Taxi Trip Duration Prediction

Kaige Yang
University College London
Kaige.yang.11@ucl.ac.uk

1 Work Flow

This project we follow the below work flow.

1. Project Overview
2. Data Loading
3. Data Understanding
4. Brain Storming
5. Data Cleaning
6. Data Transformation
7. Baseline Model
8. Exploratory Data Analysis
9. Feature Engineering
10. Baseline Model
11. Feature Selection
12. Model Selection
13. Conclusion

2 Project Overview

The task is to build a model that predicts the total ride duration of taxi trips in New York City.

Evaluation metric
The evaluation metric for this competition is Root Mean Squared Logarithmic Error (RMSLE).

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

where ϵ is the prediction error. n is the number of sample, p_i is the prediction of sample i, a_i is the target value of sample i. $\log(\cdot)$ is the natural logarithm.

3 Data Understanding

We have two datasets: training set and test set. In training set, each trip is described by the following features where "trip duration" is the target variable. In test set, "dropoff datetime" and "trip duration" are not available.

- id [sting] - a unique identifier for each trip.

- vendor id [int] - a code indicating the provider associated with the trip record.
- pickup datetime [string] - date and time when the meter was engaged.
- dropoff datetime [string] - date and time when the meter was disengaged.
- passenger count [int] - the number of passengers in the vehicle (driver entered value).
- pickup longitude [float] - the longitude where the meter was engaged.
- pickup latitude [float] - the latitude where the meter was engaged.
- dropoff longitude [float] - the longitude where the meter was disengaged.
- dropoff latitude [float] - the latitude where the meter was disengaged.
- store and fwdb flag [boolean] - 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" [int] - duration of the trip in seconds (available only in training set).

4 Brain Storming

Before diving into the data processing steps, it is better to have a deep thought on how to solve the problem. We ask fundamental questions and discuss potential solutions. We bring up the following questions:

- Q: What factors determine the trip duration?
  Intuitively, we suspect that the trip duration is determined by factors like: distance, time, location, driver, weather, datetime, passenger and directions. More specifically, the distance is obvious an important factor. Whether the fact that pickup/drop-time is peak time or off-peak time is also critical. In terms of the location, the middle town is obvious takes more time than rural areas. The skill/experience of drivers also affect the time largely. Weather means the raining, snowing, strong wind. Datetime indicates whether the day is holiday, weekday or weekend. The destination of passengers will also determine the duration. Finally, directions play an important role in terms of the direction of commute. Typically, in the morning, people commute from home to office and in the afternoon travel in the opposite direction.
- Q: What information do we need?
  It is ideal to have all the information discuss above.
- Q: What information is contained in the dataset?
  Some information desired are available explicitly such as datatime, location. More information are presented implicitly like distance, driver, direction, holiday, weekday, weekend, exact district, direction. Those information can be explicitly by transforming the existing variables. However, some information are missing completely. The weather is unknown, but the season can be obtain roughly from the datetime. The destination of passengers are not available. The skill and experience level of driver are also absent. These variables can be obtained from external dataset if exists or allowed by the task.
- Q: How to process the dataset to make the information more explicit?
  As the discuss above, we need to transform the existing variables to unleash more information.
  - datatime: season, holiday, weekend, weekday, peak/off-peak.
  - location: distance, districts and direction.
- Q: Any important information is missing in the current available dataset?
  As discussed above, information about weather, driver, passenger information is missing.
- Q: Do we need external dataset?
  Possible external data is the weather record of New York city covering the corresponding period.
- Q: What type of models are suitable for solving the problem?
  As this is a prediction problem, we have many models are suitable such as linear regression, random forest regression, neural network and xgboost.

5 Data Cleaning

In this step we deal with

- Missing Values
- Outliers

There is no missing in either training and test set

The target variable 'trip duration' shows an extremely skew distribution as the minimum duration is 1 second, while the maximum value is roughly 1000 hours. Trips last a few seconds are apparently outliers. Extreme long-time trips might be due to long distance or simply incorrect record. To have a better decision, we need check the associated distance. We take log of the trip duration variable to make it roughly a normal distribution.

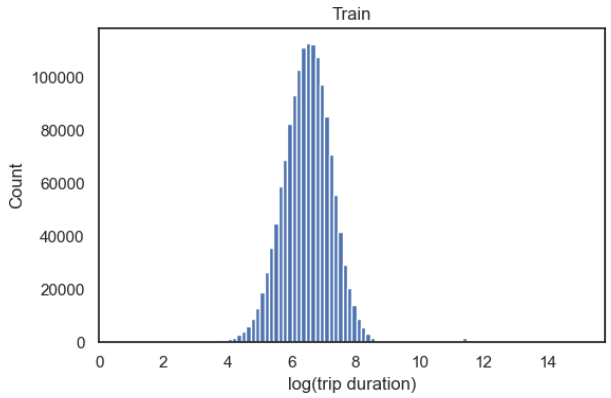

Figure 1: Log of Trip Duration

6 Basic Feature Engineering

The goal of this step is to transform all variables to numerical variables. We handle the following variables:

- Boolean: 'vendor id' is converted in to $0/1$.
- Datatime: 'pickup datetime', 'dropoff datetime' are onverted into year, month, day, week, weekday, holiday, hour, minute and second.

7 Baseline Model

As all variables are converted into numerical variables in the previous step, the dataset is ready for modelling. We fit a simple baseline model before more advanced feature engineering. This is for two reasons: 1), to see the expressive power of raw variables. 2), the performance acts as a benchmark to guide the feature engineering and model selection later.

This is a regression problem, we employ linear regression as the baseline model. We take the following steps:

- Standardization the input data
- Split the data into training and test data.
- Training linear regression model.
- Test the model on test data.

The train error is 0.77 and the test error is 0.78. This seems a reasonable performance as we expect linear regression. The model suffers some level of under-fitting. It means more expressive variables or more sophisticated models are needed.

```
used_columns = ['vendor_id',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
        'pickup_year',
       'pickup_month', 'pickup_day', 'pickup_weekday', 'pickup_weekofyear',
       'pickup_hour', 'pickup_minute', 'pickup_dt', 'pickup_week_hour', '
           log_trip_duration',]

mod_train = train[used_columns]
x = mod_train.iloc[:, :-1]
y = mod_train.iloc[:, -1]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

lr_model = LinearRegression()
lr_model.fit(x_train, y_train)

y_hat = lr_model.predict(x_test)
error = mean_squared_error(y_hat, y_test)
error = np.sqrt(error)
print('RMSLE:' error)
```

Figure 2: Linear Regression

8 Feature Engineering

We generate new features to make important information more explicit.

- Applying cluster techniques to location variables, we generate the district (area) of pickup and drop-off location.
- The distance between starting and end points can be found. Since we are in New York city, it seems largely suitable to use Manhattan distance.
- The direction is generated by Bearing direction, it tells us whether our trip started out for instance in the direction of North-West or South-East.
- From the datetime, we characterise the time as peak (8-9am, 4-6pm) and off-peak time.
- Another interesting variable could be whether the end point is airport. We introduce binary variable to indicate the fact.

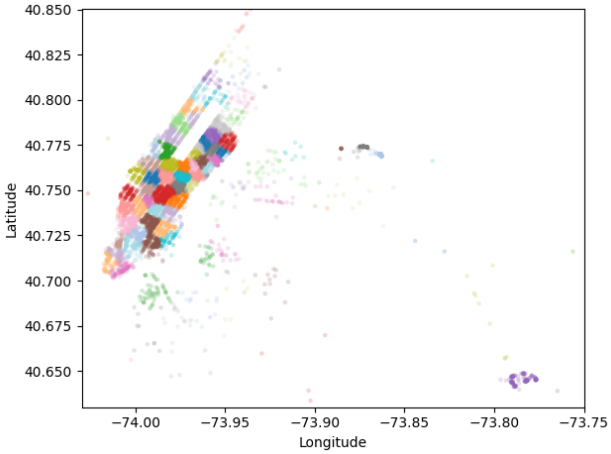

Figure 3: Clusters of Locations

After generating new features, we fit new data with linear regression. The training error is 0.64 and the test error is 0.74. It means that new generated features improve the performance.

9 Exploratory Data Analysis (EDA)

EDA is an important and the recommended first step prior to the training a machine learning model. In this section, we use some simply yet useful techniques that may help us to visually detect the presence of outliers, the distribution of the data, and the relationships between features.

- Box Plot: shows the variables distribution and detects the outliers.
- Scatter Plot: allows us to visualize the pair-wise correlations between the different features in the dataset.
- Correlation Matrix: to quantify the linear relationship between the features.

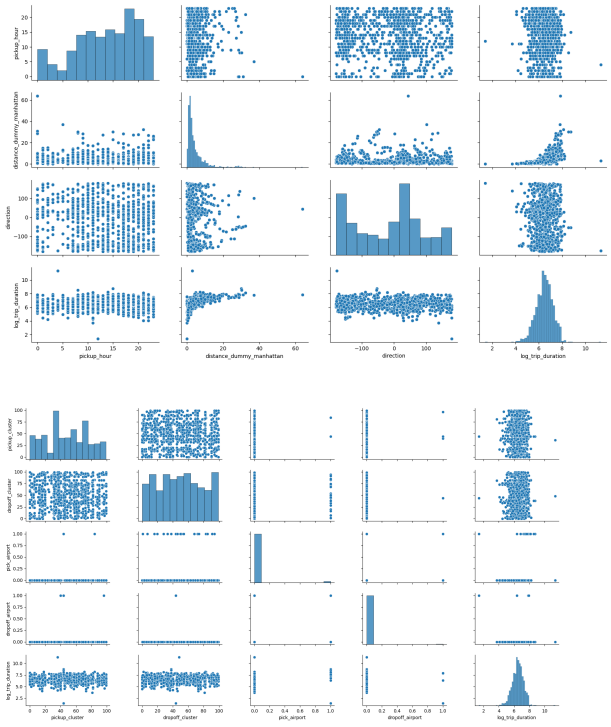

Figure 4: Scatter Plot

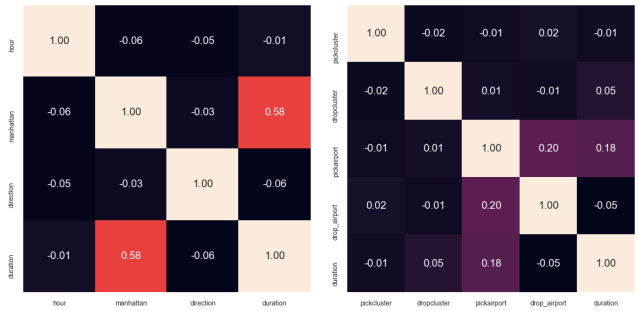

Figure 5: Correlation Matrix

10 Feature Selection

The next step is to decide which features are used in modelling. One may think it is better to use all available features. However, a large number of features might result in overfitting. There are several approaches to select features:

- Lasso Linear Regression.
- Feature importance of Random Forest
- Sequential Feature Selection (SBS)
- Explained Variance of PCA.

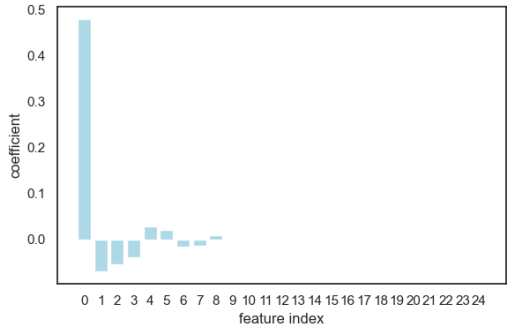

Figure 6: Lasso Coefficients

Using a random forest, we can measure feature importance as the averaged impurity decrease computed from all decision trees in the forest without make any assumptions whether out data is linearly separable or not.

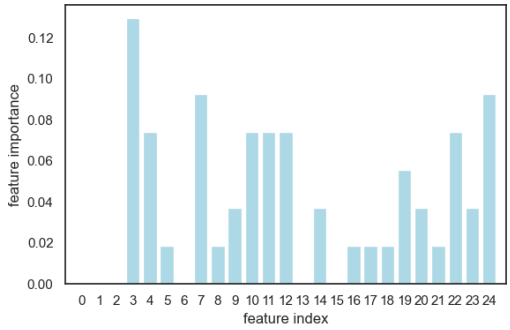

Figure 7: Random Forest Feature Importance

To obtain the explained variance of principle components, we first calculate the eigenvalues of the covariance matrix of input data. The explained variance of each principle component is the ratio between the eigenvalue and the sum of eigenvalues.

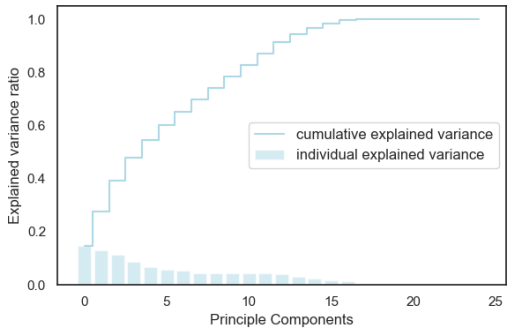

Figure 8: Explained Variance of PCA

The decision that whether use the principle components or original features are determined by the performance of modelling.

11 Model Selection

Finally, we are at the most existing step creating models. There are many models suitable for the regression problem at hand. We try the following models in sequel.

- Linear Regression
- Polynomial Linear Regression
- SVM Regression
- Decision Tree Regression
- Random Forest Regression
- Neural Network
- XGBoost
- LightGBM

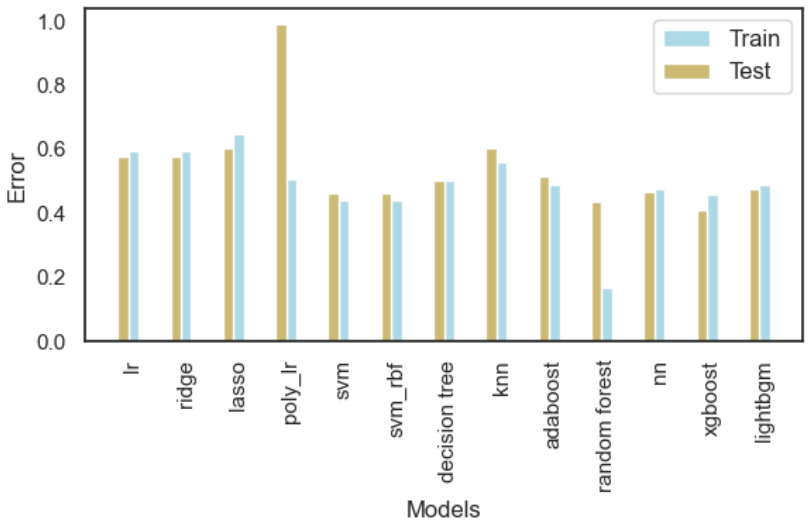

Figure 9: Prediction Error (RMSLE)

All models are trained using default hyper-parameters. Figure 9 shows that the best performs model is XGBoost. Simple models suffer under-fitting, while random forest suffers serious overfitting as the trees grow fully. The next step is to fine-tune promising models.

12 Model Fine-Tune

To push models to their limit, we need to fine-tune hyper-parameters. We focus on XGBoost, LightGBM and Neural Network models.

12.1 Fine-Tune XGBoost

When we observe high training accuracy, but low test accuracy, it is likely that we encountered overfitting problem. There are two general ways that we can control overfitting in XGBoost

- This first way to to directly control the model complexity. This includes Max-depth, min-child-weight and gamma
- The second way is to add randomness to make training robust to noise. This includes subsample and colssample-bytree. We can also reduce the stepsize η, Remember to increase num-round.

The final parameters are

- min-child-weight: 10
- eta: 0.01
- colsample-bytree: 0.3
- max-depth: 6
- subsample: 0.5
- lambda:1
- num-round: 2000

The training error is 0.37 and the test error is 0.39.

12.2 Fine-Tune LightGBM

LightGBM uses the leaf-wise tree growth algorithm, while many other popular tools use depth-wise growth. The leaf-wise algorithm converge much faster. However, the leaf-wise growth may be over-ftting if not used with the appropriate paramters. The following parameters control the model complexity.

- Num-leaves. This is the main parameter to control the complexity of the tree model. A larger num-leaves leads to highly complex model and induce over fitting.
- min-data-in-leaf. setting it to a large value can avoid growing too deep a tree, but may cause under-fitting.
- max-depth. We can use max-depth to limit the tree depth explicitly.

The final hyper-parameters are

- Learning rate: 0.1
- max-depth: 5
- num-leaves: 10
- feature-fraction: 0.5
- bagging-fraction: 0.5
- max-bin: 100
- n-round: 100

12.3 Fine-Tune Neural Network

Result in Figure 9 shows that the training and test error are similar, it means the neural network does not suffer overfitting. We first increase the complexity of the network to check any improvement. Then, deal with overfitting if exists.

- Learning rate: 0.001
- epoch num: 150
- Batch size: 32
- Layer number: 3
- Hidden nodes: 64, 32
- Activation function: ReLU

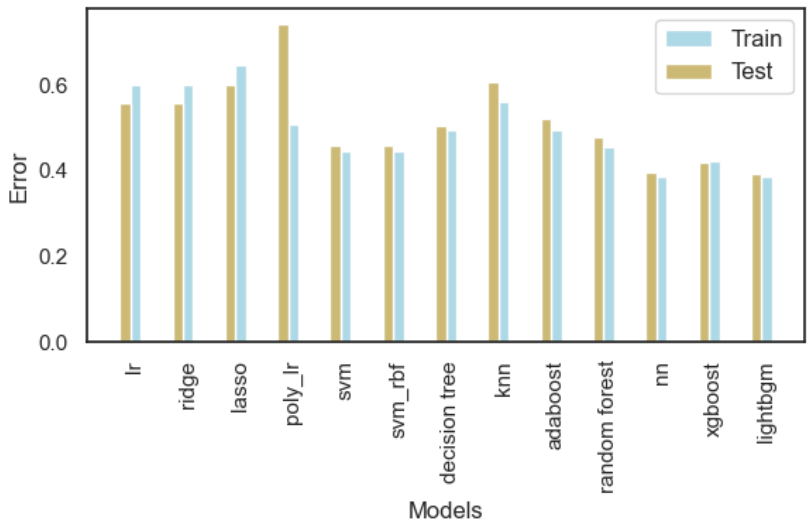

Figure 10: Prediction Error (RMSLE)

	lr	ridge	lasso	poly-lr	SVM	SVM-rbf	Decision Tree
Train	0.6	0.6	0.64	0.50	0.44	0.445	0.49
Test	0.56	0.55	0.60	0.74	0.45	0.45	0.50

	KNN	AdaBoost	Random Forest	NN	XGBoost	LightGBM
Train	0.56	0.49	0.45	0.38	0.42	0.38
Test	0.60	0.52	0.477	0.39	0.41	0.39

13 Further Improvement

Every steps conducted along the project has its own influence on the final performance. This leaves many possible combinations of decisions. For example,

- The data set can be replaced by PCA features then used to fit models.
- The cluster technique can be non-linear clustering.

14 Conclusion

In the project, we go through the whole data analytic work flow. The key steps that improve the performance (reduce error) are new feature generation and model hyper-parameter fine tuning. To have a more accurate estimation on overfitting problem, cross-validation can be used.