Machine learning Project Report

Group member：

Haoun Guo	hg1483
Da Cai	dc4069
Naisheng Zhang	nz862

Project name：New York City Taxi Fare Prediction

Description

According to the context, we know that the fare amount prediction only based on the distance between the two points will result in an RMSE of $5-$8. Our goal is to find new features from raw features and select the suitable model to make prediction more accuracy.

Hypothetical solution

First, we hope to combine the knowledge we have learned in the class with the subject. After an in-depth analysis of the problem, we hope to use a DNN model. First, we need to process the data and create several kinds of new features like time, distance, a transportation hub.

Second, we use DNN models to take advantage of the features we create and prediction. Finally, we evaluate our prediction model and compare with the previous RMSE.

Data and Method

The data is pretty large and considered we want to use DNN to analysis this question, we use google colab. Colab is convenient to connect google drive(Our train and test data) and we can choose GPU to make it faster.

Data file

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). We don’t know the real fare amount of test, so we split train data into new train data and new test data, and we construct the model by new train data and make a prediction by new test data.

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.

Target

fare_amount - float dollar amount of the cost of the taxi ride.

Reference

1: https://www.kaggle.com/breemen/nyc-taxi-fare-data-exploration

2: https://www.kaggle.com/dimitreoliveira/taxi-fare-prediction-with-keras-deep-learning

From the first example, we know the raw data locations in a map and raw features distributions in charts.

From the second one, we know how to create new features and we bring the conception transportation hub to our lab.

1 Formulation

After compare data with the question, we found raw data features can't solve question well.

Fare amount can be influenced by many factors, like distance, unit price, starting price, weather, peak period, road congestion, etc. From raw features, we can only get location, time, passenger count. These features are not enough for our question, so we need to create new suitable features from raw features in our analysis. From new closely related features, we may predict fare amount well in mathematically.

2 Approach

As we mentioned above, fare amount is related to many features, and the relation is not linear, the relationship may be complex, so we want to use DNN to solve this problem, DNN can fit complex features and target more accuracy.

From the reference and description in our code, the quality of raw data has errors, like pick up location and drop off location is the same, so we need to filter some wrong data to avoid bad influence in model parameters. We filter data in several aspects: location, distance, impossible data(value less than 0)and price.

As for parameters, the DNN model has many parameters, we can select a batch size, epochs, learning rate and add layers.

3 Analysis(main content)

Valid data filter:

First, we remove fare<=0 case, and from dataframe.describe, we know 99.9% fare is less than $80, the max fare is 500, so we select date under $80.

From reference, we know some data is out of NYC range, so we need to filter it out. We select data in the area(N 40.5°-N 41.8°, W -72.8°- W -74.5°).

After filter, data size changes from 1000000 to 967328.

3.1 Add new features based on time

First, we need to extract new features(Y/M/D/H) from feature pick up data. After we extract new Y/M/D/H features, these features are not directly related to the fare amount, so we need to transform these features to new features again.

For the year feature, it may influence the unit price or starting price, so we keep the year feature.

For month feature, it has no direct influence on the fare, but season weather is related to the fare amount, like summer has more rain and winter has more snow. We create spring to winter four features based on month feature. For example, if a sample is in October, then feature fall is set to 1, the other three season features are set to 0.

For day feature, we don't know if 1st and 31st have a different influence, so we try to reclassify it to Monday to Sunday. After classification, we add new features workday and weekends because road condition, the number of taxis and other features may be different. If a sample is in Sunday, the weekend feature is set to 1, and workday is set to 0.

For hour feature, we try to reclassify hour to three period features: day time(7am-16pm),night(16pm-21pm),late night(21pm-7am). In the day time, traffic conditions may be better. In the night, people go home from work, traffic condition may be worse. In the late night, unit price may be high.

3.2 Add new features based on distance

The raw data only has the latitude and longitude corresponding to the pickup and dropoff. This is definitely not the direct features for measuring the fare. In our visual impression, the fare is related to the distance, so the old latitude and longitude feature need to be converted to the distance feature: Geographical distance (unit km).

In real condition, the distance is based on the route of google map. However, our data does not provide distance based on the real route, so the Geographical distance is set to a two-point straight distance. The distance formula based on latitude and longitude is as follows.

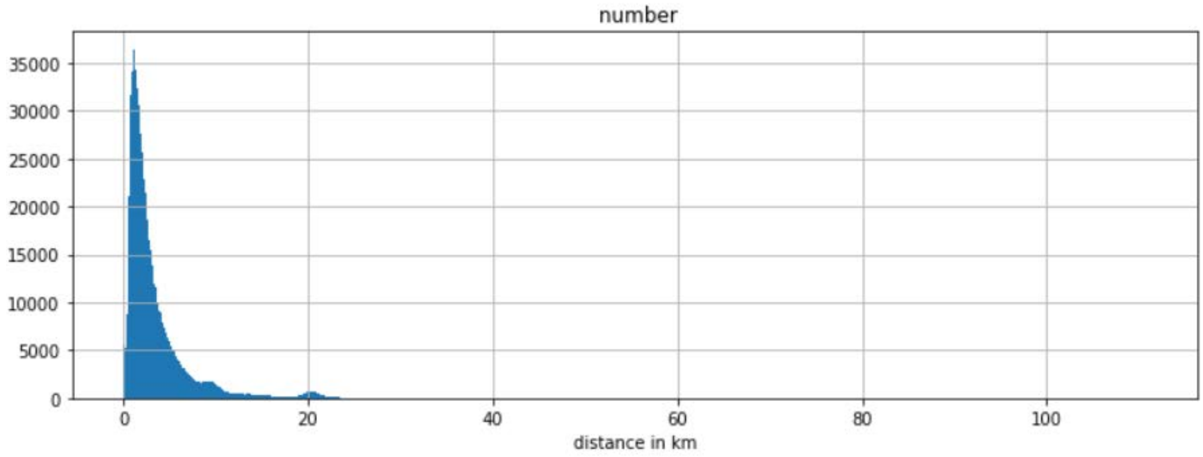
$$\theta_1 = {(longitude_{pickup} - longitude_{dropoff}) * \pi}/{2}$$

$$\theta_2 = {(latitude_{pickup} - latitude_{dropoff}) * \pi}/{2}$$

$$\theta_3 = \sin\left({\theta_1}/{2}\right) * \sin\left({\theta_1}/{2}\right) + \sin\left({\theta_2}/{2}\right) * \sin\left({\theta_2}/{2}\right) + \cos\left({\theta_1}/{2}\right) * \cos\left({\theta_2}/{2}\right)$$

$$\text{distance} = 2 * \sin^{-1}\sqrt{\theta_3} * 6371$$

From the followed diagram, most samples are under 20km.



From our code, we found that the minimum distance of train and test is less than 10m, which is obviously inconsistent with common sense. We drop these distance<10m samples.

View the distribution of the distance by using the describe function. 25% sample distance<1.3km, 75% sample distance<4km, 99%sample distance<20km. According to the New York City taxi pricing method and common sense, the distance is inversely proportional to the fare amount/distance. We need to add new features according to the distance to reflect the difference between the unit price and the distance.

Add new features according to distance: 0-1km, 1km-4km, 4km-15km, 15km~

For example: distance=13km Feature: distance_1km=1, distance_4km=3, distance_15km=9

3.3 Add new features based on location

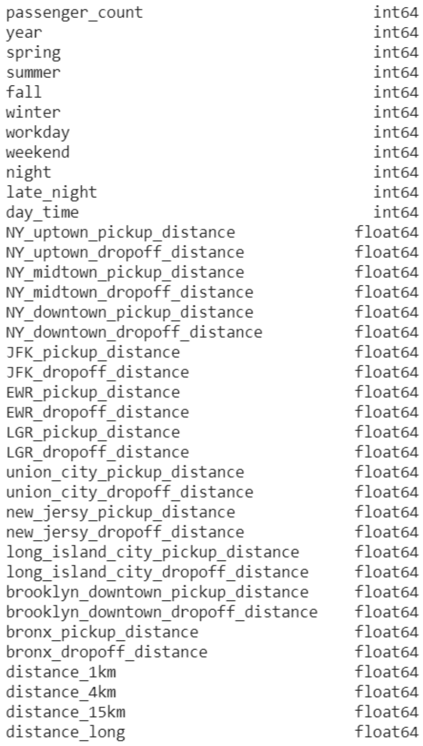
Based on our reference, we found that some areas belong to the transportation hub of New York, such as:

Manhattan downtown, midtown and uptown, JFK, EWR and LGR Airport, Union City, New Jersy, LIC, Brooklyn downtown.

These transportation hubs have an influence on fare amount. To evaluate these cases, we add features: the distance between these hubs and each sample

After we create the function of new features, we need to add new features into train and test data and delete old features.

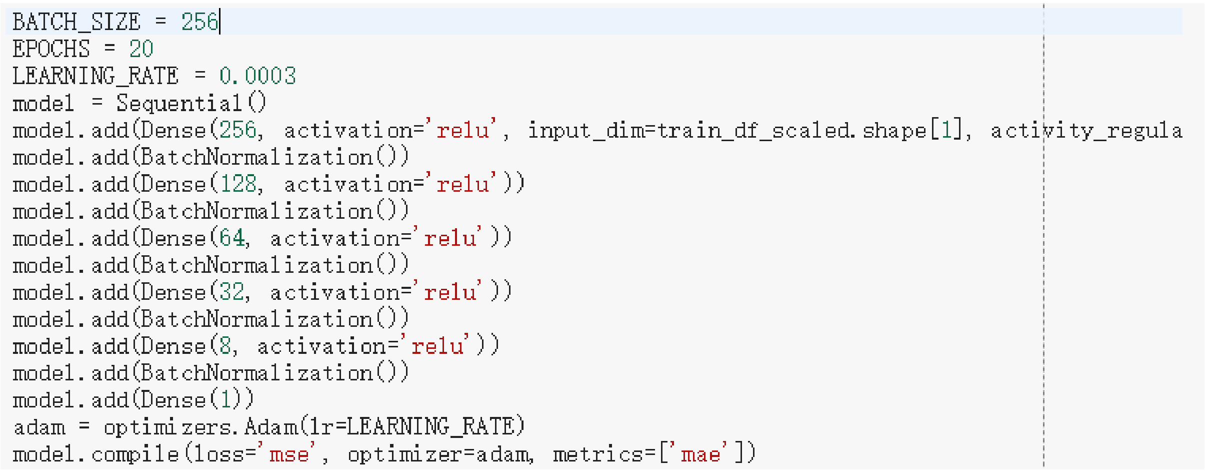
3.4 New features:



```
passenger_count                     int64
year                                int64
spring                              int64
summer                              int64
fall                                int64
winter                              int64
workday                             int64
weekend                             int64
night                               int64
late_night                          int64
day_time                            int64
NY_uptown_pickup_distance         float64
NY_uptown_dropoff_distance        float64
NY_midtown_pickup_distance        float64
NY_midtown_dropoff_distance       float64
NY_downtown_pickup_distance       float64
NY_downtown_dropoff_distance      float64
JFK_pickup_distance               float64
JFK_dropoff_distance              float64
EWR_pickup_distance               float64
EWR_dropoff_distance              float64
LGR_pickup_distance               float64
LGR_dropoff_distance              float64
union_city_pickup_distance        float64
union_city_dropoff_distance       float64
new_jersy_pickup_distance         float64
new_jersy_dropoff_distance        float64
long_island_city_pickup_distance  float64
long_island_city_dropoff_distance float64
brooklyn_downtown_pickup_distance float64
brooklyn_downtown_dropoff_distance float64
bronx_pickup_distance             float64
bronx_dropoff_distance            float64
distance_1km                      float64
distance_4km                      float64
distance_15km                     float64
distance_long                     float64
```

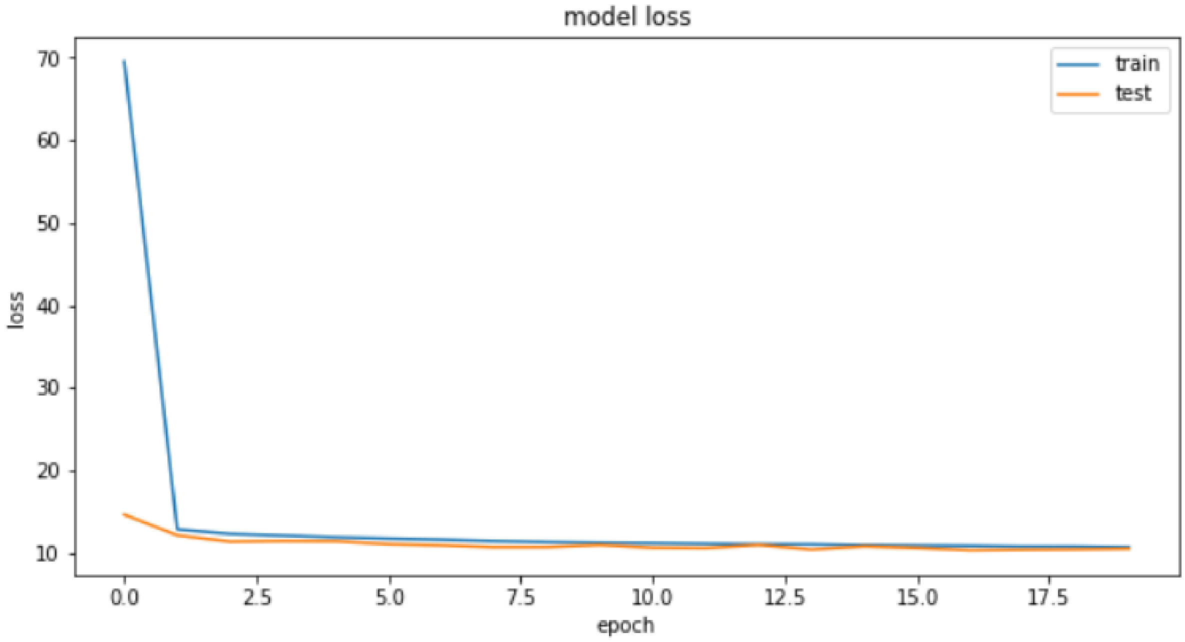
4 Model construction

For our DNN, we select batch size=256, epochs=20, learning rate=0.0003 and add layers.



```
BATCH_SIZE = 256
EPOCHS = 20
LEARNING_RATE = 0.0003
model = Sequential()
model.add(Dense(256, activation='relu', input_dim=train_df_scaled.shape[1], activity_regula
model.add(BatchNormalization())
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(8, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1))
adam = optimizers.Adam(lr=LEARNING_RATE)
model.compile(loss='mse', optimizer=adam, metrics=['mae'])
```

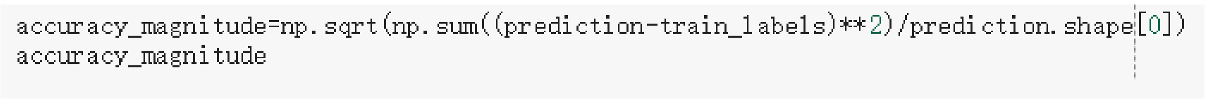
5 Evaluation



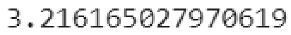
This diagram is about loss and epoch relation, we can get loss is stable when epoch after 5.

5.1 RMSE evaluation

For fare amount prediction, we want to make error less, and we evaluate it by the root mean square error(RMSE).For RMSE evaluation method, it’s 3.2, far less than original error$5-$8



```
accuracy_magnitude=np.sqrt(np.sum((prediction-train_labels)**2)/prediction.shape[0])
accuracy_magnitude
```



```
3.216165027970619
```

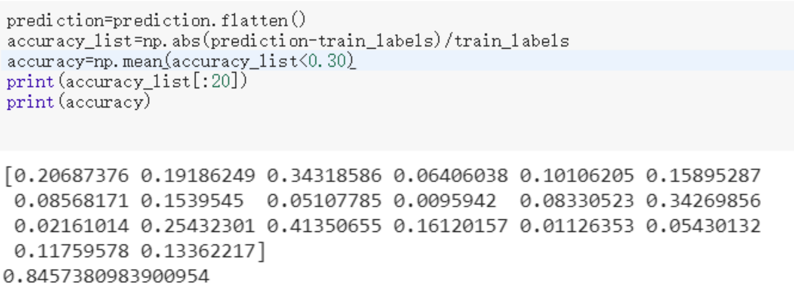
From our code, we get fare amount mean is about $11.4 if fare amounts are in a normal distribution, the probability price error<=3.2 is about 95.44%.

5.2 accuracy evaluation

By the way, we think RMSE is not enough. If the real fare amount is $5, and RMSE is $3, this is not good, mean/standard deviation=60%, this is not what we want. To make evaluation more accuracy, we define a condition:

$$\frac{\left|fare_{amount_{predict}}-fare_{amount_{real}}\right|}{fare_{amount_{real}}}\text{<=30\%}$$

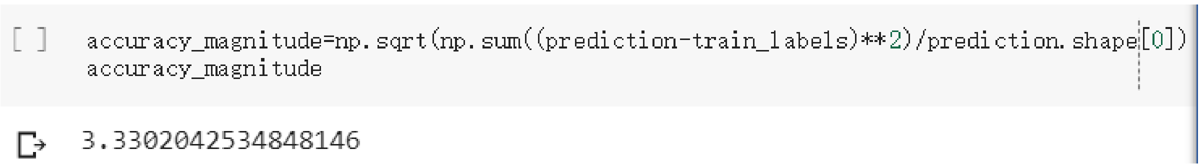
If the prediction of fare amount satisfies above inequality, we consider this prediction is accurate.



```
prediction=prediction.flatten()
accuracy_list=np.abs(prediction-train_labels)/train_labels
accuracy=np.mean(accuracy_list<0.30)
print(accuracy_list[:20])
print(accuracy)
```

```
[0.20687376 0.19186249 0.34318586 0.06406038 0.10106205 0.15895287
 0.08568171 0.1539545  0.05107785 0.0095942  0.08330523 0.34269856
 0.02161014 0.25432301 0.41350655 0.16120157 0.01126353 0.05430132
 0.11759578 0.13362217]
0.8457380983900954
```

From our model, the prediction rate is about 85%. For example, if passenger’s predict cost is 10$, the probability of real cost in $7-$13 is about 0.85



```
accuracy_magnitude=np.sqrt(np.sum((prediction-train_labels)**2)/prediction.shape[0])
accuracy_magnitude
```

```
3.3302042534848146
```