New York City Taxi Fare Prediction

Raymond Chen
January 27th, 2019

I. Definition

Project Overview

Tourists travel in an unfamiliar city may have a taxi ride, but sometimes they don't know reasonable taxi fare in this city. Few wicked taxi drivers may charge unreasonable fare by sneakily taking long route or adding initial charge.

If tourists have a tool to predict reasonable taxi fare based on some simple features like time, pickup location or dropoff location, they can notice unusual charge, take some actions and prevent from fraud. tourists make their budget on travel expense conveniently. For personal reason, when I have a business trip and have to make a budget in advance, I would use this tool to plan my means of transport. If I have sufficient budget, I can take a taxi for a more comfortable trip. Otherwise, maybe I need to take a train or a bus.

In this project, **New York City Taxi Fare Prediction** dataset provided in Kaggle is used. "train.csv "file will be used as train and valid dataset.

- **File description**
  - train.csv - Input features and target fare_amount values for the training set (about 55M rows).
  - test.csv - Input features for the test set (about 10K rows). Our goal is to predict fare_amount for each row.
  - sample_submission.csv - a sample submission file in the correct format (columns key and fare_amount). This file 'predicts' fare_amount to be $11.35 for all rows, which is the mean fare_amount from the training set.

Especially note that "test.csv" file didn't provide the groundtruth (fare_amount) and thus I can't directly use it as my test data (it's designed to be evaluated online in Kaggle competition.) Instead, I split exclusive part of the data in "train.csv" as my test data. Similar situation is in splitting data for sensitivity analysis. Here're all data I used in "train.csv":

- Data for Data Explore Analysis : Sample row from 0 to 150K.
- Train/Valid data : Sample row from 0 to 30K. 90% train data and 10% valid data with data shuffle.
- Data for sensitivity analysis : Sample row from 30K to 60K, three batches with 10K samples each.

- Test data : Sample row from 60K to 70K.

Problem Statement

Our target is to predict taxi fare in New York city, and we have several features like pickup GPS location, dropoff GPS location, or number of passengers, etc. to help us build a model to predict. This is a **regression problem** and we can express it as:

$$y = f(x_0, x_1, x_2, \dots)$$

where y is taxi fare for a ride, x_0, x_1, ... are features like time, GPS location, etc. of this ride, and f is a function or model we want to derive.

Given a dataset with many samples having ground truth taxi fare and features, we can apply different machine learning algorithms or even deep neural network to train a model based on them, i.e. finding some set of parameters that can describe the model mathematically. After model is developed, we can predict taxi fare for a given features.

After model is developed, we can evaluate the model performance using certain metric that can describe the value difference between predicted taxi fare y and ground truth taxi fare $\hat{y}$. Coefficient of determination R^2 will be the metric used in this project, it will be discussed in metrics section later.

Metrics

The metric used in this project is coefficient of determination R^2.

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{MSE}{Var(y)}$$

$$SSE = \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2, SST = \sum_{i=1}^{m} \left(\mu_y - y^{(i)}\right)^2$$

where SSE is sum squared error and SST is total sum of squared.

The values for R^2 range from $-\infty$ to 1, which captures the percentage of squared correlation between predicted and actual value of the target variable. Value between 0 and 1 indicates how well the model can explain the data. Higher R^2 is, better the model is. Note that a model can be given $R^2 < 0$, which means it's arbitrarily worse than always predicts the mean of the target variable.

There're also other metrics such as mean squared error (MSE) or mean absolute error (MAE) :

$$MSE = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$MAE = \frac{1}{m}\sum_{i=1}^{m}\left|\hat{y}^{(i)} - y^{(i)}\right|$$

They both directly penalize huge error and outliers would greatly affect their values. Since our data may exist several outliers (see Data Exploration part), they are not suitable for this problem. Instead, R^2 normalized MSE by variance of data $\boldsymbol{Var(y)}$ and this would lessen the effect of outliers. On the other hand, the values of R^2 always range from $-\infty$ to 1 and we can clearly know model that getting close to 1 is better. But for the other two metrics, the values depend on the data and we can easily judge the performance only based on their values. So, this is why R^2 is chosen as metric in this project.

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{MSE}{\boldsymbol{Var(y)}}$$

II. Analysis

Data Exploration

The data fields are summarized as ID, Input, and Output :

ID

- key - Unique string identifying each row in both the training and test sets.

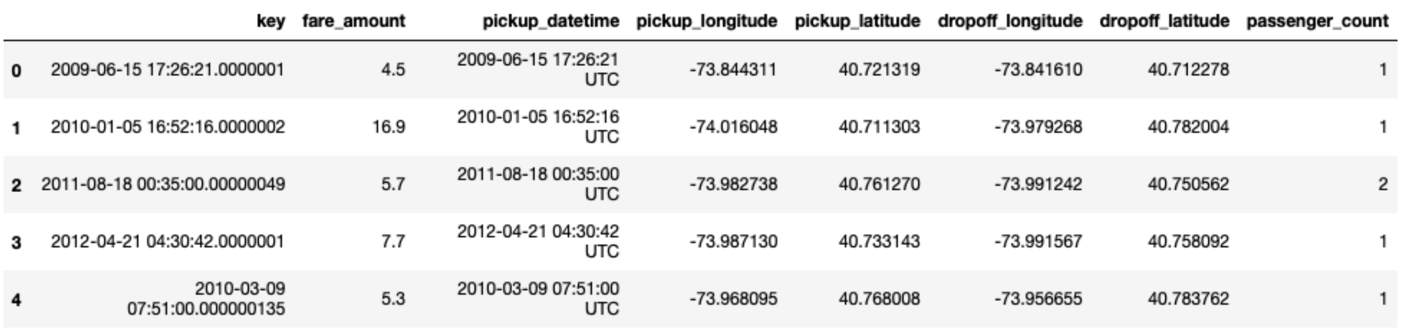
Input (features)

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

Output (taxi fare)

- fare_amout - float dollar amount of the cost of the taxi ride. This value is only in the training set

I loaded 200K samples from "train.csv" as my datum in data exploration phase, and had the head of data and statistics:



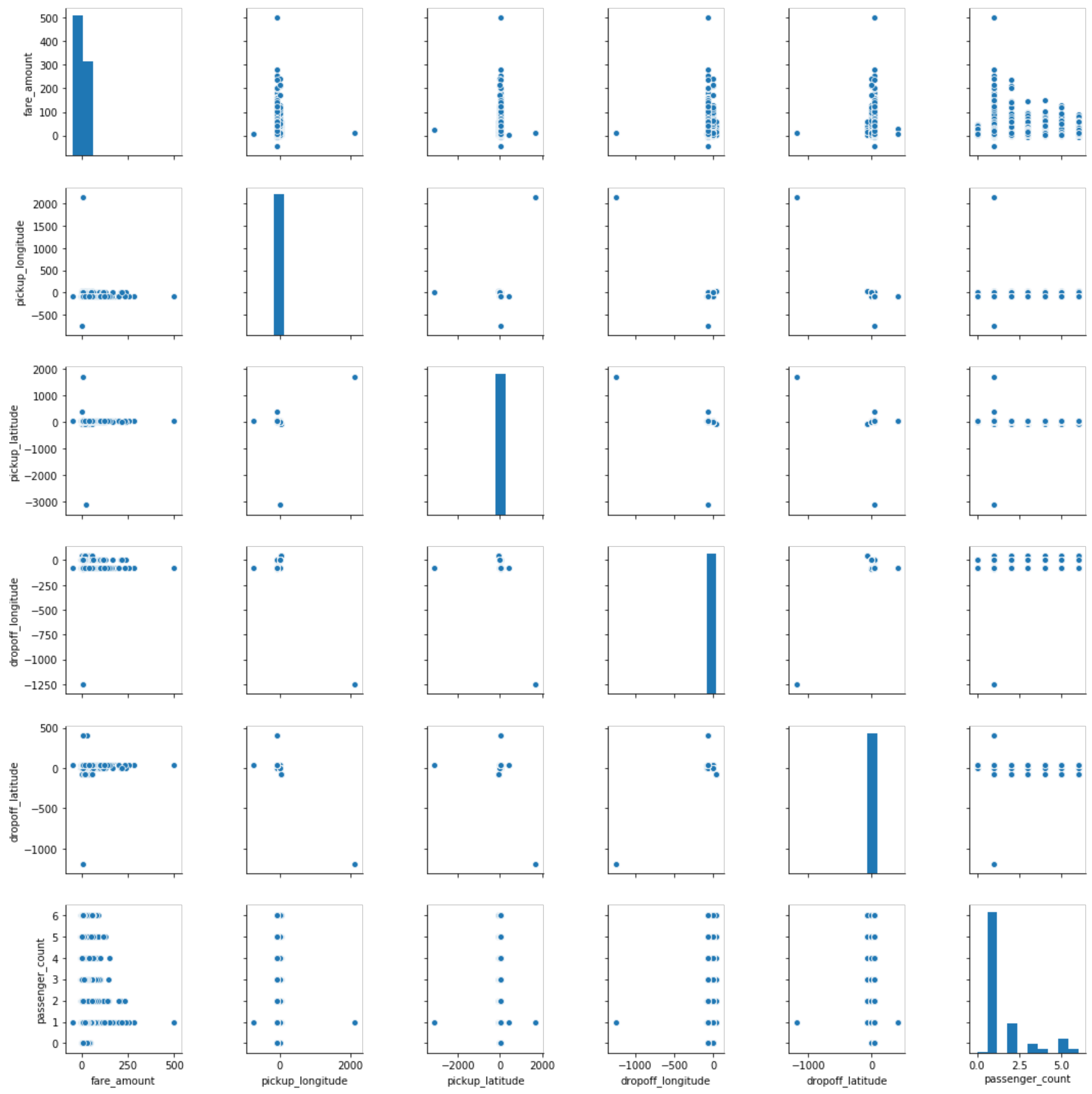
	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	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

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	11.342877	-72.506121	39.922326	-72.518673	39.925579	1.682445
std	9.837855	11.608097	10.048947	10.724226	6.751120	1.306730
min	-44.900000	-736.550000	-3116.285383	-1251.195890	-1189.615440	0.000000
25%	6.000000	-73.992050	40.735007	-73.991295	40.734092	1.000000
50%	8.500000	-73.981743	40.752761	-73.980072	40.753225	1.000000
75%	12.500000	-73.967068	40.767127	-73.963508	40.768070	2.000000
max	500.000000	2140.601160	1703.092772	40.851027	404.616667	6.000000

There're several abnormities to be fixed:
First, count values are not the same for all columns which means there're missing values (i.e. N.A. values). Second, minimum value of fare_amount shouldn't be less than $0 and the maximum value seems too high. Third, longitude value should be range from -90 to 90 and latitude value should be range from 0 to 360. Some minimum or maximum values of longitude or latitude are clearly out of range. Finally, one datatime field ('pickup_datetime') should be transformed to numerical for further processing.

Exploratory Visualization

Pair scatter plots are visualized below :



It’s clear that fare_amount have outliers <0 or >300, most latitude and longitude values distributed around 50% percentile and there’re some outliers needed to be removed.

Algorithm and Techniques

Linear Regression is used as benchmark because its simplicity and three candidate algorithms are chosen. Their default setting is listed below :

- Linear Regression (benchmark)
- Polynomial Regression : polynomial degree = 2
- Random Forest Regression : number of estimator = 50, random_state = 1

- Multiple Layer Perceptron Regression : hidden layer number = 3, hidden layer size = 10, random_state = 1, alpha = 1e-06, solver = lbfgs

These algorithms all have the following mathematical form :

$$y = f(X)$$

where y is our target, fare_amount, X is our final features, f is the algorithm model or function.
We can use scikit-learn library to feed our features and target to the above models and easily train them.

Mathematically, linear regression model can be expressed as :

$$y = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 + \cdots + \theta_n * x_n = h_\theta(\boldsymbol{x}) = \theta^T \cdot \boldsymbol{x}$$

However, the features x_i and model parameters θ_i form linear combination relationship and it could work great on linearly distributed data. But it may not a good model for nonlinear data. I used it as benchmark model and expected that it should perform poorly in my data in real world, which in most cases are nonlinear.

Polynomial regression model can be expressed as : (using single feature as example)

$$y = \theta_0 + \theta_1 * x + \theta_2 * x^2 + \cdots + \theta_d * x^d$$

It introduced nonlinearity by using $x^2,\ x^3\ \ldots\ x^d$, where d is the degree of polynomial model. It could work great on simple nonlinear data. If the hidden behaviors of data is not complex, it could get an efficient and great performance.

Random forest model is ensemble method of several decision tree. Decision tree regression use entropy and information gain concept to choose feature for data split. Entropy is defined as

$$entropy = -\sum_{i=1}^{n} p_i \, log(p_i)$$

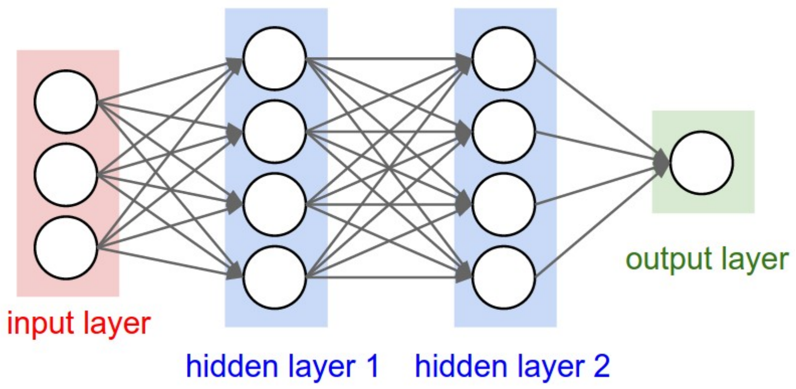
where p_i is the probability of choosing data belong to class i, and n is the number of multiple classes.
On the other hand, information gain is defined as

$$Information\ gain = entropy(parent) - (p_1 * entropy(child1) + p_2 * entropy(child2))$$

It chose feature with largest information gain to split data. After data is split as two child data, we can split its leaf data using the same method again.

We can sample subset of data and build several decision tree, predict target using these decision tree by voting or average. This is what random forest did. Random forest regression model used several weak predictors and ensemble them as a strong predictor. It typically works great on several complex data and may serve as a good model candidate for the problem.

In multiple layer perceptron, it used several neural network layers to form the model. Here's a simple example how it works :



Features x are fed in input layer (here we have 3 features), two hidden layers with 4 nodes are designed and finally only 1 node for output y (taxi fare). Here nodes are fully connected and form a linear combination relationship. We can insert nonlinear function (a.k.a. activation function) like ReLU function between each layer to introduce some nonlinearity into this model.

$$L1 = ReLU\left(h_1(x)\right)$$
$$L2 = ReLU\left(h_2(L1)\right)$$
$$y = ReLU\left(h_2(L2)\right)$$

We can define loss as mean squared error or mean absolute error and use gradient descent to train the model and get the model weights (parameters). Finally we can predict target y using these model weights and input features x. Different layers, nodes, and activation layers can be designed in this method.

Multiple layer perceptron regression is also a good model that given neural network structure and let data learn the weights. It's prominent for complex data and may capture hidden behaviors of data that is not easily found.

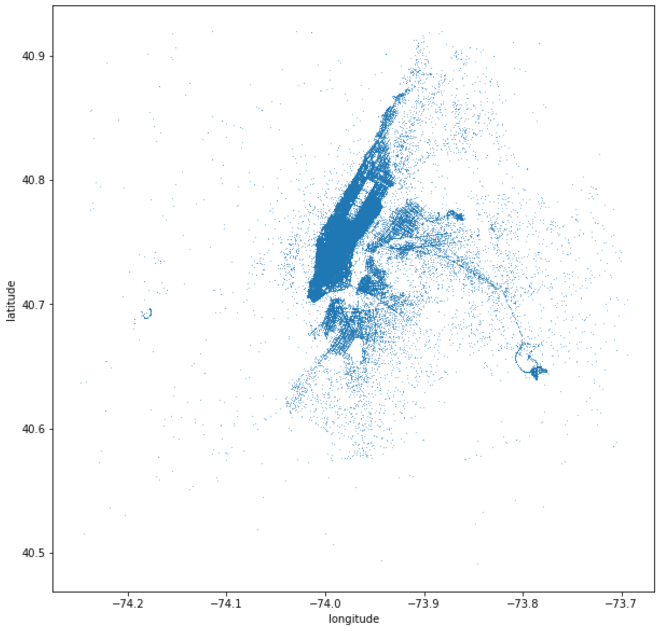
Benchmark

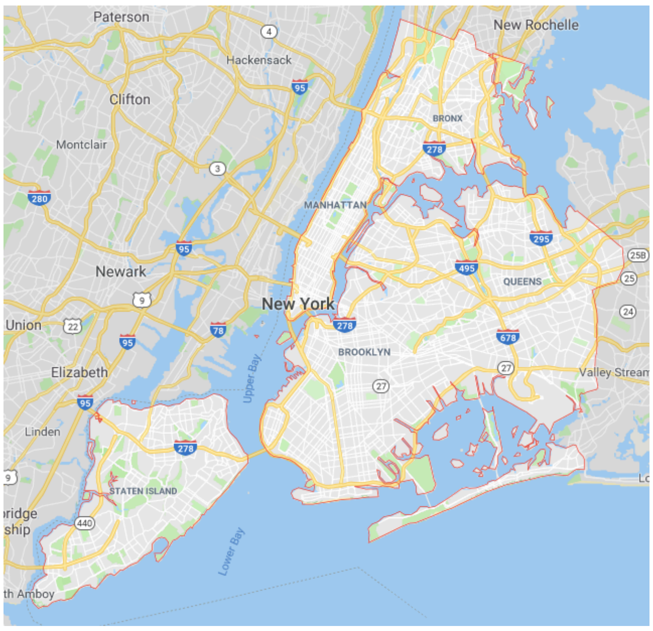
After all models are trained, the best model will be chosen to compare to benchmark model based on R^2 metric. Test data will be evaluated R^2 using the two models, respectively. Higher R^2 score near 1 means a good model. It should be expected that the best model outperform the benchmark model.

III. Methodology

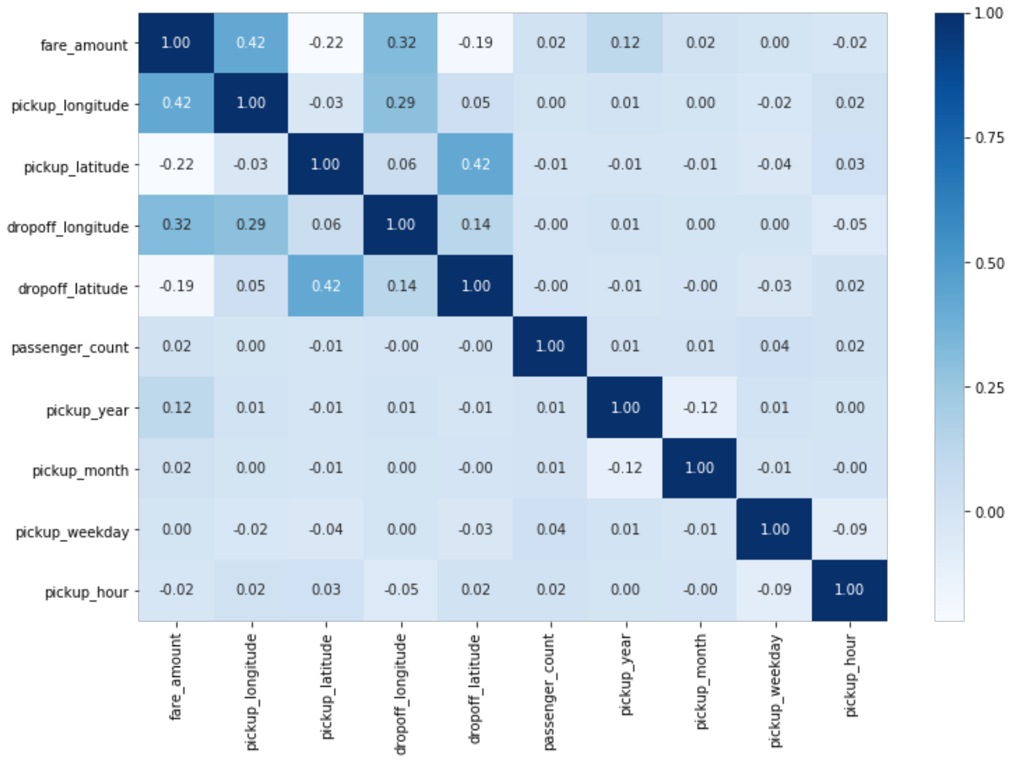
Data Preprocessing

First, I removed missing data and GPS outliers, and had a scatter plot based on pickup/dropoff latitude and longitude. I can see GPS locations distributed in New York city by comparing to New York city map from Google.



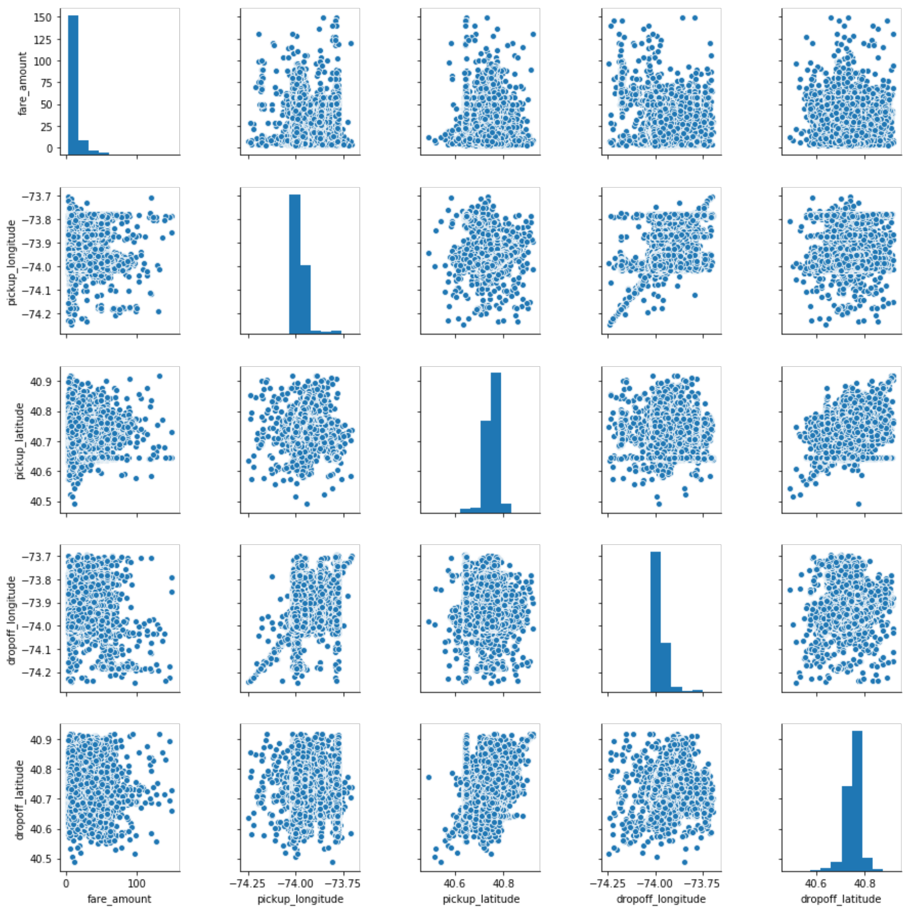


Second, I removed 'fare_amount 'outliers (<$2.5 or >$150), transform 'pickup_datetime' to four numerical columns 'pickup_year', 'pickup_month', 'pickup_weekday', and 'pickup_hour'. Afterwards, I compared feature pair correlation :



	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_year	pickup_month	pickup_weekday	pickup_hour
fare_amount	1.00	0.42	-0.22	0.32	-0.19	0.02	0.12	0.02	0.00	-0.02
pickup_longitude	0.42	1.00	-0.03	0.29	0.05	0.00	0.01	0.00	-0.02	0.02
pickup_latitude	-0.22	-0.03	1.00	0.06	0.42	-0.01	-0.01	-0.01	-0.04	0.03
dropoff_longitude	0.32	0.29	0.06	1.00	0.14	-0.00	0.01	0.00	0.00	-0.05
dropoff_latitude	-0.19	0.05	0.42	0.14	1.00	-0.00	-0.01	-0.00	-0.03	0.02
passenger_count	0.02	0.00	-0.01	-0.00	-0.00	1.00	0.01	0.01	0.04	0.02
pickup_year	0.12	0.01	-0.01	0.01	-0.01	0.01	1.00	-0.12	0.01	0.00
pickup_month	0.02	0.00	-0.01	0.00	-0.00	0.01	-0.12	1.00	-0.01	-0.00
pickup_weekday	0.00	-0.02	-0.04	0.00	-0.03	0.04	0.01	-0.01	1.00	-0.09
pickup_hour	-0.02	0.02	0.03	-0.05	0.02	0.02	0.00	-0.00	-0.09	1.00

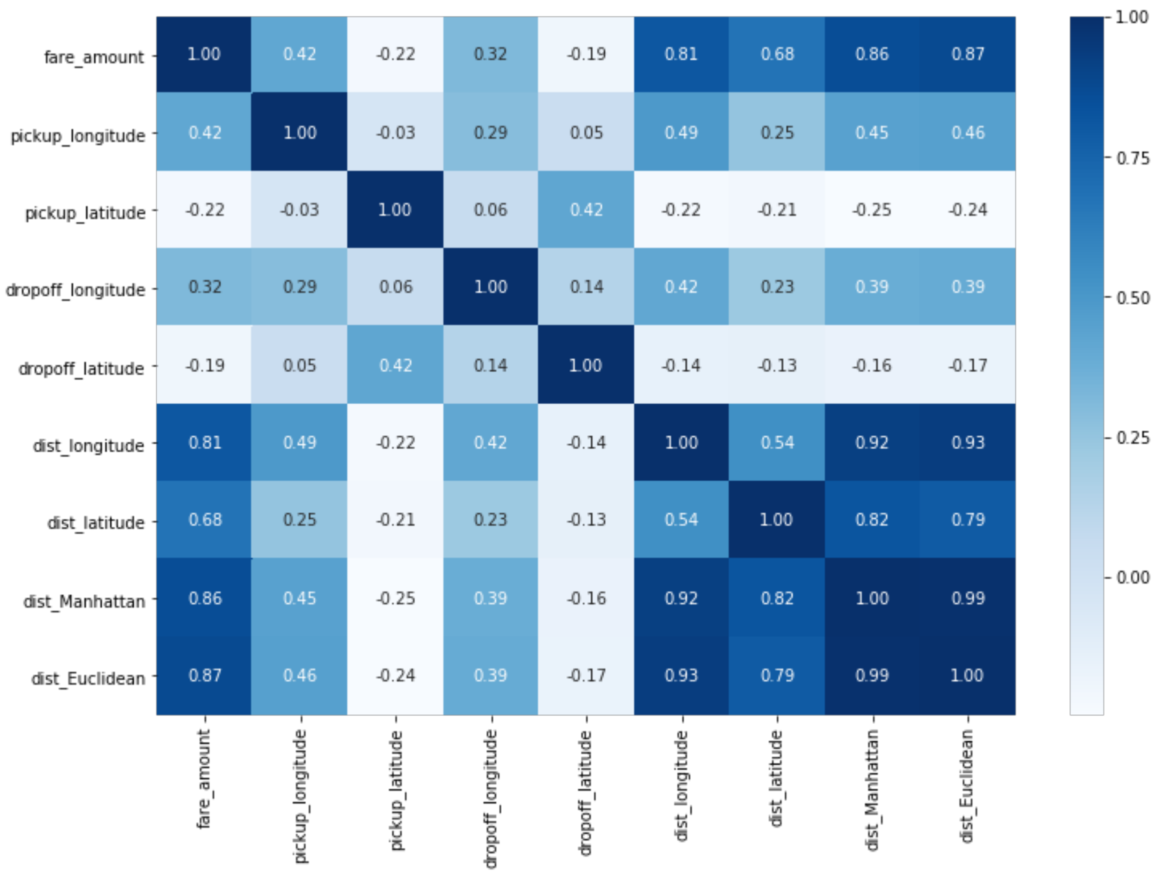
It can be clearly found that 'fare_amount' has high correlation with 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', and 'dropoff_latitdue' based on correlation coefficient (>0.15). Third, I select the above four features, target fare_amount and remove other insignificant features to have pair scatter plot :



I found there're no obvious outliers and their distribution corresponds to their correlation coefficient, which means all the measures works in my preprocess pipeline.

Intuitively, taxi fare should have some relationships with travel distance between pickup location (longitude/latitude) and dropoff location (longitude/latitude). To further extract features, I add absolute distance between pickup and dropoff location for both longitude and latitude value, and also add two typical distance between pickup location and dropoff location : Manhattan distance and Euclidean distance.

After adding the distance features, we have four very good new features that have high correlation coefficient (>0.6) with our target, 'fare_amount' :



	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	dist_longitude	dist_latitude	dist_Manhattan	dist_Euclidean
fare_amount	1.00	0.42	-0.22	0.32	-0.19	0.81	0.68	0.86	0.87
pickup_longitude	0.42	1.00	-0.03	0.29	0.05	0.49	0.25	0.45	0.46
pickup_latitude	-0.22	-0.03	1.00	0.06	0.42	-0.22	-0.21	-0.25	-0.24
dropoff_longitude	0.32	0.29	0.06	1.00	0.14	0.42	0.23	0.39	0.39
dropoff_latitude	-0.19	0.05	0.42	0.14	1.00	-0.14	-0.13	-0.16	-0.17
dist_longitude	0.81	0.49	-0.22	0.42	-0.14	1.00	0.54	0.92	0.93
dist_latitude	0.68	0.25	-0.21	0.23	-0.13	0.54	1.00	0.82	0.79
dist_Manhattan	0.86	0.45	-0.25	0.39	-0.16	0.92	0.82	1.00	0.99
dist_Euclidean	0.87	0.46	-0.24	0.39	-0.17	0.93	0.79	0.99	1.00

Implementation

Part A. Explore Data Analysis

- Import necessary modules : import all necessary modules.
- Load data :
  - load 200K sample data from 'train.csv'
  - have a statistics of loaded data to identify abnormal data
  - show pair scatter plot of features to justify the abnormities
- Remove N.A. and outliers
  - Implement check_na and remove_na function to check and remove N.A. data samples. If N.A. data are found, use remove_na function to remove them
  - Implement remove_GPS_outliers function to remove GPS location out of New York city. Show GPS location using show_GPS function to justify all GPS outliers are removed.
  - Implement remove_fare_outliers function to remove unreasonable fare (It should be at least $2.5 initial charge according to New York's taxi information for yellow cab . On the other hand, I limit the maximum value of fare_amount to 300 by observing the scatterplot)
- Transform data
  - Implement transform_pickup_datetime function to transform 'pickup_datetime' and add four features : 'pickup_year', 'pickup_month', 'pickup_weeekday', and 'pickup_hour'.

- Explore Data and Feature Selection
    - ■ Remove ‘key’ and ‘pickup_datetime’ and show pair feature correlation. Based on the correlation plot, select four significant features (absolute correlation value >0.15) : ‘pickup_longitude’, ‘pickup_latitude’, ‘dropoff_longitude’, and ‘dropoff_latitude’
- Feature extraction
    - ■ Implement add_GPS_dist function to extract four features ‘dist_longitude’, ‘dist_latitude’, ‘dist_Manhattan’, and ‘dist_Euclidean’ from ‘dropoff_longitude’, ‘pickup_longitude’, ‘dropoff_latitude’, and ‘pickup_latitdue’.
- Data Standardization
    - ■ Implement data_standardize function to standardize numerical features except for the target, ‘fare_amount’.

Part B. Data Preprocess

- Import necessary modules
- Load Data : Instead of using whole dataset, I only use 50K samples (sample row from 0 to 50K) in "train.csv" file for training and validation (90% data for training and 10% data for validation).
- Preprocess and Split Data : Implement preprocess function to wrap data imputing (i.e. remove N.A. value), features selection, extraction, outlier-removal and standardization function pipeline for convenience. After preprocessed the 50K samples in “train.csv”, they are split. 90% data for training and 10% data for validation.

Part C. Model Training

- Define evaluation metric : Test r2_score function in sklearn library to evaluate R^2
- Shuffle and Split Data : Among training and validation data, 80% data are used for training and 20% data are used for validation.
- Model Training and Selection:
    - ■ Implement Linear_reg function as multiple linear regression model (benchmark model)
    - ■ Implement Poly_reg function as polynomial regression model. Optimize model by changing polynomial degree from 2, 3, to 4.
    - ■ Implement RF_reg function as random forest regression model. Optimize model by changing number of estimators from 50, 100, 150 to 200.
    - ■ Implement MLP_reg function as multiple layer perceptron regression model. Optimize model by changing number of nodes in each layer from 10, 20, 30, to 40. For simplicity, the default number of layer used is 3.
    - ■ Select model with highest R^2 value in both training and validation data as best model.
- Sensitivity Analysis : recall the trained model estimator of best model and benchmark model and perform sensitivity analysis for another three batches of data.

Part D. Model Evaluation

- Load test data : load 10K test samples from 'test.csv' file as test data
- Preprocess : preprocess the test data using preprocess function implemented before.
- Model evaluation : evaluate R^2 using benchmark model and best model based on test data. Compare the best model predicted fare_amount with groundtruth fare_amount.

Refinement

The hyperparameters to optimize the three candidate models are :

- Multiple Linear Regression model (benchmark) : no hyperparameter
- Polynomial Regression model : Change polynomial degree from 2, 3, to 4.
- Random Forest Regression model : Change number of estimators from 50, 100, 150 to 200.
- Multiple Layer Perceptron Regression model. Change number of nodes in each layer from 10, 20, 30, to 40. For simplicity, the default number of layer used is 3.

IV. Results

Model Evaluation and Validation

Here's the training log and estimated R^2 for training and validation data for different hyper-parameters :

- **Multiple Linear Regression model (benchmark) :**
  R^2 (train) : 0.7729741316254147
  R^2 (valid) : 0.809754753525481

- **Polynomial Regression model :**
  Polynomial degree 1
  R^2 (train) : 0.7729843036205323
  R^2 (valid) : 0.8098699723098052
  Polynomial degree 2
  R^2 (train) : 0.7922608093755379
  R^2 (valid) : 0.8272034943361776
  Polynomial degree 3
  R^2 (train) : 0.8021432880961903
  R^2 (valid) : 0.8338251696982749
  Polynomial degree 4
  R^2 (train) : 0.8112781920025682

R^2 (valid) :   0.8025435308497839

We can found polynomial degree = 3 is the best model among polynomial regression models because it has high training R^2 value and validation R^2 is also the peak value before reverting.

- **Random Forest Regression model :**

  Random Forest number of estimators :   50
  R^2 (train) :   0.9733243999496467
  R^2 (valid) :   0.8533891018408051
  Random Forest number of estimators :   100
  R^2 (train) :   0.9744579157443554
  R^2 (valid) :   0.8541811948795832
  Random Forest number of estimators :   150
  R^2 (train) :   0.9747864419162937
  R^2 (valid) :   0.8541748899779061
  Random Forest number of estimators :   200
  R^2 (train) :   0.9750437379662068
  R^2 (valid) :   0.8540403418177958

  We choose number of estimators = 100 as the best model among random forest regression models because it has high training R^2 value and validation R^2 saturates.

- **Multiple Layer Perceptron Regression model.**

  MLP hidden layer =3, number of hidden nodes =   10
  R^2 (train) :   0.8158648517376604
  R^2 (valid) :   0.8479220681658175
  MLP hidden layer =3, number of hidden nodes =   20
  R^2 (train) :   0.8246558228373615
  R^2 (valid) :   0.850849950748826
  MLP hidden layer =3, number of hidden nodes =   30
  R^2 (train) :   0.8257800335298277
  R^2 (valid) :   0.8500624174084247
  MLP hidden layer =3, number of hidden nodes =   40
  R^2 (train) :   0.8237470179256542
  R^2 (valid) :   0.8448035054943113

We can found number of hidden nodes = 20 is the best model among multiple layer perceptron regression models because it has high training R^2 value and validation R^2 is also the peak value before reverting.

Among all models, random forest regression model with number of estimators = 100 has highest training R^2 = 0.974 and validation R^2 = 0.854. Thus, it seems to be our best model candidate. On the other hand, the benchmark Multiple Linear Regression model has R^2(train) = 0.773 and R^(valid) = 0.810.

To justify the sensitivity of best model and benchmark model, I used sample rows range from 100K to 130K in 'train.csv' file to test model sensitivity. Three sensitivity batch data with 10K samples are used. Here're R^2 and average R^2 for the three sensitivity batch data :

Sensitivity of benchmark model (multiple linear regression model) :
R^2 : -3.082568191050259e+20
R^2 : -1.2677135170730803e+20
R^2 : -5.474876256878982e+20
Average R^2 : -3.2750526550007736e+20

Sensitivity of best model (random forest regression model) :
R^2 : 0.83434451537281
R^2 : 0.7276965321528759
R^2 : 0.8062664829977666
Average R^2 : 0.7894358435078175

We can find the benchmark model (multiple linear regression model) is very sensitive to data, compared with R^2 for valid dataset 0.810, the average R^2 for sensitivity batches is -3.2750526550007736e+20. It has very large variation. On the other hand, our best model (random forest regression model) is not that sensitive to data, the R^2 for valid data is 0.854, and average R^2 for sensitivity batches is 0.789. It's comparable and has similar performance among different dataset.

Justification

To evaluate our final model performance, I loaded 10K data samples (sample row from 50K to 60K) as test data in 'test.csv' file, and used benchmark model (multiple linear regression model) and best model (random forest regression model) to predict the test data and then calculate R^2.

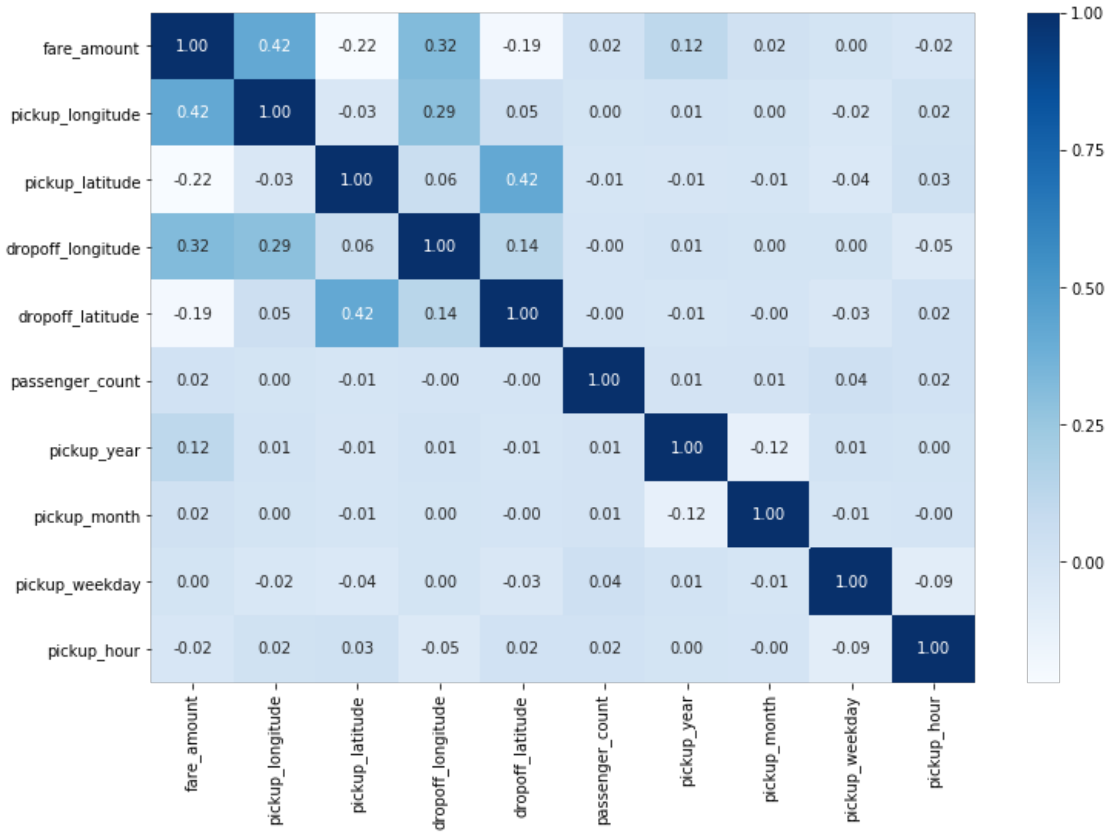
I got R^2 (test) of best model = 0.8448035054943113, which is comparable to R^2 in train and valid data. It performs pretty good as expected, and it's not sensitive to data as discussion in sensitivity analysis section. Therefore, our best model can serve as a good solution to this problem.

On the other hand, the benchmark model (linear model) is terrible because R^2 = -3.5642511557435007e+19 < 0. Predicting average value of taxi fare would even have better R^2 than the benchmark model. Thus, this model fails to solve the problem.

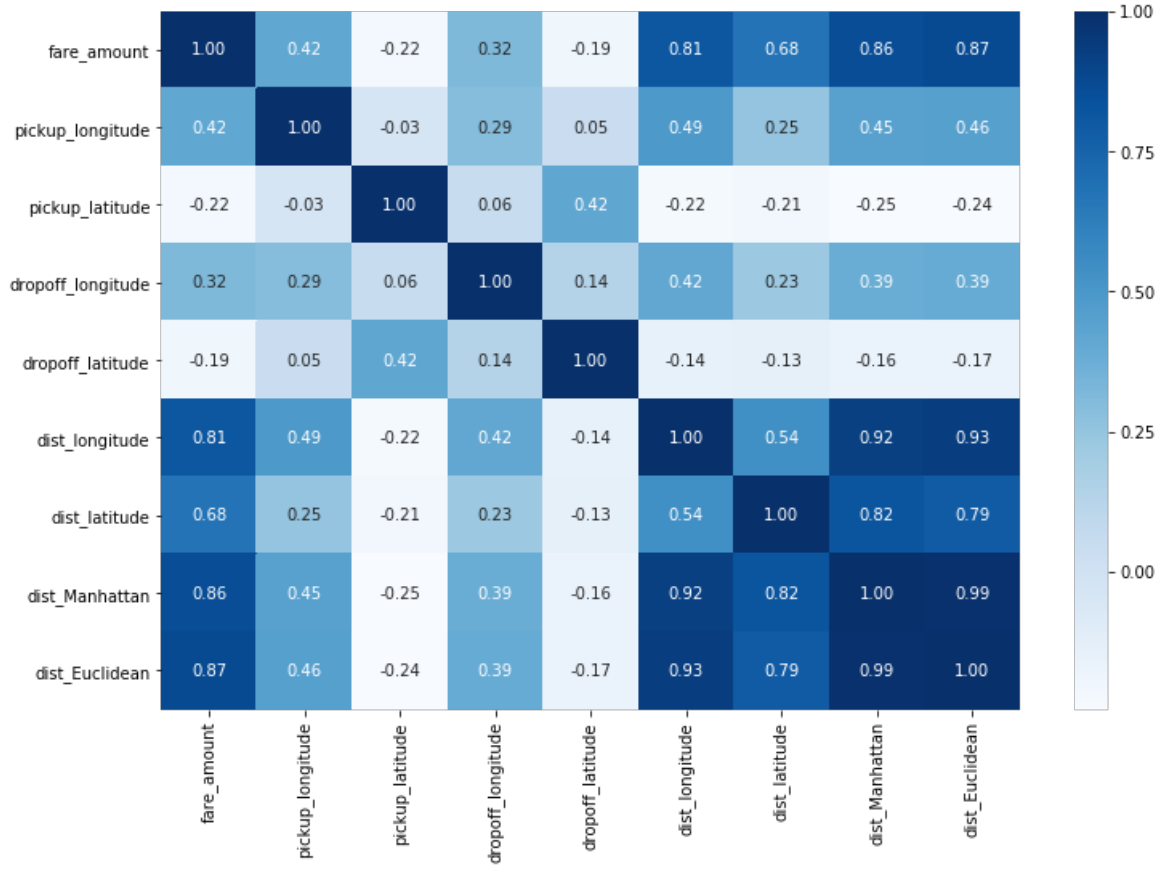
V. Conclusion

Free-Form Visualization

In PART A. Explore Data Analysis (EDA), I explored the data, removed n.a. values and outliers, transformed all data to numerical form and got first correlation plot as below. It's noted that there're only four features can be viewed as significant (absolute value > 0.15) and the most important feature is "pickup_longitude".



However, after extracting features using the four features, I got new features : "dist_longitude", "dist_latitude", "dist_Manhattan", and "dist_Euclidean". By removing insignificant features and adding the four new features, I got second correlation plot below. It's quite impressive that new features have much higher correlation with "fare_amount" than previous four features. Until that time, I got a set of prominent features that can be used for training a better model than before.



	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	dist_longitude	dist_latitude	dist_Manhattan	dist_Euclidean
fare_amount	1.00	0.42	-0.22	0.32	-0.19	0.81	0.68	0.86	0.87
pickup_longitude	0.42	1.00	-0.03	0.29	0.05	0.49	0.25	0.45	0.46
pickup_latitude	-0.22	-0.03	1.00	0.06	0.42	-0.22	-0.21	-0.25	-0.24
dropoff_longitude	0.32	0.29	0.06	1.00	0.14	0.42	0.23	0.39	0.39
dropoff_latitude	-0.19	0.05	0.42	0.14	1.00	-0.14	-0.13	-0.16	-0.17
dist_longitude	0.81	0.49	-0.22	0.42	-0.14	1.00	0.54	0.92	0.93
dist_latitude	0.68	0.25	-0.21	0.23	-0.13	0.54	1.00	0.82	0.79
dist_Manhattan	0.86	0.45	-0.25	0.39	-0.16	0.92	0.82	1.00	0.99
dist_Euclidean	0.87	0.46	-0.24	0.39	-0.17	0.93	0.79	0.99	1.00

Reflection

As my first machine learning project using Kaggle's dataset, I found it's quite challenging to propose and implement a full machine learning solution.

[Part A. Explore Data Analysis] I did a brief statistics about the data to get a overview understanding and figured out some data abnormality including N.A. data, GPS outliers and fare outliers. Then I transformed 'pickup_datetime' field to four numerical fields : 'pickup_year, 'pickup_month', 'pickup_weekday', and 'pickup_hour'. Then, I compared correlation coefficient of feature pairs and selected four significant features : 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', and "dropoff_latitude". Moreover, I extract four new features : 'dist_longitude', 'dist_latitude', 'dist_Manhattan', and 'dist_Euclidean' based on previous selected four features. In this stage, I'm impressed by high correlation these new features with 'fare_amount'. Afterwards, I built a function to standardize the eight features because most machine learning algorithms performed better when data standardized.

[Part B. Data Preprocess] I loaded 50K samples in “train.csv” file, split them as 90% training data and 10% validation data, preprocessed the data using the pipeline discussed in Part A, and examined the correlation plot for pair features.
[Part C. Model Training and Selection] I chose four models to train. First model is my benchmark model : multiple linear regression model. The other three models are my experiment models : polynomial regression, random forest regression and multiple layer perceptron regression. Hyper-parameters are experimented while training and validating. Based R^2 value for train data and validation data, I picked forest regression model with number of estimators =30 as my best model candidate. To analyze data sensitivity of my best model, I used data of sample rows range from 100K to 130K in 'train.csv' file to test model sensitivity by examining average R^2. The average R^2 is comparable to R^2 in validation, which means it’s not too sensitive for different data and the model is robust.

[Part D. Model Evaluation] Finally, I loaded 10K samples (sample row from 50K to 60K) in "train.csv" for testing. R^2 values for both benchmark model and best model are evaluated using test data. The best model performed well as expected. The R^2 value in testing is comparable to R^2 in validation, which justified my best model can be served as a good solution to this project. On the other hand, benchmark model had R^2 < 0. Even always predicting average value of “fare_amount” can be better than it.

Throughout this project, I found it’s not easy to build a complete preprocess pipeline such as data imputation, data explore, data transform, feature extraction, and feature selection. Also, carefully cleaning noisy data and extracting new features are challenging and they significantly affect the final performances. However, I’m glad that I can build it from scratch and made it work.

Improvement

There’re many improvement can be tried. First, I can use more data for training or try other models. Second, the hyper-parameters are not searched thoroughly. I can test more hyper-parameter settings. Third, the outlier removal can be more precise based on mean and variance value. Fourth, clusters of samples can be experimented. It’s possible to find data cluster related to city zone, and it provides a new direction to improve.

Reference

[1] https://www.kaggle.com/c/new-york-city-taxi-fare-prediction#description
[2] Python Machine Learning 2nd edition, by Sebastian Raschka and Vahid Mirjalili
[3] Hands-On Machine Learning with Scikit-Learn & TensorFlow, by Aurelien Geron

[4] https://www.kaggle.com/dster/nyc-taxi-fare-starter-kernel-simple-linear-model

[5] https://www.kaggle.com/aiswaryaramachandran/eda-and-feature-engineering

[6] https://www.quora.com/How-does-random-forest-work-for-regression-1