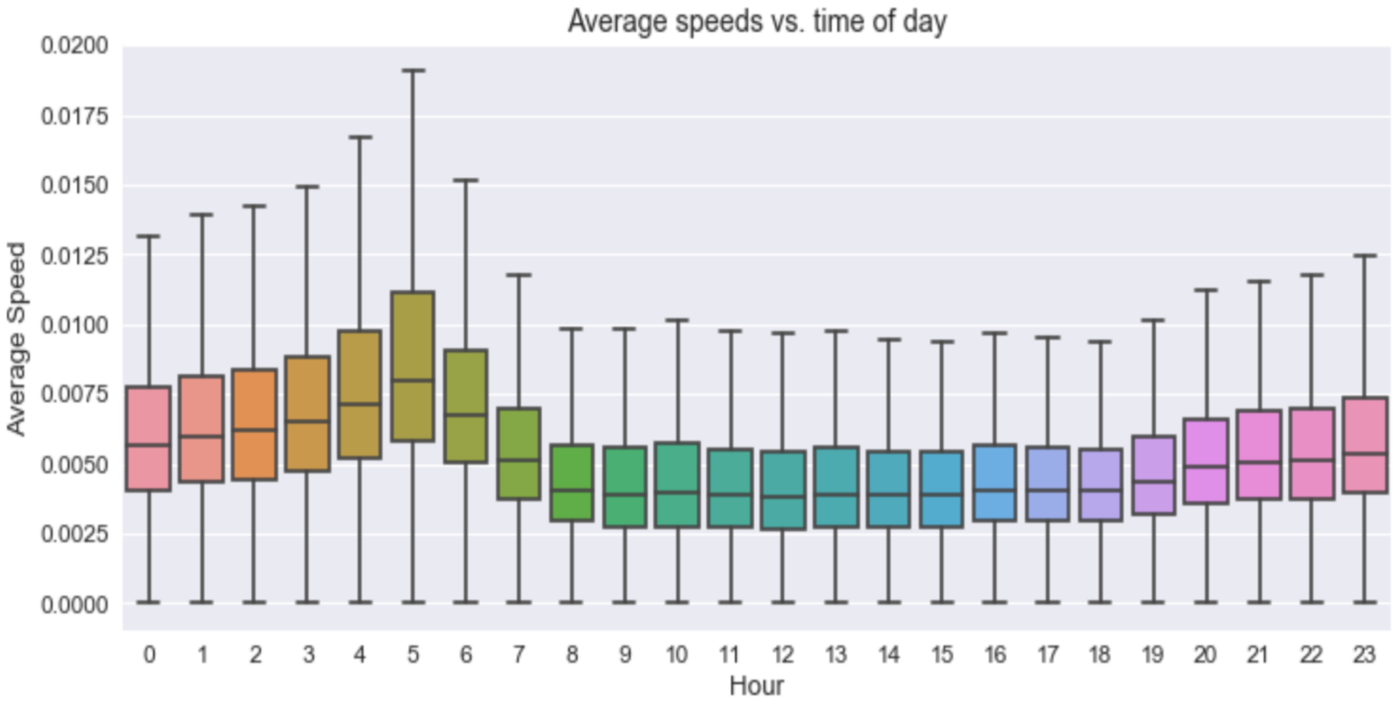
The graph represents the relation between the log_trip_duration (log(trip_duration+1)) and the hour of the day. (sns boxplot with showfliers =False)



The graph points to the fact that the trip duration between 5am to 8am is noticablly lower than the other times of the day this could be due to the increas in traffic post 8am, to get some more insight in this assumption I decided to compare the speed of taxi with respect to the hour of the day.

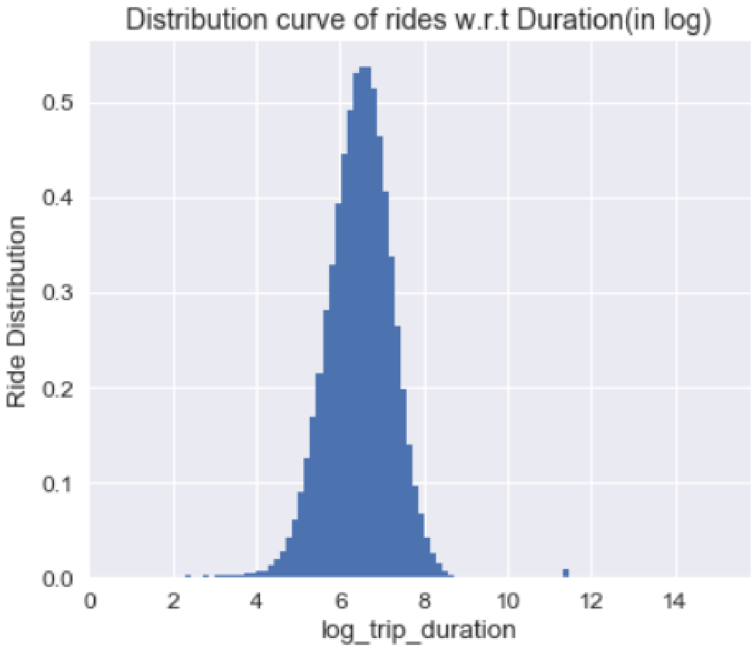
Since we have only been provided with the coordinates I have used three different distance metrics namely haversine distance manhattan distance and bearing distance which are defined in detail in the next section.

To obtain a graph for speed vs time of day I haves used the manhattan speed which is equal to Manhattan speed = Manhattan distance / Time

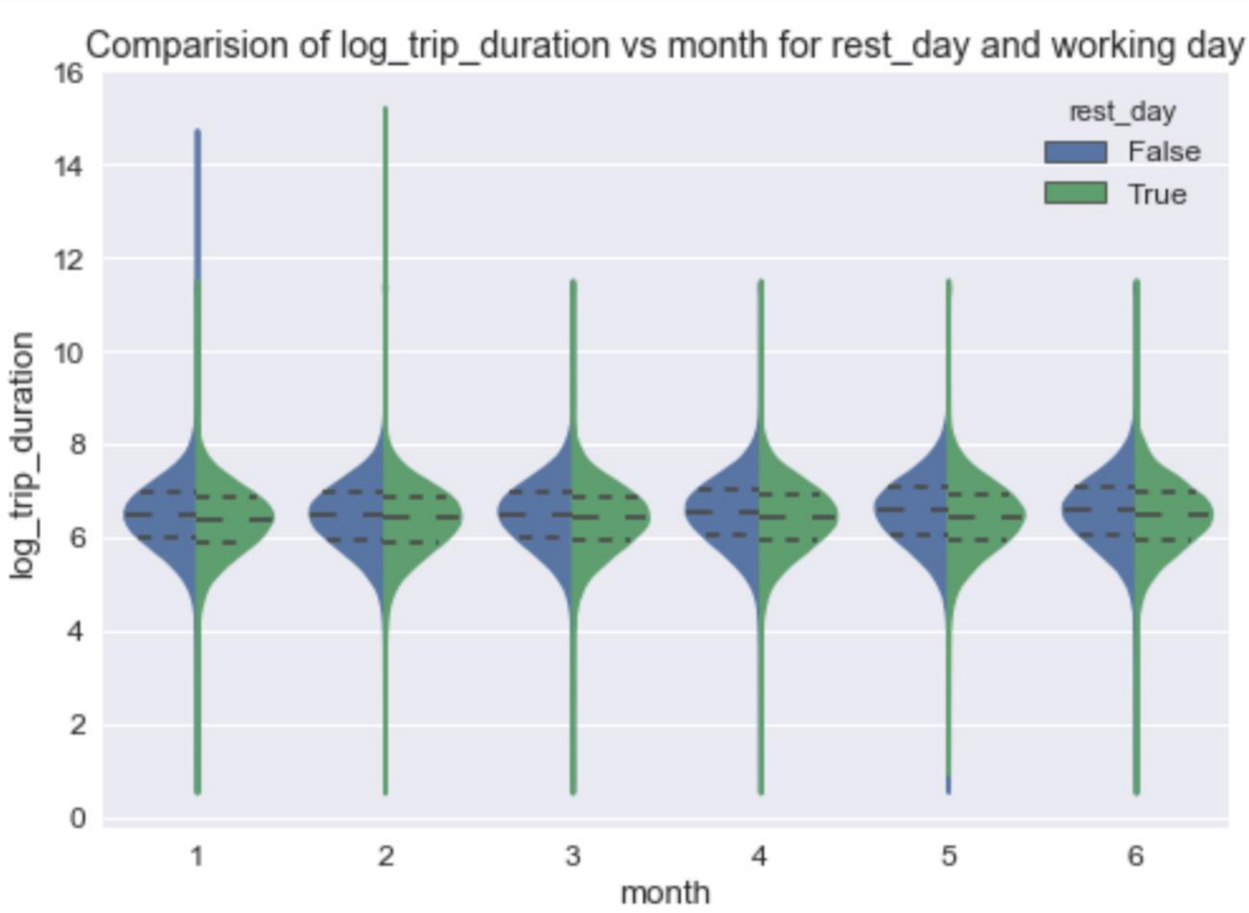


This graph gives us a clear picture of how the speed varies throughout the day. Its clearly observed that the speed increases as the night proceeds and peaks during the super early hours of the day and then starts to fall heavily after 8am. A potential reason could be the increase in traffic due to office hours. Keeping this in mind I will try to create features which will take into account whether the day is a working day or a holiday.

Further I wanted to explore the distribution of the trip_duration to understand the nature of data and pick a model.



After creating the feature space, using my secondary dataset I isolated all the workdays and the rest days. The rest days includes the weekends and the public holidays. In order to check the dependency of this with the log trip duration over the months I plotted a violin plot. The violin plot compares the trip duration vs the months of the year for both the rest days and the work days. Here it was observed that the values for rest day were lower than that of a work day.



2.3 ALGORITHM

The algorithm of choice for this problem will be XGBoost, XGBoost stands for extreme gradient boosting and is an implementation of an ensemble method called the gradient boosting trees. This model tends to outperform other similar bagging models like random forest as this is sequential in nature, i.e. each weak tree learns from the mistakes of the tree before it instead of ensembling multiple independent trees.

In simple words the structure of the algorithm is as follows:

1. A decision tree is created and fitted to the data. This tree is then compared against all the data entries.

2. Based on the errors of the first tree the weights are modified, and another tree is created which uses the updated weights.
3. This new tree will gradually start covering more data points and perform better, this tree is then added, and then previous steps are repeated.

Being a tree based algorithm it has a lot of advantages like:

1. It is scale invariant which reduces the need of preprocessing and transformations.
2. Being a tree based technique it can also map nonlinear data for example here it could easily map to pick up coordinates.

Having mentioned the above advantages, it is also important to take care of possible overfitting. To counter such possibilities, I haves used early stopping mechanism and also a proper validation set.

The XGBoost has a lot of parameters to work with, these parameters can be split into categories like General parameters, Tree Boosting parameters and learning task parameters. The general parameters deal with the performance based parameters like the booster to be used (gbtree being the default), number of threads to be used etc. The learning task parameters comprise of special parameters that define the objective like linear regression etc. The main optimization is done on tree Boosting parameters which define the characteristics of the trees. The parameters I have used for training and optimization are:

1. Max_depth: It defines the number of splits per each individual tree, the higher the max depth the more complex the model becomes and the chances for overfitting increases. There is no direct limit to this parameter. The default value is 6.
2. Eta: It defines the step size or also known as the learning rate for updating weights. The smaller the value the better the results but slower the training. The range of eta is [0,1] and is set to 0.3 by default.
3. Subsample: This parameter defines the proportion of data that the model uses randomly for each tree, this is used to prevent overfitting. The default value is 1.
4. Lambda: This is a regularization term the higher the value the more conservative the model will be.
5. Colsample_bytree: subsample ratio of columns when constructing each tree. Default value is 1.
6. Colsample_bylevel: subsample ratio of columns when performing each split on each level. Default value is 1.
7. Min_child_weight: This defines the minimum number of instances needed to be in each node of a tree to be included. The default value is 1.

The subsample, colsample_bytree, colsample_bylevel are mechanisms in place to reduce overfitting along with early stopping.

2.4 BENCHMARK

For the benchmark in this project I chose a vanilla for of XGBoost i.e. default parameters and no new added features. The implementation of a vanilla XGB regressor gave me a RMSLE of 0.46906 which places me equivalent to a 757 rank on the competition leaderboard.

In order for our model to be successful I have to beat the benchmark model and score as much as I can on the competition leaderboard.

3. METHODOLOGY

3.1 DATA PREPROCESSING

Since our chosen method "XGBoost" can utilize numerical values only, I had to convert the input data to a suitable form, for which I made the following changes:

1. vendor_id: Initially being a categorical feature, this column have only 2 possible values. Since only 2 values are valid we can skip the on hot encoding as we can represent it as a Boolean variable instead.
2. store_and_fwd_flag: similar to the vendor_id this has 2 possible values 'Y' and 'N', so I mapped this into a Boolean 1 for 'Y' and 0 for 'N'.
3. pickup_datetime: since this is a datetime data, it was important to extract the individual components namely year, month, day, hour and minute.

3.2 FEATURE ENGINEERING

Followed by the preprocessing of data comes a very intergral part of the project, the step of feature engineering includes choosing the right features, transforming them and creation of more feature with the current data.

REST DAY

From the data exploration and visualization, we see a clear dependency of the trip duration and the nature of the day (work or rest). The first set of new features in my feature space are the rest_day and work_day. The restday function created in the code reads the data from the dataset "NYC 2016 Holidays" and for any given date checks if it is a weekend or a public holiday. These features are sored in the columns 'rest_day' and 'weekend'.

FASTEST ROUTE

The reason to use the 'New York City Taxi with OSRM' data base as my primary database was to use the data relating to fastest route as a feature, this information wouldn't have been possible for me to compile hence decided to use this feature provided by Kaggle in the playground. The columns we will be using from the fastest_rout.csv are:

[id, total_distance, total_travel_time, number_of_steps, step_direction].

Where,

number_of_steps: The number of actions performed to reach the destination

step_direction: the sequence of actions performed to reach the destination (turning left/right, none, arrive)

I have then used the step_direction to calculate the number of left and right turns made and stored them in columns ‘left_steps’ and ‘right_steps’ respectively. Post this I have dropped the step_direction and used all other features as a part of my feature space.

DISTANCE AND DIRECTION

Since we are only given coordinate pairs for our pickup and dropoff points estimating the right distance metric is really important. To calculate the spherical distance between the two points I am using rhe haversine formula, to emulate the vertical and horizontal travelling I will be calculating the manhattan distance and finally to get an Idea of theh direction of travel I will be calculating the Bearing. The three metrics mentioned are define as:

Haversine Distance:

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

Where

d = distance along the sphere

r = radius of sphere (earth in our case)

λ_1, λ_2 = longitude of point 1 and point 2

φ_1, φ_2 = latitude of point 1 and point 2

Manhattan Distance:

The manhattan distance only takes into account the vertical (north/south) and horizontal movments (east/west) and can be defined as:

$$Manhattan\ Distance = |\,\lambda 2 - \lambda 1| + |\varphi 2 - \varphi 1|$$

Where,

λ_1, λ_2 = longitude of point 1 and point 2

φ_1, φ_2 = latitude of point 1 and point 2

We can calculate the manhattan distance in terms of haversine distance as

Manhattan distance = haversine(φ_1, λ_1 φ_1, λ_2) + haversine(φ_1, λ_1, φ_2 , λ_1)

Bearing:

$\theta = \text{atan2}(\sin \Delta\lambda \cdot \cos \varphi_2 , \cos \varphi_1 \cdot \sin \varphi_2 - \sin \varphi_1 \cdot \cos \varphi_2 \cdot \cos \Delta\lambda)$

Where

λ_1, λ_2 = longitude of point 1 and point 2

φ_1, φ_2 = latitude of point 1 and point 2

$\Delta\lambda = \lambda_2 - \lambda_1$

For our feature space I have calculated the mentioned metrics for both training data and test data and have stored them as ‘haversine_dist’, ’manhattan_dist’ and ’bearing’ respectively.

LOCATION CLUSTERS

The influence behind this approach was that, different pickup and dropoff locations will impact the trip duration as different areas will have different traffic patterns. To incoorperate this I did a kmeans clustering on the pickup and dropoff coordinates and divided the entire data set into 10 location clusters.

WEATHER DATA

My next set of features include the weather information, I strongly believe weathehr conditions that impacts visibility or driving conditions like fog and snow will directly impact the trip duration. For this I am using data from the data set KNYC Metars 2016” to extract the visibility and temperature based features for 2016 in NY City.

I first isolated the events related to snow and fog and for these entries I used the following features: ['month','day','hr','Temp.','Precip','snow', 'Visibility']

Where

'month','day','hr' is being used to map the data to the training and testing data

'Temp.','Precip','snow', 'Visibility' are the actual features added.

The final feature space contains the original features present in the training data, the rest_day ,weekend from the restday set, , total_distance, total_travel_time, number_of_steps, left_steps, right_steps from the fastest route set, the manhattan_dist, haversine_dist and bearing from the distance metrics set, the location one hot encoding 0-9 from location cluster set and finally Temp., Precip, snow, Visibility from the weather set.

3.3 IMPLEMENTATION

The entire project is implemented in the form of a jupyter notebook. The notebook can be split into four parts namely preprocessing, feature engineering, visualization and modelling. The notebook has the following ordering:

1. the training and testing files are read into dataframes using the pandas library, and the basic information and statistical profile of the training data is explored.
2. The datetime variable ‘pickup_datetime’ is split into its 5 components ‘year’, ‘month’, ‘day’, ‘hr’, ‘min’. at this stage the ‘store_and_fwd_flag ’ is also converted to ‘0’ or ‘1’ from its previous categorical state of ‘Y’ and ‘N’. This is done to both training and testing data.
3. Distribution of data by vendor_id, store_and_fwd_flag are calculated and number of datapoints out of the longitude range are calculated for potential outliers, however number being too low no action was required.
4. The target variable trip_duration is now converted to log(trip_duration+1) as the evaluation metric is RMSLE.

This is where the feature engineering starts, while doing the project this is where I did some visualization as explained in the previous sections, in order to keep the notebook organized I moved all the feature engineering before the visualization:

5. The first feature set I introduce is ‘restday’ and ‘weekend’. These are obtained from the secondary dataset “NYC_2016Holidays”. The days are then compared using the isoweekend() function and the final features are stored in time_data fro training dataset and time_test for test data. The time data is subsequently changed into floats for convenience.
6. The next set of features I work with are the features from the NYC_taxi_with_OSRM database, this database includes features like fastesroute which I used. As explained in the above sections I extract the number_of_steps and also calculate right_steps and left_steps along with the total_travel_time. This is then replicated for the test data as well. The final features are stored in osrm_data and osrm_test respectively.
7. The next set of features were the distance based features, I first defined functions for all the distance metrics and then calculated these metrics for both training and testing data. These features were stored as Other_dist_data and Other_dist_test respectively.
8. The next feature set is the location clusters, I simply used the MiniBatchKmeans to form 10 cluster from the entire data and store them in kmean_data and kmean_test.
9. Finally, the last feature set I wanted to add was the weather data, for this I read the dataset ‘KNYC_Metars’ and isolated the events related to fog and snow. I then used the datetime stamp to map this data to the training and testing data and used the Temp., Precip, snow and visibility columns. The results from this were stored as weather_data and weather_test respectively.

This is the part of the notebook which deals with all the visualizations:

10. The first plot includes the distribution of trip durations throughout the day, I used sns boxplots for this.
11. In the second plot I wanted to plot the relationship between the speed and the hour of the day, I calculated the manhattan speed here by dividing manhattan distance by time per trip to find data for this plot. Again, I used sns boxplot for the representation.

12. The next thing I wanted to check was the actual distribution of trip durations for the entire training data.
13. Now I wanted to compare the trip durations for rest day and workday throughout the year, for this I came across an interesting concept of violin plots and used the sns.violinplit for the implementation.

This brings us to the final part of the notebook, the modelling and optimization:

14. The first step was to actually merge all the features formed and the training data and testing data. After converting the location clusters into one hot encoding I merged all the features mentioned above with the training and testing data using pandas.concat the final dataframes were stored as mydf for training data and testdf for test data.
15. Just to be sure that the training dataframe and the test dataframe are similar in structure I verified is the keys were equal.
16. The next step was defining the xgb model, first I performed a 80-20 split on the data for cross validation and created the Dmatrix which are consumed internally by the XGB model.
17. The next step was defining the parameters for the model, Started with an educated guess, I wrote a parameter search function substituting the GridSearchCV (a technique I learned in one of the kernels (here) on Kaggle to incorporate the early stopping).
18. Using the above parameter search function and finding optimal parameters, I trained the model and also using the xgb.plot_importance() function I plotted the feature importance graph for more information. Number of iterations are set to 2000 and an early stopping of 50 on RMSLE values from validation set is in place.
19. Finally using the test data I predict the taxi trip durations and convert them back from their existing log(trip_duration+1) form to seconds form. Using this a submission file is then created for Kaggle.
20. In order to check the robustness of my model I plot the predicted log trip duration of my validation set vs. the actual values in my validation set using a scatter plot.

3.4 REFINEMENT

My initial results with parameters decided based on an educated guess + defaults ended up with an error of 0.39325. To improve this result and for parameter tuning and optimization I initially used the GridSearchCV but it proved to be very time consuming in this setting due to the lack of early stopping. SO on further research for parameter optimization I came across a technique mentioned in one of the kernels on Kaggle(can be found here) and wrote a similar function with my parameter set. This function proved to be quicker than GridSearchCV.

After the parameter tuning I got to a score of 0.38241.

4 RESULTS

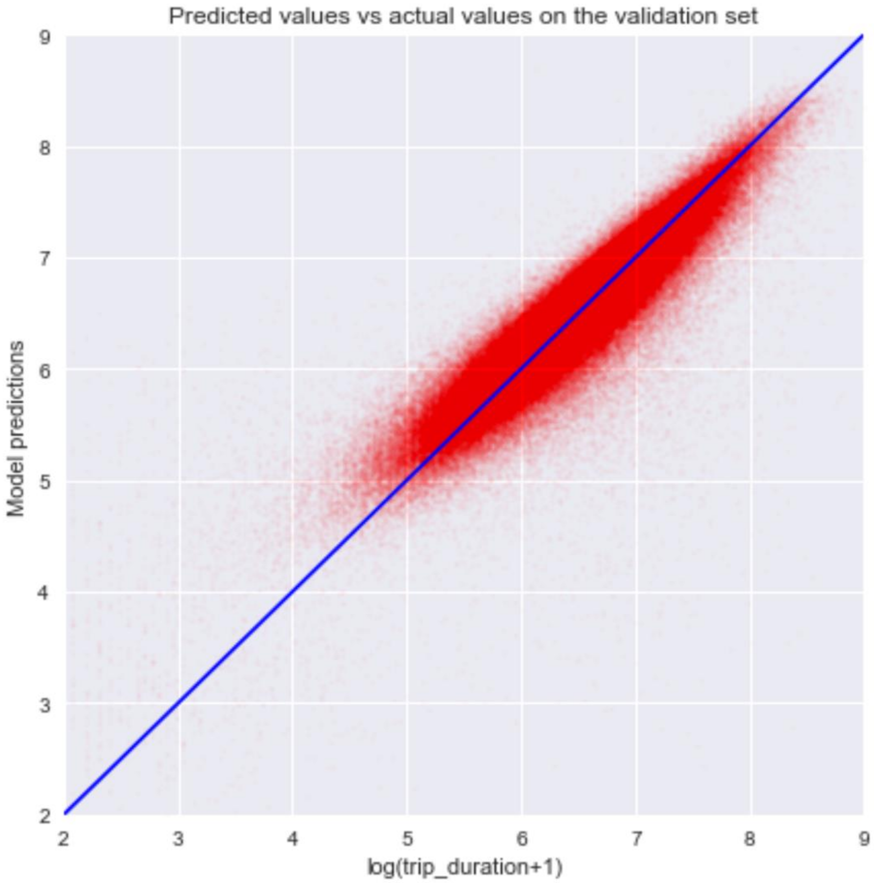
4.1 MODEL EVALUATION AND VALIDATION

The optimal parameters found were:

[min_child_weight': 0.5, 'eta': 0.05, 'colsample_bytree': 0.6 , 'colsample_bylevel':0.7, 'max_depth': 15, 'subsample': 0.9, 'lambda': 3].

This final parameter set gave me an error value of 0.38241 which places me at 230 rank out of 1257 present teams.

In order to check how well my model did in terms of prediction I plotted the actual values vs the predicted values for my validation set.



Looking at the graph its safe to say the model does a decent job, however at very small durations the data can get a little unpredictable but overall it seems to be robust.

4.2 JUSTIFICATION

The final model gives me an error score of 0.38241 which is drastic improvement from my initial vanilla XGB benchmark model with the score of 0.46906. The new features added improved the score by over 22%. Increasing my score to a rank equivalent of 230 from a rank equivalent of 756 on the benchmark model which is a great improvement.

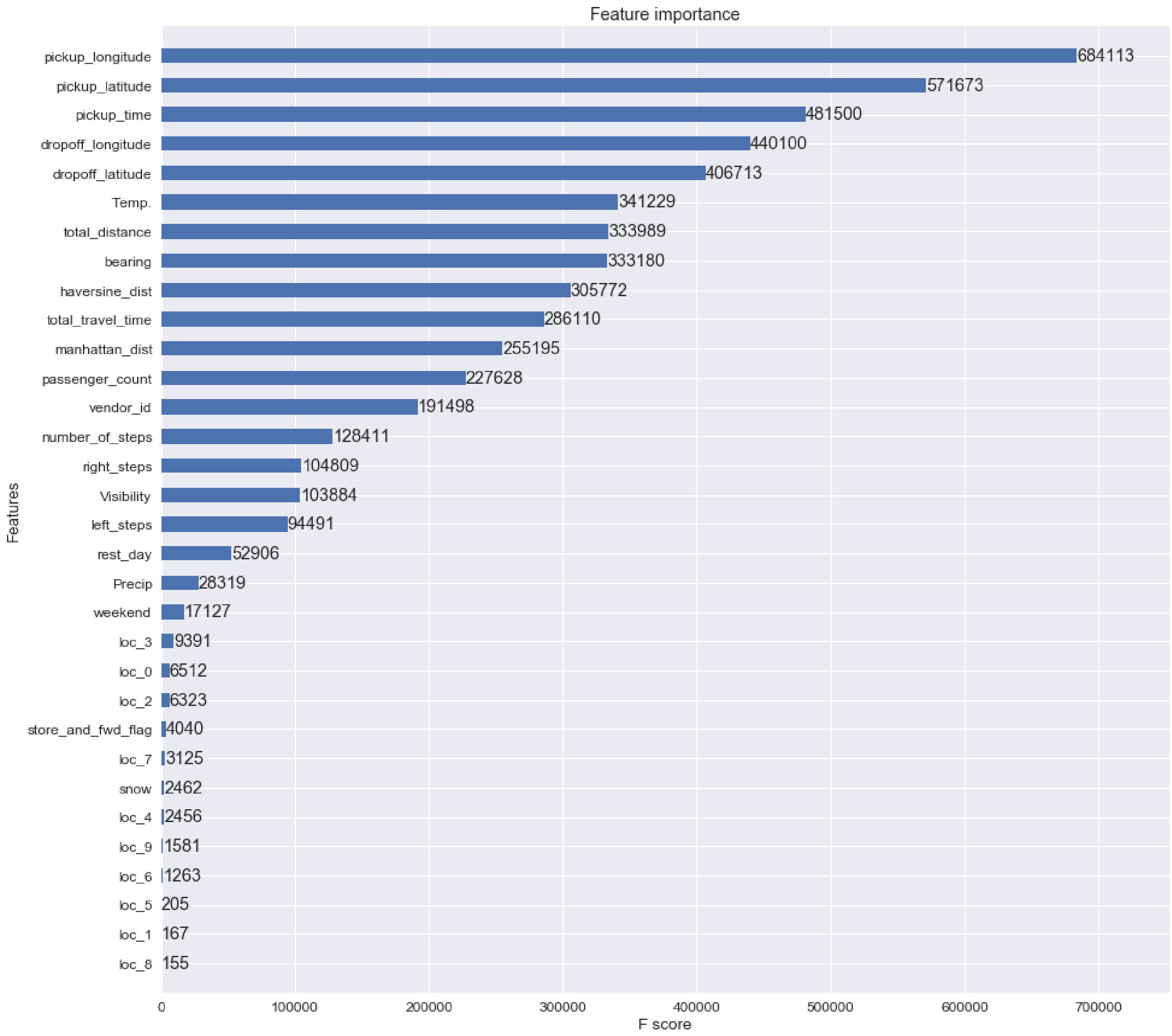
The final model with the added feature, according to me, has done a great job at modelling the data and predicting the taxi trip duration.

5 CONCLUSION

5.1 FREE-FORM VISUALIZATION

To get a final picture of the model I plotted the feature importance plot using the xgb.plot_importance() function. This plot holds the information on the most impactful features. The findings from this plot are as follows:

1. The most important or impactful features are the pickup_longitude and pickup_latitude, this suggests that there is a probable traffic pattern depending on the busy areas of the city, I wanted to emulate this using the location clusters however I feel my clusters were too spread out.
2. The next priority was given to the pickup_time making the ‘traffic hypothesis’ even more concrete, the time plays an important role because of the traffic patterns.
3. For the distance metrics the direction of travel ‘Bearing’ and haversine distance were the important ones.
4. Surprisingly the temperature was rated more than the visibility and snow, a potential cause for this could be that in cases of extreme temperatures like too hot or too cold, people in general avoid stepping outside.
5. Another surprise to me was the low rating of weekend and rest_day, however I feel splitting these features into two different ones have probably diluted its impact because I still feel it plays an important role in predicting the trip duration.



5.2 REFLECTION

The entire project can be broken down into a few steps.

1. Preprocessing and exploratory data analysis: this includes the study of data as it is, visualizations to get a better insight of the data and is a foundation of feature engineering.
2. Feature Engineering: arguably the most important step, this includes creating a feature space for the final model, from the features present in the primary and secondary datasets. This also includes transforming the features into compatible forms for the model.

3. Modelling and Optimization: this include choosing an appropriate model and fitting the model on the feature space created in the previous step to with random/guessed parameters. Now the task transforms from fitting to optimization and parameter tuning for optimal results.

According to me the most challenging and interesting step was that of feature engineering. This is a step that is unique to the problem we are dealing with and there is no fixed way to do it. It involves a lot of brain storming and thinking out of the box speculating multiple scenarios.

I think the second most challenging part was the parameter tuning, it involves in a lot of experimentation and educated guesses. It teaches you how each parameter of the model impacts the performance which is a great learning.

For this project while researching I learned a lot of new techniques for visualization from various Kaggle kernels and even came across a lot of different ways of parameter tuning.

5.3 IMPROVEMENT

Certain aspects that could be implemented were using PCA on the coordinates might lead to a better solution, another thing I would have liked would be instead of using the fastest rout feature from the OSRM database I could have taken data from google maps API which could probably give a better estimation.

Apart from the possible improvements in feature engineering some improvements could be done on the modelling side as well, a popular approach on Kaggle is stacking of multiple models which usually improves the results quite significantly. This however is out of my capabilities at the moment and is something I intend to use in the near future.

REFERENCES

Apart from the links linked in the report here are a few additiona resources worth mentioning

- The XGBoost Paramaeters from the documentation present at https://github.com/dmlc/xgboost/blob/master/doc/parameter.md.
- Complete guide to parameter tuning in XGBoost: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
- Complete guide to parameter tuning in Gradient Boosting (GBM): https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
- Official Competition Page: https://www.kaggle.com/c/nyc-taxi-trip-duration
- From EDA to Top, Kaggle kernel by beluga: https://www.kaggle.com/gaborfodor/from-eda-to-the-top-lb-0-367
- A complete Guide to NY Taxi Data, Kaggle kernel by Weiying Wang: https://www.kaggle.com/onlyshadow/a-practical-guide-to-ny-taxi-data-0-379