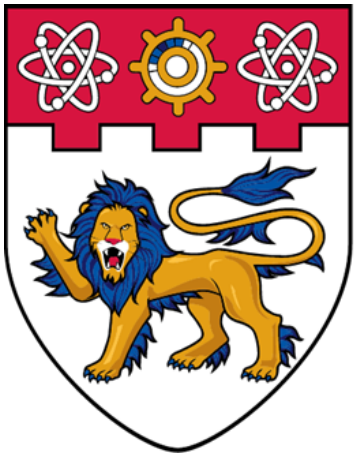


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE/CZ4041: Machine Learning

Project Report

Group 22

Group Members:

HENG CHENG KIAT (U1822588L)

NORMAN FUNG ZI JIE (U1820628C)

LEE SU BIN (N2101698B)

GUO FEIYAN (U1922356D)

Video Presentation Link: https://youtu.be/1Grg2C7dPyU

Table of Contents

Overview

Our group decided to participate in the New York City Taxi Fare Prediction Competition organized by Kaggle. Our task is to develop algorithms that use a broad spectrum of features to predict the taxi fares in New York City.

Both the train and test sets provide details of each taxi ride. The target variable "`fare_amount`" is the label we need to predict for the test set.

The evaluation metric used for this competition is **Root Mean Square Error (RMSE)**.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Fig. 1: Formula of RMSE

Roles and Contributions

Our team only consists of 4 members. The project was carried out in a linear approach where each member is involved in each part of the project. This includes data exploration, pre-processing, feature engineering, and implementation of various Machine Learning Algorithms.

Score and Ranked Position

Fig. 2: Snapshot of the total number of participating teams.

There are a total of 1483 teams that have participated in this competition. A lower RMSE score indicates a better prediction and a higher-ranking position.

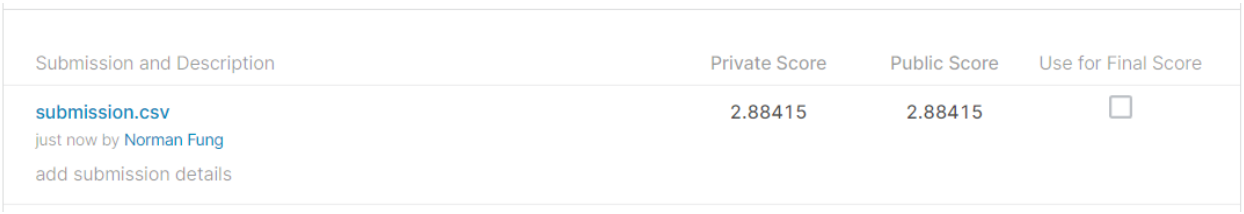


Fig. 3: Snapshot of best final score

For the Kaggle competition, the private leaderboard is calculated over the same rows as the public leaderboard. The best final score we are able to achieve is **2.88415** as shown in figure 3 above.

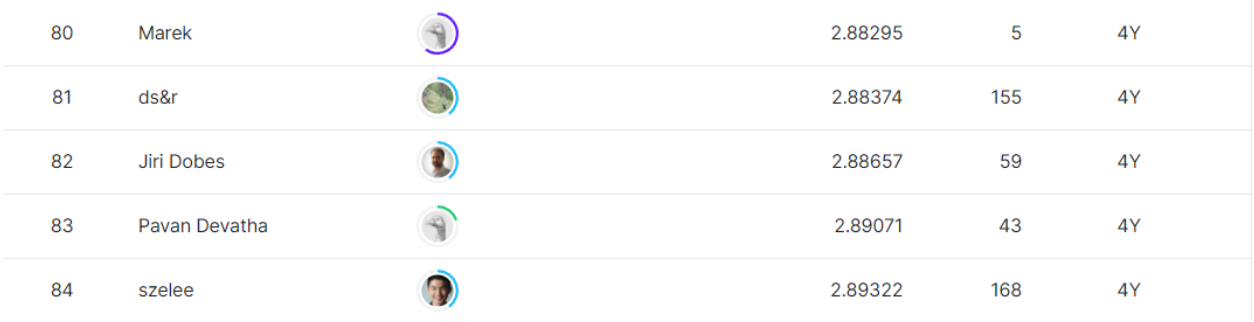


Fig. 4: Leaderboard Ranking.

As shown in Fig.4, our leaderboard ranking is between the 81st and 82nd positions. Hence for this competition, we are in the top **5.5%**.

Data Exploration

In the train dataset, there are about 55 million rows and 8 columns. Since this is a huge dataset, we have decided to use a subset of 20 million rows instead for our experiment. The dataset includes the "`key`" of each input as well as the "`fare_amount`" variable that we are supposed to predict. The following two figures show the head of the train dataset and some basic statistical details, respectively.

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1

Fig. 5: Head of train dataset.

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	2.000000e+07	2.000000e+07	2.000000e+07	1.999986e+07	1.999986e+07	2.000000e+07
mean	1.134298e+01	-7.251140e+01	3.992070e+01	-7.251060e+01	3.991950e+01	1.685312e+00
std	1.689916e+01	1.298955e+01	9.388927e+00	1.290617e+01	9.570037e+00	1.321177e+00
min	-1.077500e+02	-3.439245e+03	-3.492264e+03	-3.442025e+03	-3.493652e+03	0.000000e+00
25%	6.000000e+00	-7.399207e+01	4.073491e+01	-7.399140e+01	4.073403e+01	1.000000e+00
50%	8.500000e+00	-7.398181e+01	4.075263e+01	-7.398016e+01	4.075315e+01	1.000000e+00
75%	1.250000e+01	-7.396709e+01	4.076712e+01	-7.396368e+01	4.076809e+01	2.000000e+00
max	6.155086e+04	3.457626e+03	3.406008e+03	3.457622e+03	3.400392e+03	2.080000e+02

Fig. 6: Basic statistical details of train dataset.

In the test dataset, there are 9914 rows and 7 columns. The following two figures show the head of the test dataset and some basic statistical details, respectively.

	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24.0000002	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24.0000003	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44.0000002	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12.0000002	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12.0000003	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1

Fig. 7: Head of test dataset.

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	9914.000000	9914.000000	9914.000000	9914.000000	9914.000000
mean	-73.974722	40.751041	-73.973657	40.751743	1.671273
std	0.042774	0.033541	0.039072	0.035435	1.278747
min	-74.252193	40.573143	-74.263242	40.568973	1.000000
25%	-73.992501	40.736125	-73.991247	40.735254	1.000000
50%	-73.982326	40.753051	-73.980015	40.754065	1.000000
75%	-73.968013	40.767113	-73.964059	40.768757	2.000000
max	-72.986532	41.709555	-72.990963	41.696683	6.000000

Fig. 8: Basic statistical details of test dataset.

```
key                  object
fare_amount         float64
pickup_datetime      object
pickup_longitude    float64
pickup_latitude     float64
dropoff_longitude   float64
dropoff_latitude    float64
passenger_count       int64
dtype: object
```

Fig. 9: List of object type data in train dataset.

Data Pre-processing & Cleaning

During the exploration we have observed a few missing values as well as questionable values in the training set. There are "`fare_amount`" which are below 0, as well as longitude and latitude that are beyond the boundaries of New York City.

```
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude  139
dropoff_latitude   139
passenger_count      0
dtype: int64
```

Fig. 10: Null Values count in train dataset.

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	2.000000e+07	2.000000e+07	2.000000e+07	1.999986e+07	1.999986e+07	2.000000e+07
mean	1.134298e+01	-7.251140e+01	3.992070e+01	-7.251060e+01	3.991950e+01	1.685312e+00
std	1.689916e+01	1.298955e+01	9.388927e+00	1.290617e+01	9.570037e+00	1.321177e+00
min	-1.077500e+02	-3.439245e+03	-3.492264e+03	-3.442025e+03	-3.493652e+03	0.000000e+00
25%	6.000000e+00	-7.399207e+01	4.073491e+01	-7.399140e+01	4.073403e+01	1.000000e+00
50%	8.500000e+00	-7.398181e+01	4.075263e+01	-7.398016e+01	4.075315e+01	1.000000e+00
75%	1.250000e+01	-7.396709e+01	4.076712e+01	-7.396368e+01	4.076809e+01	2.000000e+00
max	6.155086e+04	3.457626e+03	3.406008e+03	3.457622e+03	3.400392e+03	2.080000e+02

Fig. 11: Fare amount below zero in training dataset (min value).

To address this issue, we have removed all data points which have a "`fare_amount`" below 0 as well as restricting data points within a set boundary. Since the latitude and longitude of New York City are **40.730610, -73.935242**. We have kept data points that are only within the latitude range of **35 to 45** and the longitude range of **-80 to -70**.

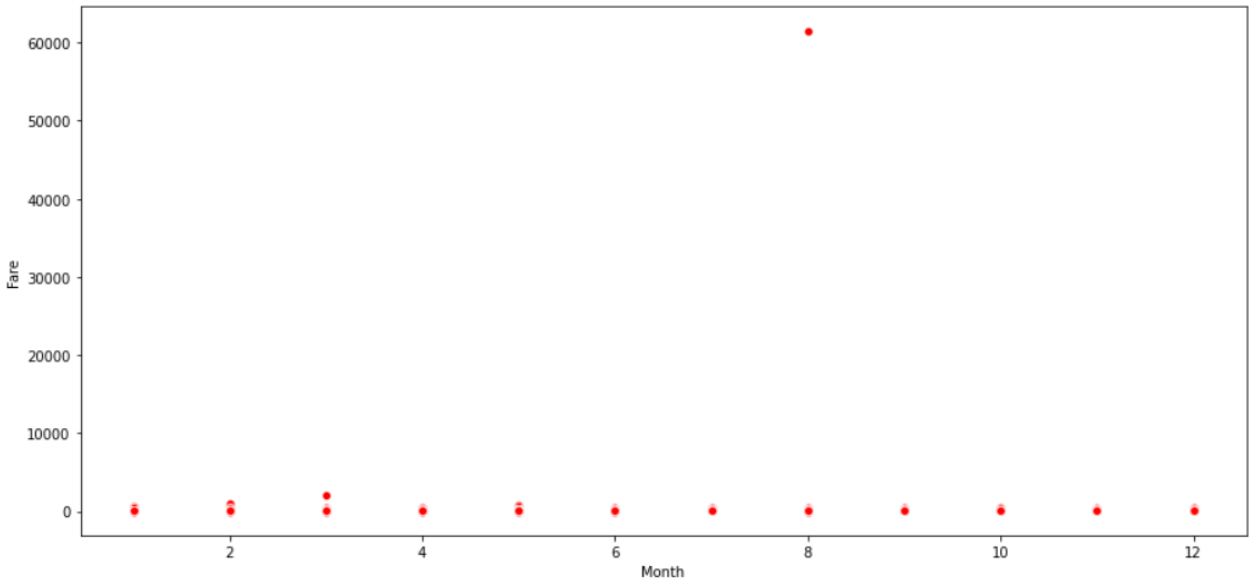


Fig. 12: Extreme outliers in fare_amount.

We have also observed some extremely high fare in the training dataset. We have decided to drop all data points which are above $500 since the majority of cab fares are below that amount.

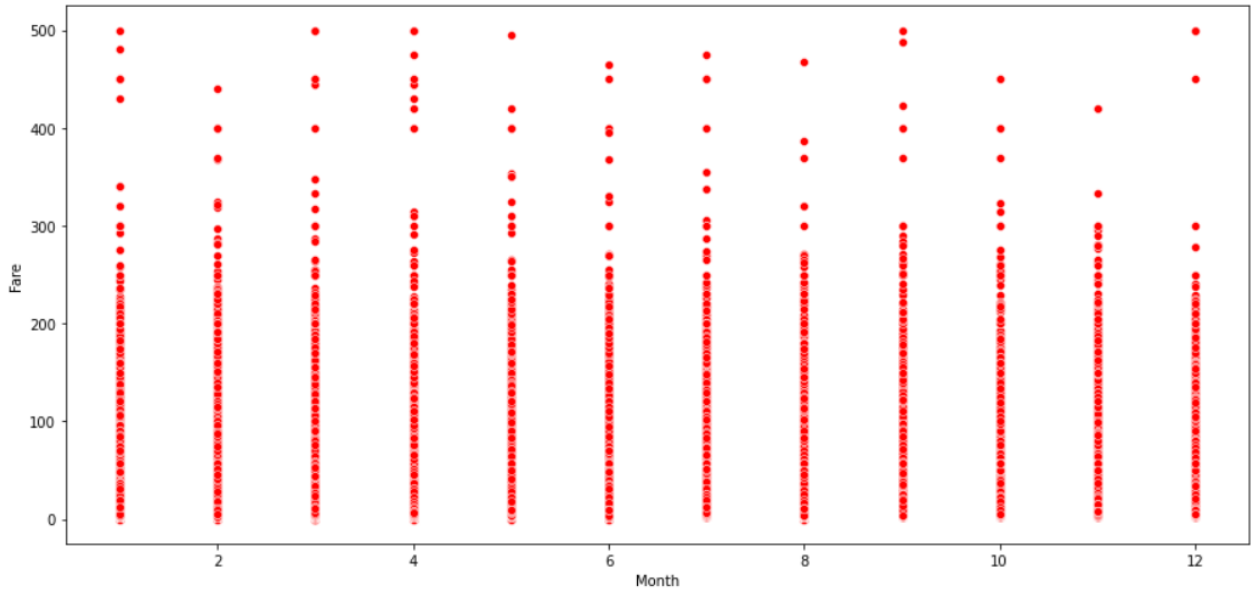


Fig. 13: Screenshot of fare_amount below 500.

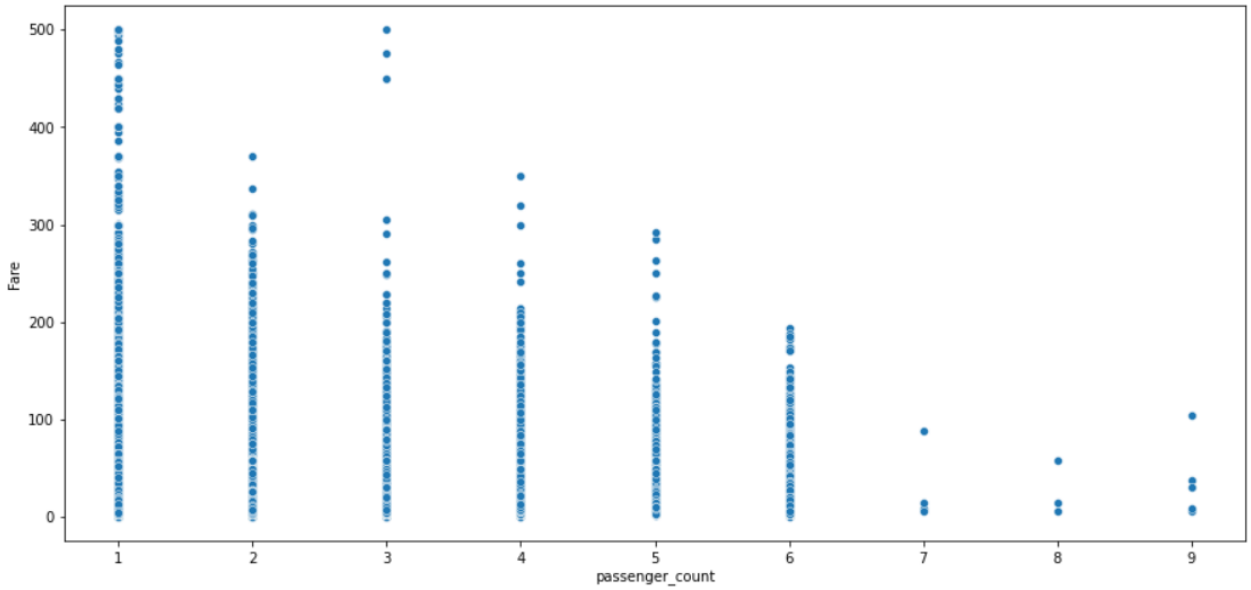


Fig. 14: Screenshot of passenger_count

We have also observed that there were instances where passenger counts were above 6. We have decided to remove all data points that were above 6 passenger counts due to 2 very important reasons. Firstly, it is highly unlikely for more than 7 passengers to be able to fit in a regular taxi. Secondly, the max passenger count for the Test dataset is 6. Therefore, such instances will more likely become noises during training and will affect the performance of our models.

Feature Engineering

Since our dataset has limited features, we have to do some feature engineering for us to get better predictions. The first feature we will create is the separation of the "`pickup_datetime`" column into hour, day, month, weekday, and year.

Datetime Object

pickup_datetime
2009-06-15 17:26:21
2010-01-05 16:52:16
2011-08-18 00:35:00
2012-04-21 04:30:42
2010-03-09 07:51:00

hour	day	month	weekday	year
17	15	6	0	2009
16	5	1	1	2010
0	18	8	3	2011
4	21	4	5	2012
7	9	3	1	2010

Fig.15 Conversion Datetime object into hour, day, weekday, and year.

Haversine Distance

Since our dataset has the longitude and latitude provided for us. We will be able to calculate the distance between the pickup and dropoff point by utilizing the Haversine Formula.

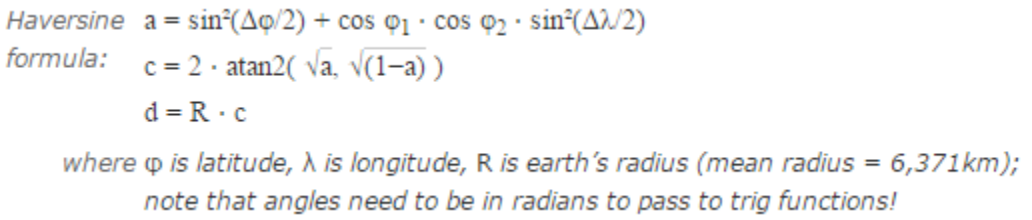


Fig.16 Haversine Distance Formula.

Distance from Airport

Pickup and dropoff distance between random locations and different airports in New York City can also be another important feature since taxis impose a flat rate between the airport and the city. For example Taxis at JFK Airport charge a **flat fare of $52** for trips between the airport and Manhattan. We have included the pickup and dropoff distance for John F. Kennedy International Airport: "`jfk_dist`", Newark Liberty International Airport: "`ewr_dist`", and LaGuardia Airport: "`lga_dist`". The distance is calculated using haversine distance formula and the respective Airport's coordinates.

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	weekday	year	jfk_dist	ewr_dist	lga_dist	distance
0	4.5	2009-06-15 17:26:21	-73.844311	40.721319	-73.841610	40.712278	1	17	15	6	0	2009	9.647141	27.523051	6.660494	1.030764
1	16.9	2010-01-05 16:52:16	-74.016048	40.711303	-73.979268	40.782004	1	16	5	1	1	2010	21.525635	13.029641	8.996046	8.450134
2	5.7	2011-08-18 00:35:00	-73.982738	40.761270	-73.991242	40.750562	2	0	18	8	3	2011	21.735040	16.282199	9.442812	1.389525
3	7.7	2012-04-21 04:30:42	-73.987130	40.733143	-73.991567	40.758092	1	4	21	4	5	2012	20.401167	15.949684	10.241872	2.799270
4	5.3	2010-03-09 07:51:00	-73.968095	40.768008	-73.956655	40.783762	1	7	9	3	1	2010	21.397709	18.867107	7.113348	1.999157
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9999995	5.7	2012-08-12 01:18:00	-73.999464	40.728452	-73.993299	40.742100	2	1	12	8	6	2012	21.053961	14.806838	10.890492	1.604023
9999996	5.5	2013-08-07 10:28:00	-73.968467	40.759367	-73.964967	40.769027	1	10	7	8	2	2013	20.794409	18.431905	7.830685	1.113854
9999997	14.0	2013-10-29 08:29:00	-73.997952	40.733717	-73.973448	40.759122	5	8	29	10	1	2013	21.101710	15.096350	8.727883	3.498755
9999998	10.5	2012-04-07 16:41:33	-73.992700	40.752021	-73.964705	40.772849	1	16	7	4	5	2012	21.551274	16.234784	7.770259	3.304974
9999999	8.5	2010-03-30 19:27:00	-73.965390	40.768572	-73.998188	40.761073	1	19	30	3	1	2010	21.270437	16.259713	7.872131	2.885319

Fig.17 Screenshot of final Dataframe to be used for training.

Fig.17 shows the dataset which we will be using to train with different machine learning algorithms. The same operations have been done to the test set to ensure that the number of features remains the same between the train and test set.

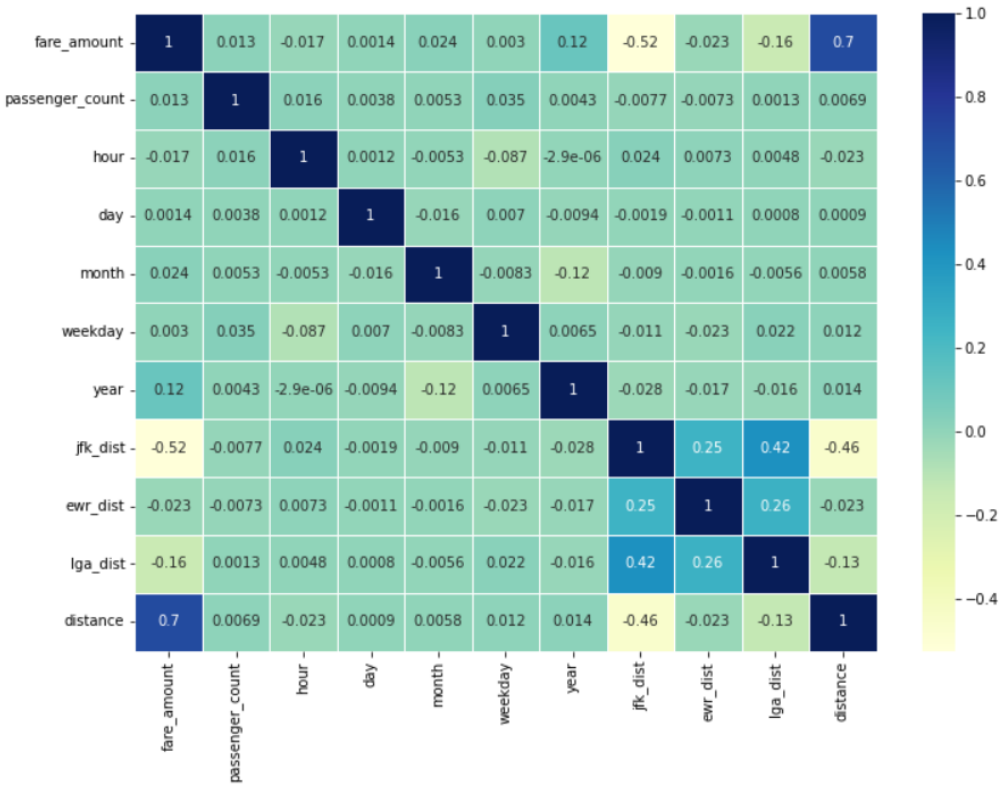


Fig.18 Correlation Matrix

Fig.18 shows the correlation matrix between the existing features. We can clearly see that distance and fare_amount share a high positive correlation of 0.7. We have also created a feature importance plot using XGBoost, and the result also suggested that distance is a strong feature for our model to make better predictions.

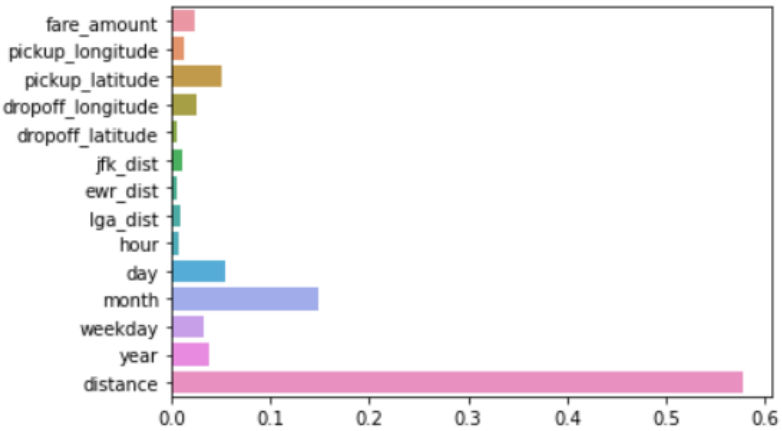


Fig.19 Feature Importance plot

Methodology & Experiments

For the models, we tried many different regression models as listed:

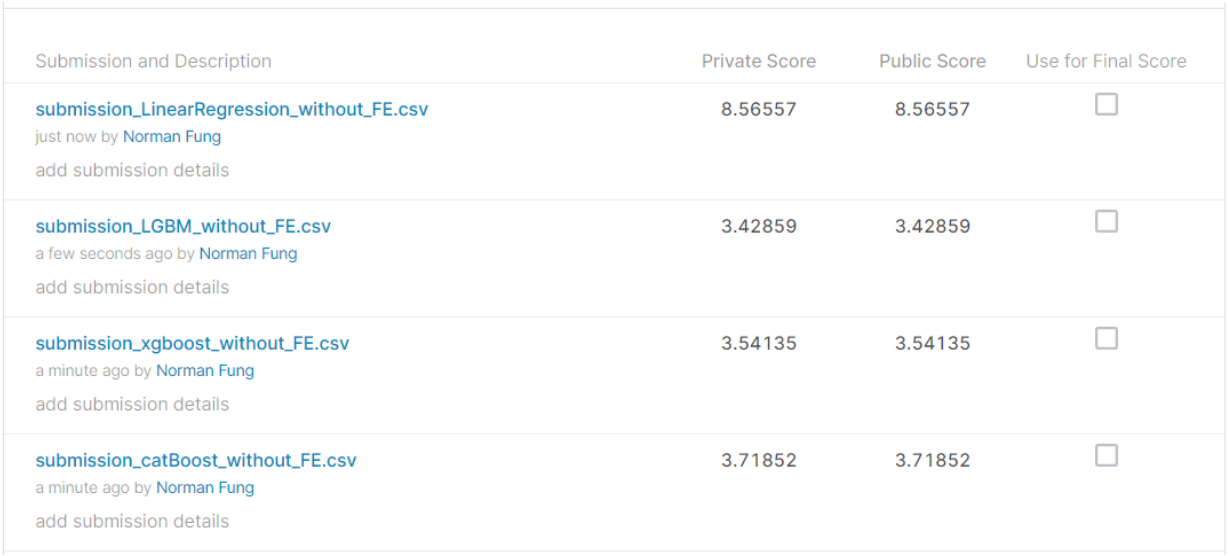
- Linear regression
- XGBoost
- LightGBM
- CatBoost

Our approach was to work by iterations and increase the number of steps taken at each iteration. The table shows the iterations that we did, on top of the data cleaning done previously.

Firstly, we attempt to train the data with just the given features. This helps us to establish a baseline performance for our models. The results from the (X_text, y_test) are conducted locally after splitting the training dataset into 80% training set and 20% testing set. The model will then be used to predict the Kaggle test set.

First iteration	Results (X_test, y_test): RMSE
Training using only given features:	- linear reg: 8.95424 - LightGBM: 3.99232 - XGBoost: 4.21093 - Catboost: 4.36731

Table.1 Results for training using only given features.



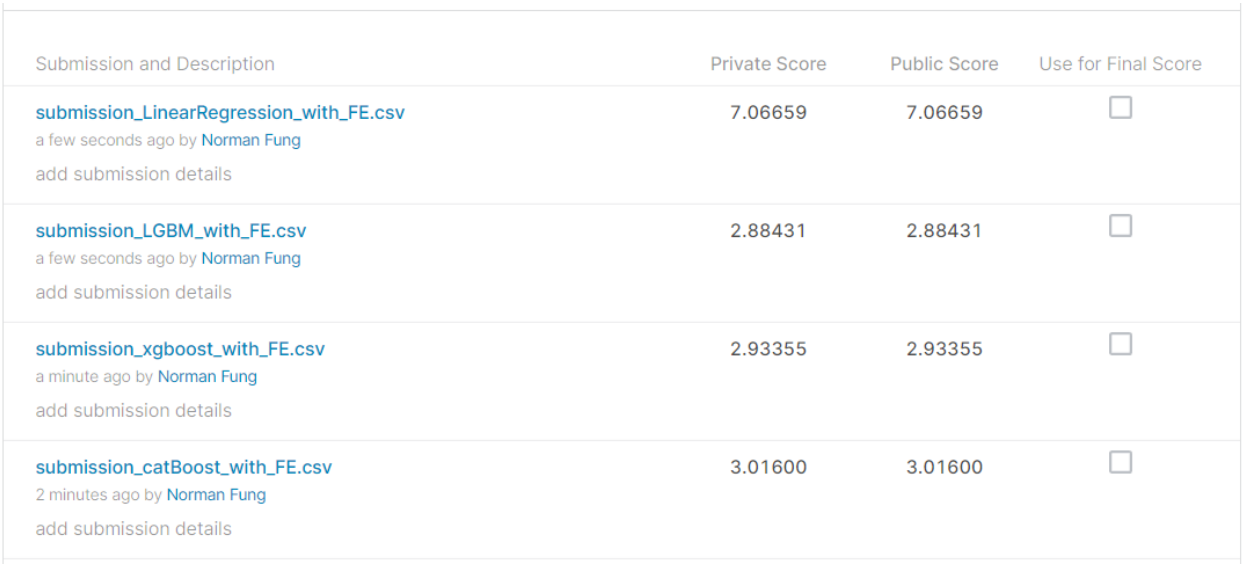
Submission and Description	Private Score	Public Score	Use for Final Score
submission_LinearRegression_without_FE.csv just now by Norman Fung add submission details	8.56557	8.56557	☐
submission_LGBM_without_FE.csv a few seconds ago by Norman Fung add submission details	3.42859	3.42859	☐
submission_xgboost_without_FE.csv a minute ago by Norman Fung add submission details	3.54135	3.54135	☐
submission_catBoost_without_FE.csv a minute ago by Norman Fung add submission details	3.71852	3.71852	☐

Fig.20 Kaggle Result using only given features.

As expected, if we use just train on the given features, our models don't produce a good result. Now that we obtain a baseline, we shall proceed to retrain our models with the additional features obtained from feature engineering.

Second iteration	Results (X_test, y_test): RMSE
Training using additional features obtained from feature engineering.	- linear reg: 7.69064 - LightGBM: 3.49294 - XGBoost: 3.55324 - Catboost: 3.76962

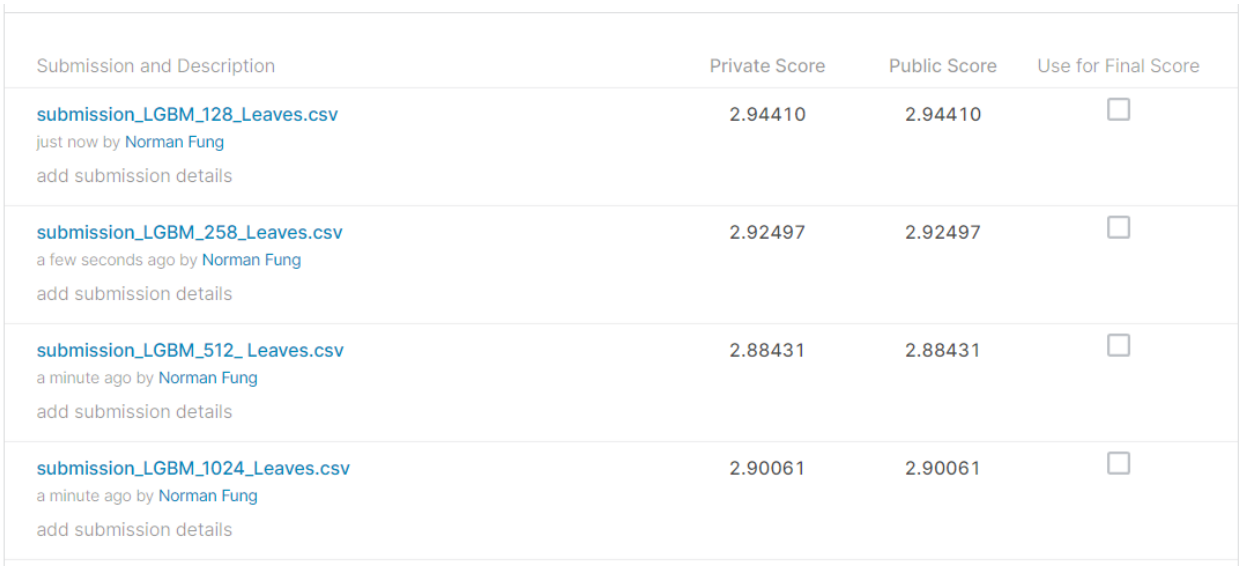
Table.2 Results from using additional features obtained using feature engineering.



Submission and Description	Private Score	Public Score	Use for Final Score
submission_LinearRegression_with_FE.csv a few seconds ago by Norman Fung add submission details	7.06659	7.06659	☐
submission_LGBM_with_FE.csv a few seconds ago by Norman Fung add submission details	2.88431	2.88431	☐
submission_xgboost_with_FE.csv a minute ago by Norman Fung add submission details	2.93355	2.93355	☐
submission_catBoost_with_FE.csv 2 minutes ago by Norman Fung add submission details	3.01600	3.01600	☐

Fig.21 Kaggle Result from using additional features obtained using feature engineering.

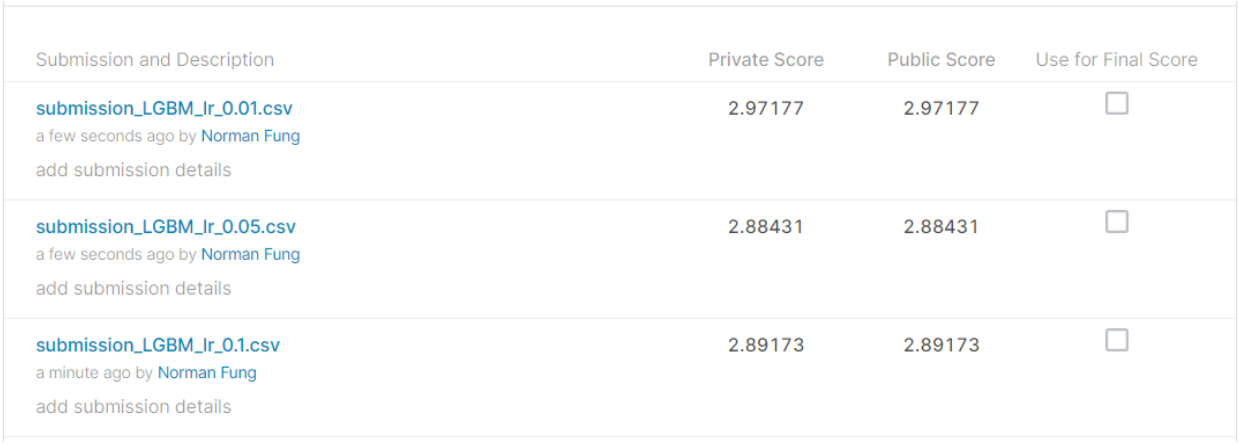
From the results obtained in Figures 20 and 21, we can clearly observe a huge improvement across our models after training the dataset with additional features. The results reflected in Fig.21 are the best scores achieved from each regressor algorithm. To obtain the best model for each algorithm, we have experimented with different hyperparameters.



Submission and Description	Private Score	Public Score	Use for Final Score
submission_LGBM_128_Leaves.csv just now by Norman Fung add submission details	2.94410	2.94410	☐
submission_LGBM_258_Leaves.csv a few seconds ago by Norman Fung add submission details	2.92497	2.92497	☐
submission_LGBM_512_ Leaves.csv a minute ago by Norman Fung add submission details	2.88431	2.88431	☐
submission_LGBM_1024_Leaves.csv a minute ago by Norman Fung add submission details	2.90061	2.90061	☐

Fig.22 Experiment with different numbers of leaves for LightGBM.

We've experimented with different numbers of leaves for LightGBM, the accuracy of the model has continued to improve as we increase the number of leaves for the first 3 iterations. However, on the fourth iteration, the performance decreases, which shows signs that the model is overfitting the data. Next, we experimented with different learning rates.



Submission and Description	Private Score	Public Score	Use for Final Score
submission_LGBM_lr_0.01.csv a few seconds ago by Norman Fung add submission details	2.97177	2.97177	☐
submission_LGBM_lr_0.05.csv a few seconds ago by Norman Fung add submission details	2.88431	2.88431	☐
submission_LGBM_lr_0.1.csv a minute ago by Norman Fung add submission details	2.89173	2.89173	☐

Fig.23 Experiment with different Learning Rates for LightGBM.

After experimenting with different learning rates, we can see that a lower learning rate doesn't always guarantee better performance. This may be due to the model getting stuck in an undesirable local minimum.

Submission and Description	Private Score	Public Score	Use for Final Score
submission_xgboost_500_n_estimators.csv 5 minutes ago by Norman Fung add submission details	3.00818	3.00818	☐
submission_xgboost_1000_n_estimators.csv 6 minutes ago by Norman Fung add submission details	2.97296	2.97296	☐
submission_xgboost_1500_n_estimators.csv 9 minutes ago by Norman Fung add submission details	2.95333	2.95333	☐
submission_xgboost_2000_n_estimators.csv 9 minutes ago by Norman Fung add submission details	2.94650	2.94650	☐

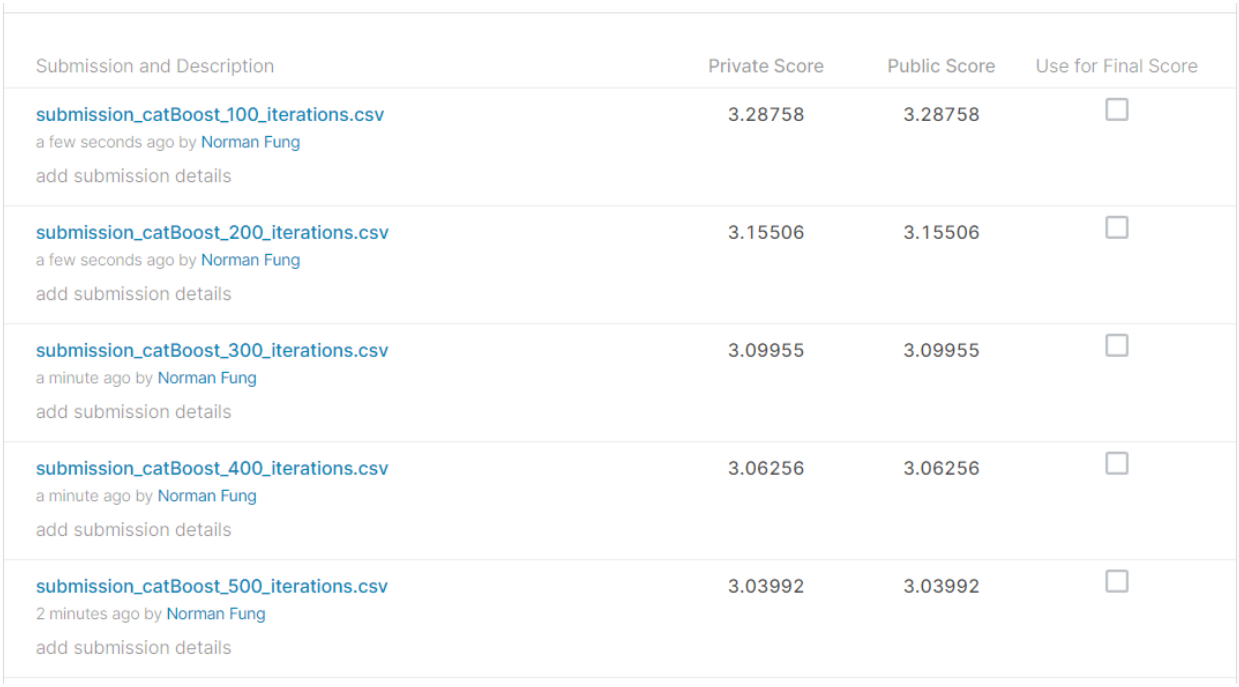
Fig.24 Experiment with different n_estimators for XGBoost.

The n_estimators parameter refers to the number of Gradient Boosted Trees which is equivalent to the number of boosting rounds. We can see that the model accuracy as we increase the number of boosting rounds, however, increasing the n_estimators also increases the time required for training significantly, therefore the maximum value we have experimented with is 2000.

Submission and Description	Private Score	Public Score	Use for Final Score
submission_xgboost_maxDepth_3.csv just now by Norman Fung add submission details	3.23397	3.23397	☐
submission_xgboost_maxDepth_5.csv a few seconds ago by Norman Fung add submission details	3.03519	3.03519	☐
submission_xgboost_maxDepth_8.csv a minute ago by Norman Fung add submission details	2.94650	2.94650	☐
submission_xgboost_maxDepth_10.csv a minute ago by Norman Fung add submission details	2.93355	2.93355	☐

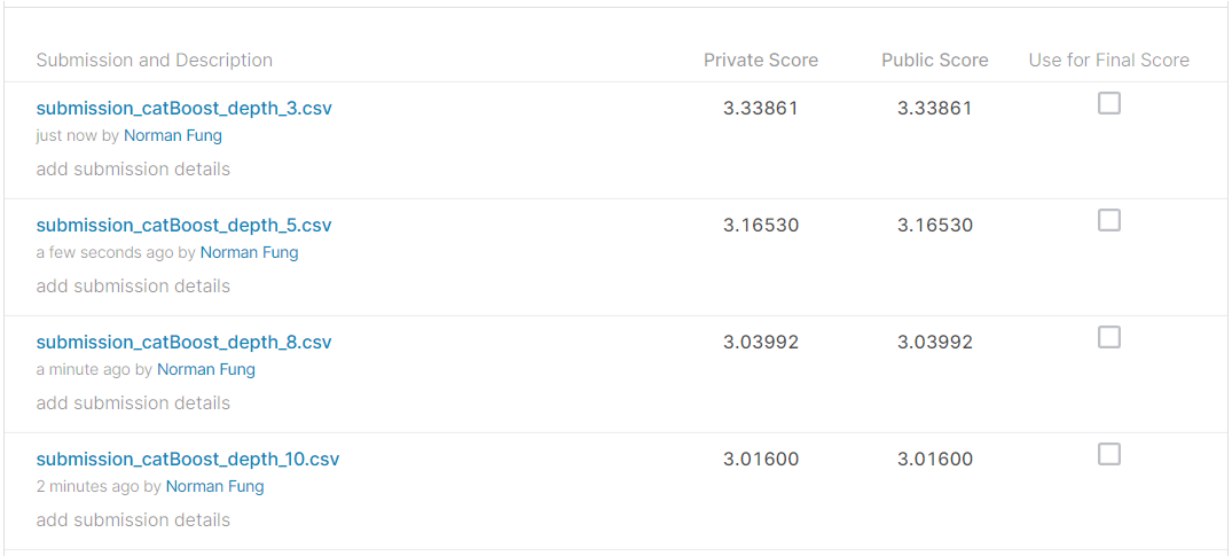
Fig.25 Experiment with different max_depth for XGBoost.

Similar to the n_estimators parameter, when we increase the max_depth, we can also observe an increase in the performance of our model. However, the drawback is an exponential increase in training time.



Submission and Description	Private Score	Public Score	Use for Final Score
submission_catBoost_100_iterations.csv a few seconds ago by Norman Fung add submission details	3.28758	3.28758	☐
submission_catBoost_200_iterations.csv a few seconds ago by Norman Fung add submission details	3.15506	3.15506	☐
submission_catBoost_300_iterations.csv a minute ago by Norman Fung add submission details	3.09955	3.09955	☐
submission_catBoost_400_iterations.csv a minute ago by Norman Fung add submission details	3.06256	3.06256	☐
submission_catBoost_500_iterations.csv 2 minutes ago by Norman Fung add submission details	3.03992	3.03992	☐

Fig.26 Experiment with different number of iterations for CatBoost.



Submission and Description	Private Score	Public Score	Use for Final Score
submission_catBoost_depth_3.csv just now by Norman Fung add submission details	3.33861	3.33861	☐
submission_catBoost_depth_5.csv a few seconds ago by Norman Fung add submission details	3.16530	3.16530	☐
submission_catBoost_depth_8.csv a minute ago by Norman Fung add submission details	3.03992	3.03992	☐
submission_catBoost_depth_10.csv 2 minutes ago by Norman Fung add submission details	3.01600	3.01600	☐

Fig.27 Experiment with different depths for CatBoost.

A similar approach has been done with CatBoost where we experimented with the different number of iterations and depths.

The best results of each different model are compiled in the following table:

Submissions	Private scores
XGBoost Regressor	- 2.93355
LightGBM Regressor	- 2.88431
CatBoost Regressor	- 3.01600

Table.3 Best private scores achieved for different models.

Ensemble Learning

After we have obtained the best models for each Regressor algorithm, our group explored whether we can leverage the concept of ensemble learning by combining the result of the 3 regressor algorithms. Since this is a regression problem, we will be using the mean predicted value as well as a weighted average of the 3 best models.

The mean predicted value is calculated as follows:

$$predicted\ value\ = \frac{resultA + resultB + resultC}{3}$$

where resultA , resultB , resultC represents the predicted values from the Catboost, XGBoost and LightGBM respectively.

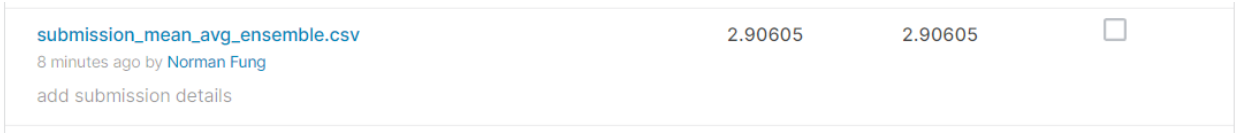


Fig.28 Result obtained from the mean predicted value approach.

The weighted average value is calculated as follows:

$$predicted\ value\ = \alpha\ *\ resultA\ +\ \beta\ *\ resultB\ +\ \gamma\ *\ resultC$$

where α, β, γ are pre-defined coefficients.

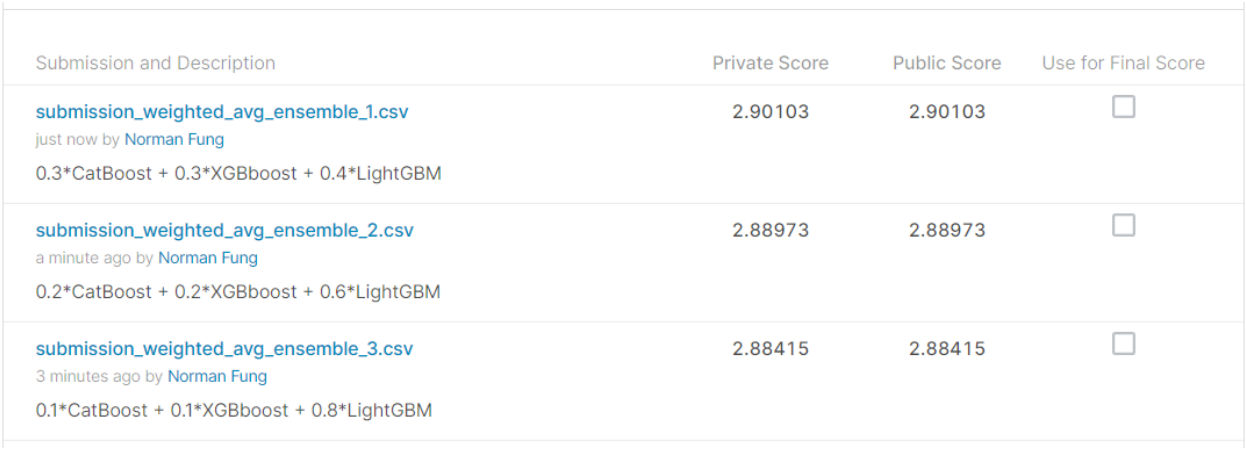


Fig.29 Result obtained from the weighted average approach.

Finally, we are able to obtain our best score of 2.88415 for the competition using the weighted average approach.

Conclusion

In conclusion, we have learned that there are many various ways to improve the performance of our models. For our scenario, we were provided a huge dataset but with limited features. Therefore, it is important to rely on features engineering to create additional features from the given dataset. It is also important to have a good understanding of the domain so that good features can be extracted from the dataset. For our scenario, we relied upon domain knowledge such as fix pick-up rate from the Airport as well as the maximum passenger limit of a regular taxi to create better features for our model.

Since we have limited features in our dataset, we have invested a substantial amount of time experimenting with different parameters for our different models. From those experiments, we have learned how different parameters such as learning rate, depth of trees, number of boosting rounds can help increase the accuracy of the model.

In addition, we produced the best score with the help of ensemble learning by combining the results from different gradient boosting libraries. However, even though the best score was obtained using the weighted average approach, the result was not significant when compared to the best result obtained from the LightGBM model. Hence, we believe that using a single model approach can still be a reliable way to approach a problem.