New York City Taxi Fare Prediction

AI & ML Associate Learning Path
Practical Project

POD 2:

Aleksandr Vystoropskyi
Kevin Mochi
Mike Leske
Selim Budakoglu

I - Definition

Project Overview

New York City is world-famous for its amount of yellow cabs. With the rise of market disrupters like Uber and Lyft it is essential to upfront be aware of the expected ride cost to select the 'best' option in terms of monetary benefit for the user. In August 2018 Kaggle hosted a competition together with its partners Google and Coursera aiming to predict regular taxi fares in New York City. [1]

As part of the competition a training dataset of 55 million historic taxi rides (2009 - 2015) has been provided (see section Data Exploration for further dataset information).

Problem Statement

The problem statement of this Kaggle competition is defined as "predicting the fare amount (inclusive of tolls) for a taxi ride in New York City given the pickup and drop-off locations". [1] The predicted fare amount is to be provided as US Dollar and can serve as an estimate for future taxi rides. A potential use case for such a prediction algorithm is the integration into online navigation services which – in addition to the routing information – intend to provide an estimate of taxi fare costs to a given route inside New York City.

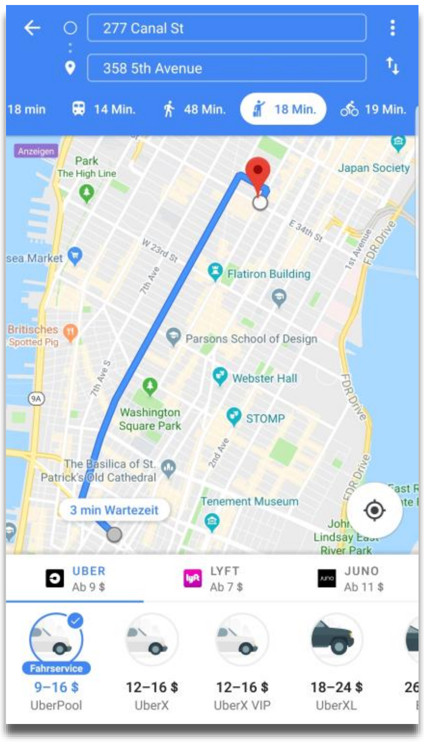


Figure 1 Google Maps providing Uber fare estimation

The outlined problem fits well into the domain of supervised regression problems for which a multitude of potential solutions exist including classical (multivariate) linear regression, decision trees, artificial neural networks, gradient boosting machines and many more.

As part of this project one algorithm will be implemented and accuracy will be checked.

Metrics

As the problem statement points into the direction of a regression analysis, the competition evaluation is based on the root mean-squared error (RMSE) [2][3]. RMSE is well-suited for this problem statement as it measures the difference between the predicted taxi fares and the real taxi ride fares provided as part of the training dataset. In this sense the taxi ride fare is the regression's dependent variable.

RMSE is calculated by:

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

A smaller RMSE means the predicted values are close(r) to the real fares of the training set than the predictions of a model with a resulting larger RMSE.

Therefore, the solution seeks to minimize the RMSE.

II - Analysis

Data Exploration

The dataset [4] to be used for this project is provided by Kaggle and its partners Google and Coursera. It contains two essential files:

1. A train.csv file with a training set of 55 million taxi rides and
2. A test.csv file with a test set of approx. 10,000 taxis rides.

For each observation (individual taxi rides) the following parameters are provided:

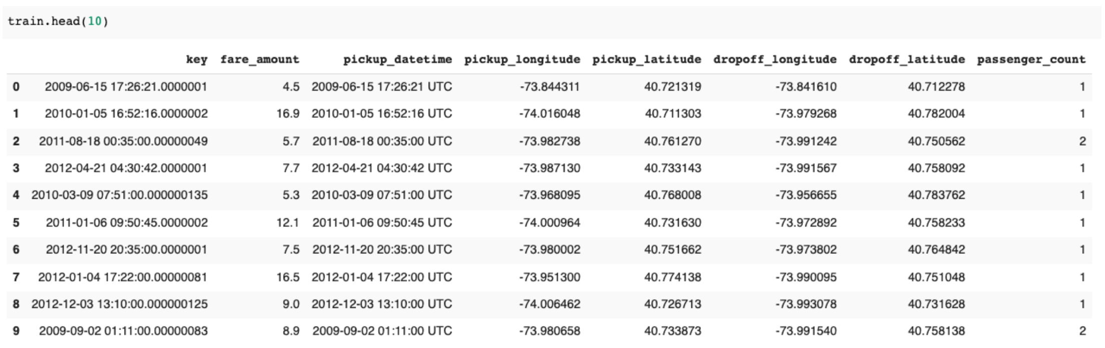
- key
- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- passenger_count

While the official training set (train.csv) contains the final taxi ride fare, the official test set used for the project does not provide this final insight.

Variable	Dataset "train"	Dataset "test"	Data Type
key	Yes	Yes	String representing pickup_datetime
fare_amount	Yes	No	Recorded fare of taxi ride
pickup_datetime	Yes	Yes	Timestamp of the taxi ride start.
pickup_longitude	Yes	Yes	GPS longitude coordinate where taxi ride started
pickup_latitude	Yes	Yes	GPS latitude coordinate where taxi ride started
dropoff_longitude	Yes	Yes	GPS longitude coordinate where taxi ride ended
dropoff_latitude	Yes	Yes	GPS latitude coordinate where taxi ride ended
passenger_count	Yes	Yes	Number of passengers in the taxi

Table 1 Structure of provided dataset

Note:
Importing the complete training dataset of 55 million observations is very memory demanding. Hence, the following data explorations and model operations will be executed on subsets of the training data, i.e. 500K observations.



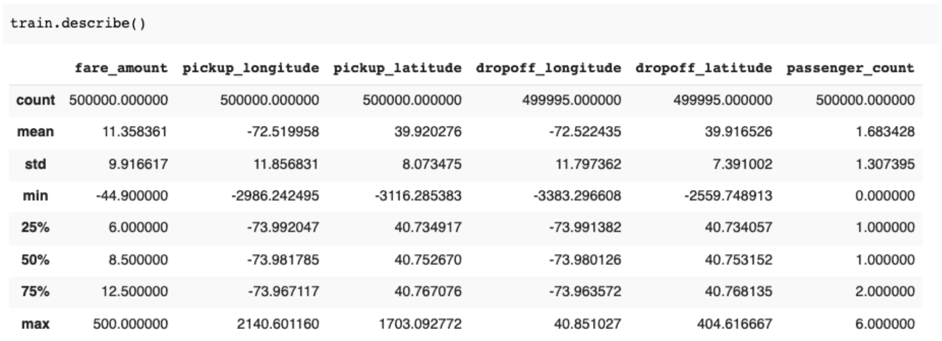
```
train.head(10)
```

	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
5	2011-01-06 09:50:45.0000002	12.1	2011-01-06 09:50:45 UTC	-74.000964	40.731630	-73.972892	40.758233	1
6	2012-11-20 20:35:00.0000001	7.5	2012-11-20 20:35:00 UTC	-73.980002	40.751662	-73.973802	40.764842	1
7	2012-01-04 17:22:00.00000081	16.5	2012-01-04 17:22:00 UTC	-73.951300	40.774138	-73.990095	40.751048	1
8	2012-12-03 13:10:00.000000125	9.0	2012-12-03 13:10:00 UTC	-74.006462	40.726713	-73.993078	40.731628	1
9	2009-09-02 01:11:00.00000083	8.9	2009-09-02 01:11:00 UTC	-73.980658	40.733873	-73.991540	40.758138	2

Figure 2 Sample output showing the content of the training dataset

Figure 3 provides an initial statistical overview of the reduced training dataset of 500K observations. Several observations for data cleaning immediately become visible:

- fare_amount:
    - Negative fares are included in dataset
    - Max fare is much higher than mean and likely represents outliers or extreme rides
- pickup_longitude/pickup_latitude:
    - min & max values represent extreme outliers from mean and 25%/50%/75% percentiles
- dropoff_longitude/dropoff_latitude:
    - count is only 5, representing missing data
    - min & max values represent extreme outliers from mean and 25%/50%/75% percentiles



```
train.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	500000.000000	500000.000000	500000.000000	499995.000000	499995.000000	500000.000000
mean	11.358361	-72.519958	39.920276	-72.522435	39.916526	1.683428
std	9.916617	11.856831	8.073475	11.797362	7.391002	1.307395
min	-44.900000	-2986.242495	-3116.285383	-3383.296608	-2559.748913	0.000000
25%	6.000000	-73.992047	40.734917	-73.991382	40.734057	1.000000
50%	8.500000	-73.981785	40.752670	-73.980126	40.753152	1.000000
75%	12.500000	-73.967117	40.767076	-73.963572	40.768135	2.000000
max	500.000000	2140.601160	1703.092772	40.851027	404.616667	6.000000

Figure 3 Statistical overview over 500K observations

Algorithm and Technique

The New York City Taxi Fare Prediction Kaggle competition presents itself as a classical regression type of problem, which is a well-studied domain with several potential solutions available ranging from classical Machine Learning (ML) algorithms to more sophisticated Deep Learning (DL) architectures based on Artificial Neural Networks (ANN). As described in the Problem Statement section, the provided training data is labelled with typical taxi ride variables like pickup and dropoff coordinates and the number of passengers as well as the "label" in terms of cost of the ride ('fare_amount'). Therefore, the solution to this prediction problem can be targeted using Supervised Learning algorithms.

The fitness of the following specific algorithm/solution will be evaluated:

- Light Gradient Boosting Machines (Light GBM) in Python using Light GBM and the scikit-learn libraries.

Light Gradient Boosting Machines (Light GBM)

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So, when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.

Before is a diagrammatic representation by the makers of the Light GBM to explain the difference clearly.

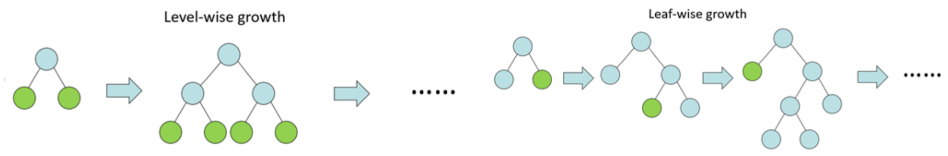


Figure 4 Leaf-wise growth for Light GBM

Advantages of Light GBM;

- Faster training speed and higher efficiency: Light GBM use histogram-based algorithm i.e. it buckets continuous feature values into discrete bins which fasten the training procedure.
- Lower memory usage: Replaces continuous values to discrete bins which result in lower memory usage.
- Better accuracy than any other boosting algorithm: It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy. However, it can sometimes lead to overfitting which can be avoided by setting the max_depth parameter.
- Compatibility with Large Datasets: It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST.

Benchmark

When the Kaggle competition was launched a starter solution using a Simple Linear Model was provided to establish a baseline performance. [10] This solution calculates the taxi travel distance from the provided pickup and drop-off longitude/latitude values and subsequently executes numpy matrix multiplication operations.

This Simple Linear Model results in a competition evaluation metric (root-mean-squared-error, see section Metrics) of $ **5.74**.

The solutions to be implemented as part of this project will be measured and compared against this baseline performance value.

III - Methodology

Data Cleansing and Feature Engineering

As already indicated in the section for data analysis feature engineering and data cleaning was necessary.

So, for that purpose following activities taken;

- Check for NaNs and drop them (if any)
- Check for outliers and drop them (if any)
- Type conversion of relevant fields
- Feature engineering

Key and pickup_datetime columns were formatted from object to datetime format for further processing.

The following features have been engineered for the original training and test dataset:

Name	Comment
"Year", "Month", "Date", "Hour", "Day of Week"	Learn time-based dependencies in the training taxi ride fares. These features were created from the "pickup_datetime" provided with the original dataset.
"H_Distance"	Calculate the taxi ride distance based on the Haversine formula [11] based on the provided GPS coordinates for pickup and dropoff.

The following data cleaning was applied to the original training dataset:

Name	Comment
(any)	Assumed that any column which does not have value was no use, so they were dropped.
drop	NA
"pickup_longitude", "dropoff_longitude"	Longitude values limited to the interval [-180, 180]. This effectively excluded extreme outliers and wrong information.
"pickup_latitude", "dropoff_latitude"	Latitude values limited to the interval [-90, 90]. This effectively excluded extreme outliers and wrong information.
"passenger_count"	Assumed that the Max capacity of a taxi is 6 passengers, any value above had to be considered an outlier so they were dropped.
"fare_amount"	Negative fare values may be due to a refund, typo or other error.
"distance_mile"	NA

Also following conditions considered as outliers and records removed from data sets accordingly.

1. Pickup latitude and pickup longitude are 0 but drop-off latitude and longitude are not 0, but the fare is 0.
2. Vice versa of point 1.
3. Pickup latitude and pickup longitude are 0 but drop-off latitude and longitude are not 0, but the fare is not 0.
4. Distance between pickup and drop-off (H_Distance) value is more than a 100 kms. In NYC it is unlikely that passengers would take cabs to travel more than a 100 kms.
5. Distance (H_Distance) value is 0 and fare is 0.
6. As in New York a minimum of $2.50 is charged per ride (Google search), fare values smaller than that can't be accurate for distances greater than 0 KM.

After data cleaning the training dataset still contains 490052 observations (instead of 500000).

```
train.shape
```

```
(490052, 14)
```

The data cleaning targeted removing either outlier observations or observations with wrong information. Observations with missing variables also have been removed. The algorithms were implemented using chunks of 500K observation samples, and the number of removed rows due to missing variable was negligibly low.

Hypothesis on Price Influencers

Assuming a direct correlation between fare - ride duration and fare - distance travelled, we investigated the influence of the additional factors on the final price:

- Does the number of passengers affect the fare?

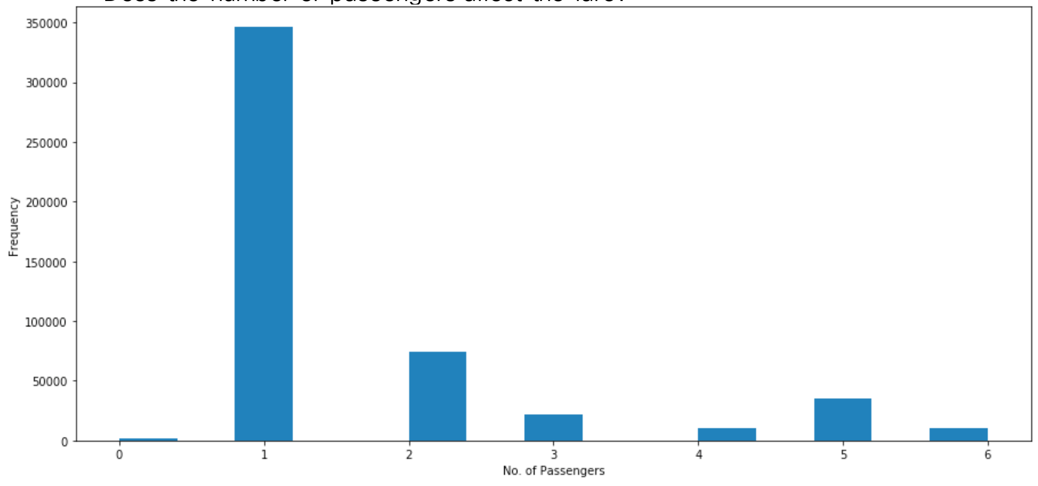


Figure 5 Frequency vs. Number of Passengers

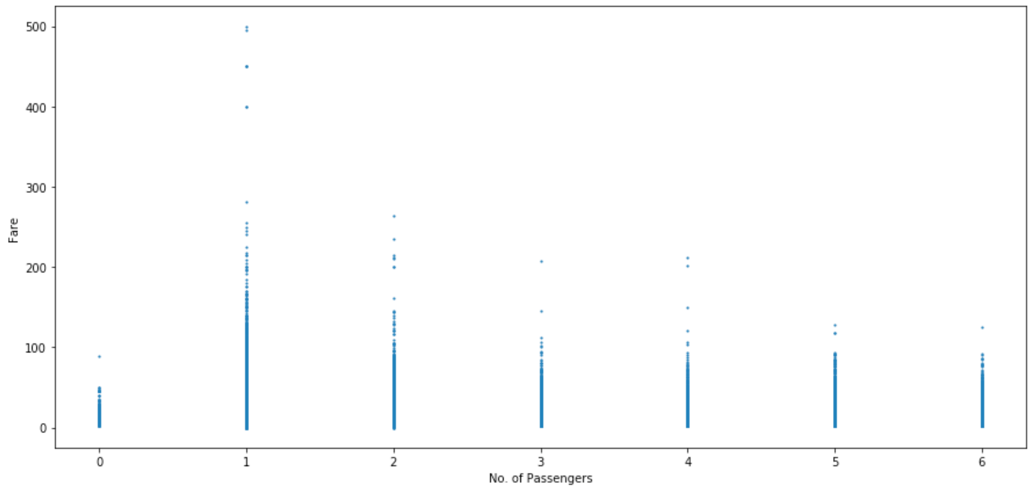


Figure 8 Fare vs. Number of Passengers

Above two graphs show that single passengers are the most frequent travelers, and the highest fare also seems to come from cabs which carry just 1 passenger.

- Does the date and time of pickup affect the fare?

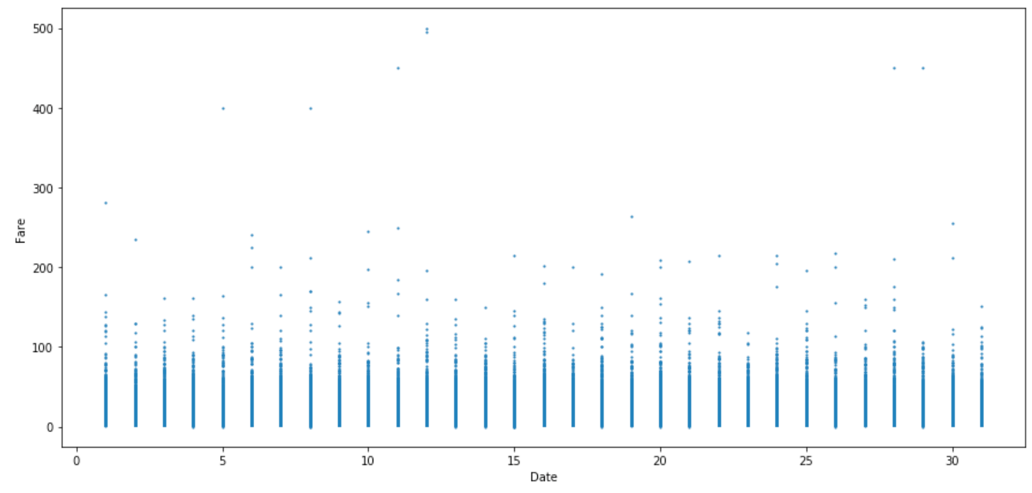


Figure 9 Fare vs. Date

The fares through the month mostly seem uniform, with the maximum fare received on the 12th.

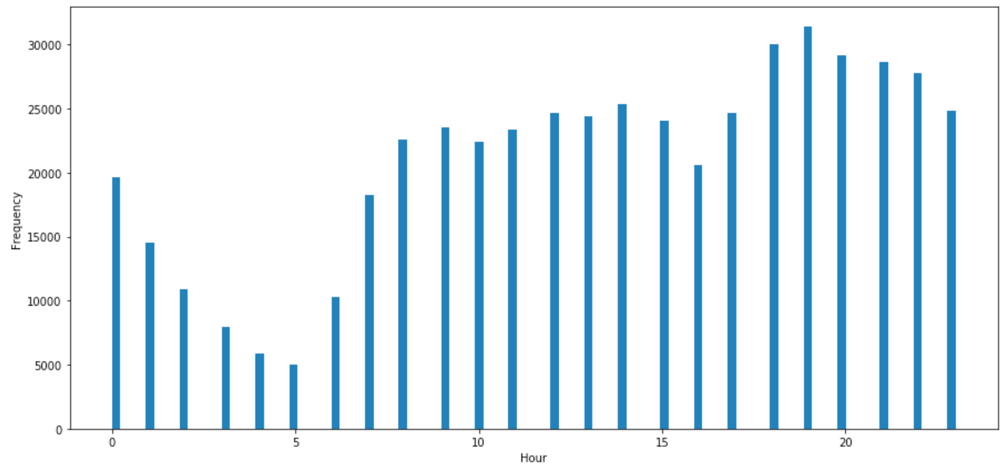


Figure 10 Frequency vs. Hour

The time of day definitely plays an important role. The frequency of cab rides seems to be the lowest at 5AM and the highest at 7PM.

- Does the day of the week affect the fare?

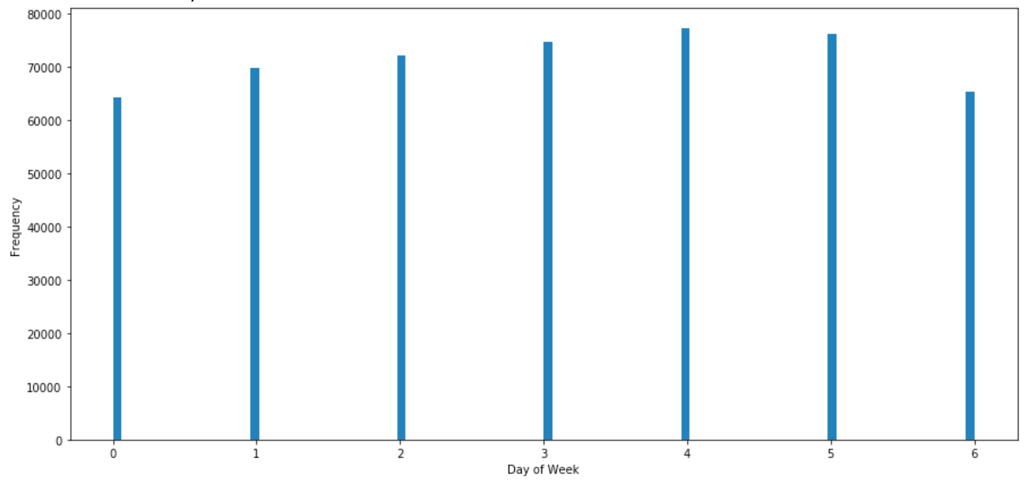


Figure 6 Frequency vs. Day of Week

The day of the week doesn't seem to influence the number of cab rides

IV - Models & Results

Light GBM Implementation & Random Forest

Modelling was performed with the two-stage pipeline; Model selection and hyperparameters optimization, model training and inference.

1. Train, validation and test splits creation

```
 from sklearn.model_selection import train_test_split
 from sklearn.metrics import mean_squared_error
 y=train['fare_amount']
 X=train.drop(['key','pickup_datetime','fare_amount'], axis = 1)
 X_train, X_val, y_train, y_val = train_test_split(X,y, test_size=0.25,random_state=123)
 X_test = test.drop(['key','pickup_datetime'], axis = 1)
```

```
Training set size: 367539
Validation set size: 122513
Testing set size: 9914
Training labels: 367539
Validation labels: 122513
```

2. Random Forest Model Evaluation

```
 from sklearn.ensemble import RandomForestRegressor
 rf = RandomForestRegressor(n_estimators = 100, random_state = 883,n_jobs=-1)
 rf.fit(X_train,y_train)
 rf_pred= rf.predict(X_val)
 rf_rmse=np.sqrt(mean_squared_error(rf_pred, y_val))
 rf_train_rmse=np.sqrt(mean_squared_error(rf.predict(X_train), y_train))
 rf_variance=abs(rf_train_rmse - rf_rmse)
```

```
RMSE for Random Forest is  4.119176670256823
Bias for Random Forest is  1.6374418814176757
Variance for Random Forest is  2.481734788839147
```

3. LightGBM Model Evaluation

```
 import lightgbm as lgb
 train_data=lgb.Dataset(X_train,label=y_train)
 param = {'num_leaves':60, 'objective':'regression'}
 param['metric'] = 'l2_root'
 num_round=5000
 cv_results = lgb.cv(param, train_data, num_boost_round=num_round, nfold=10,verbose_eval=
 20, early_stopping_rounds=20,stratified=False)
 lgb_bst=lgb.train(param,train_data,len(cv_results['rmse-mean']))
 lgb_pred = lgb_bst.predict(X_val)
 lgb_rmse=np.sqrt(mean_squared_error(lgb_pred, y_val))
 lgb_train_rmse=np.sqrt(mean_squared_error(lgb_bst.predict(X_train), y_train))
 lgb_variance=abs(lgb_train_rmse - lgb_rmse)
```

```
[20]     cv_agg's rmse: 4.63785 + 0.298731
[40]     cv_agg's rmse: 4.36825 + 0.307314
[60]     cv_agg's rmse: 4.30757 + 0.309269
[80]     cv_agg's rmse: 4.28275 + 0.310407
[100]    cv_agg's rmse: 4.27096 + 0.309713
[120]    cv_agg's rmse: 4.26317 + 0.310044
[140]    cv_agg's rmse: 4.25915 + 0.311033
[160]    cv_agg's rmse: 4.25741 + 0.312445
[180]    cv_agg's rmse: 4.25566 + 0.312578
[200]    cv_agg's rmse: 4.25582 + 0.312389

RMSE for Light GBM is  4.006892908586611
Bias for Light GBM is  3.5638131729309306
Variance for Light GBM is  0.4430797356556804
```

Evaluation Analysis

The higher bias for LGBM during simple model evaluations is normal since it expects the data to be in linear dependencies, while RF model is more flexible by combining the probabilities from several decision trees without linearity assumption. Also, Linear Regression as well as selected RMSE metric are both susceptible to outliers thus cleaning outliers from dataset is most important thing in the pipeline. The LGBM model with high bias cannot learn training data accurately as RF model, while an RF model with high variance essentially memorizes the training data and cannot generalize to new examples. Since the goal of machine learning is to generalize to new data and considering that RMSE is close to NY Taxi start fee, we want to select the LGBM model with low variance as bias-variance trade-off.

Model Selection

The design of good model aims both low bias and variance to avoid overfitting. Our regression model gave the RMSE of 4.00 on validation data but the bias is higher than Random Forest. On the other hand, the variance of LGBM model was 0.44 as compared to 2.48 in our Random Forest model. Since LightGBM has a comparable error rate to Random Forest and has a lower variance and runs faster than the latter, we will use LightGBM as our model for further analysis. The lower variance with LightGBM should give us higher confidence in the results we get in stage 2.
One of the options for improvement of the model accuracy and generalization capabilities could be usage of both LGBM and RF models as assembling pipeline and predictions averaging. This approach can leverage benefits from both models
.

Hyperparameters optimization for LightGBM:

```
from hyperopt import hp, tpe, fmin, Trials, STATUS_OK
from sklearn.metrics import mean_squared_error

def objective(space):
  clf = lgb.LGBMRegressor(
          objective = 'regression',
          n_jobs = -1, # Updated from 'nthread'
          verbose=1,
          boosting_type='gbdt',
          num_leaves=60,
          bagging_freq=20,
          subsample_freq=100,
          max_depth=int(space['max_depth']),
          subsample=space['subsample'],
          n_estimators=5000,
          colsample_bytree=space['colsample'])
          #metric='l2_root')
  eval_set=[( X_train, y_train), ( X_val,y_val)]
  clf.fit(X_train, np.array(y_train),
          eval_set=eval_set,eval_metric='rmse',
          verbose=0,
          early_stopping_rounds=20)
  pred = clf.predict(X_val)
  rmse = np.sqrt(mean_squared_error(y_val, pred))
  # print("SCORE:", rmse)
  return{'loss':rmse, 'status': STATUS_OK }

space ={
        'max_depth': hp.quniform("x_max_depth", 5, 30, 3),
        'subsample': hp.uniform ('x_subsample', 0.8, 1),
        'colsample':hp.uniform ('x_colsample', 0.3, 1)
    }
trials = Trials()
best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=100,
            trials=trials)
print(best)
```

```
100/100 [23:09<00:00, 11.95s/it, best loss: 3.982348254507461]
{'x_colsample': 0.9288663288431255, 'x_max_depth': 18.0, 'x_subsample': 0.9759443020815998}
```

Final training

```
train_data=lgb.Dataset(X_train,label=y_train)
param = {'num_leaves':60,
         'objective':'regression',
         'subsample':0.9759443020815998,
         'colsample_bytree':0.9288663288431255,
         'max_depth':18,
         'metric':'l2_root',
         'bagging_freq':20,
         'subsample_freq':100,
         }
num_round=5000
cv_results = lgb.cv(param, train_data, num_boost_round=num_round, nfold=10,verbose_eval=
20, early_stopping_rounds=20,stratified=False)
lgb_bst=lgb.train(param,train_data,len(cv_results['rmse-mean']))
lgb_pred = lgb_bst.predict(X_val)
lgb_rmse=np.sqrt(mean_squared_error(lgb_pred, y_val))
print("RMSE for Light GBM is ",lgb_rmse)
```

```
[20]     cv_agg's rmse: 4.62853 + 0.298662
[40]     cv_agg's rmse: 4.36293 + 0.307667
[60]     cv_agg's rmse: 4.31755 + 0.309071
[80]     cv_agg's rmse: 4.2929 + 0.310302
[100]    cv_agg's rmse: 4.28401 + 0.316174
[120]    cv_agg's rmse: 4.2749 + 0.316473
[140]    cv_agg's rmse: 4.26876 + 0.317479
[160]    cv_agg's rmse: 4.26706 + 0.317308
[180]    cv_agg's rmse: 4.26402 + 0.315536
[200]    cv_agg's rmse: 4.26204 + 0.313867
RMSE for Light GBM is  3.9934165613405415
```

Prediction

```
lgb_test = lgb_bst.predict(X_test)
print(lgb_test)
```

```
[10.46676864 10.7587567   4.67365769 ... 53.13798232 19.97074385
  6.79468332]
```

V – Conclusion

Reflection

In this project 2 different algorithms have been implemented including a classical a Random Forest and a boosted tree-based solution (LGBM) including optimization.

Integral to finding better-than-baseline predictions was to understand the problem domain and notice that certain pickup and drop-off locations in NYC result in special fares, e.g. airports. Therefore, additional features have been created in order to represent approximate length of the taxi ride in miles and how far/close either pickup or drop-off where to any of the airports. This allowed the algorithms to more easily find patterns in the data than by just processing the raw GPS coordinates.

We initially ran unoptimized versions of both algorithms that showed similar predictive quality (RMSE $4.11 for Random Forest, RMSE $4.00 for LGBM). The decision to select LGBM as primary model was based partly on the lower RMSE metric, but also due to the variance and bias analysis of both models. LGBM showed a much smaller variance, indicating overall better predictions, and the model bias of $3.56 can be understood as capturing the initial taxi fares ($2.50 + additional fees, e.g. $1 during rush hours).

The hyperparameter optimization of the LGMB model only resulted in negligible improvement.

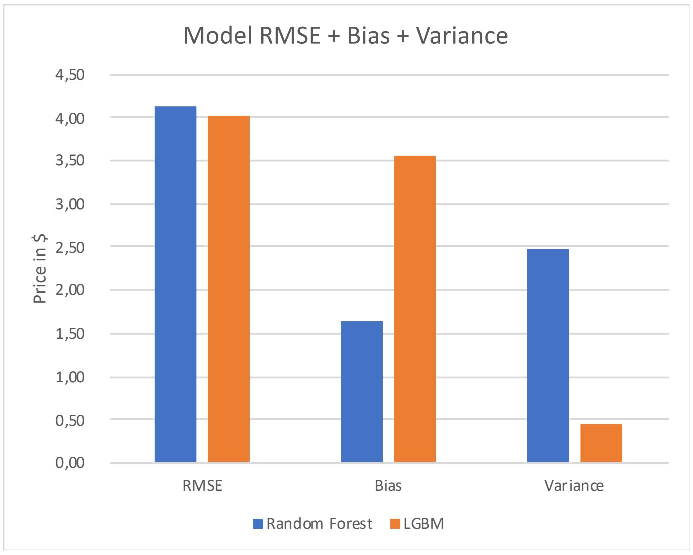


Figure X Algorithm Prediction RMSE + Bias + Variance

Improvement & recommendations

Certainly, one of the biggest drawbacks of both implementations shown in this report is the inability of training each algorithm on the complete training dataset of 55M observations due to memory constraints. Investing more time to find less memory-intensive representations might improve the situation.

Another commonly used method to improve machine learning algorithms is leveraging ensemble techniques, where the result of multiple algorithms is used to create a final prediction.

Lastly, completely different algorithms might be tried. The dense nature of the dataset, basically relying solely on GPS coordinates also shows potential for usage by a similarity algorithm of the Approximated Nearest Neighbour class, e.g. annoy.

VI - References

[1] https://www.kaggle.com/c/new-york-city-taxi-fare-prediction
[2] https://www.kaggle.com/c/new-york-city-taxi-fare-prediction#evaluation
[3] https://en.wikipedia.org/wiki/Root-mean-square_deviation
[4] https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/data
[5] http://scikit-learn.org/stable/modules/linear_model.html
[6] Book: Machine Learning Mastery - Discover XGBoost With Python!
[7] https://xgboost.readthedocs.io/en/latest/parameter.html
[8] https://keras.io/
[9] http://www.fast.ai/
[10] https://www.kaggle.com/dster/nyc-taxi-fare-starter-kernel-simple-linear-model
[11] https://en.wikipedia.org/wiki/Haversine_formula
[12] http://www.fast.ai/2018/04/29/categorical-embeddings/