Prediction on New York City Taxi Trip Duration and Neighborhood Division

Wenjie Gu, Qianying Huang, Rong Huang, Bo Jiang

I. Introduction

We want to predict the duration of taxi rides in New York City based on features such as pick-up time and location, drop-off time and location, and number of passengers. We obtained the dataset from Kaggle. The training data contains 1.5 million entries and the testing data contains about 600 thousand entries. We will build our prediction model based on the training data and test its performance by applying the model to the testing data. We first excluded potential outliers that went beyond the scope of New York City. Next, we proceeded to feature engineering by labeling the neighborhood by K-means clustering. Finally, we built a neural net to predict taxi ride duration.

II. Assumptions and Analysis

The training observations and the testing observations should cover the same time period and geographical area. First, we excluded the trip that goes We check this assumption by visualizing the time-date and long-lat features of train and test dataset. See Fig.1 and Fig.2. Hence, the assumptions are satisfied.

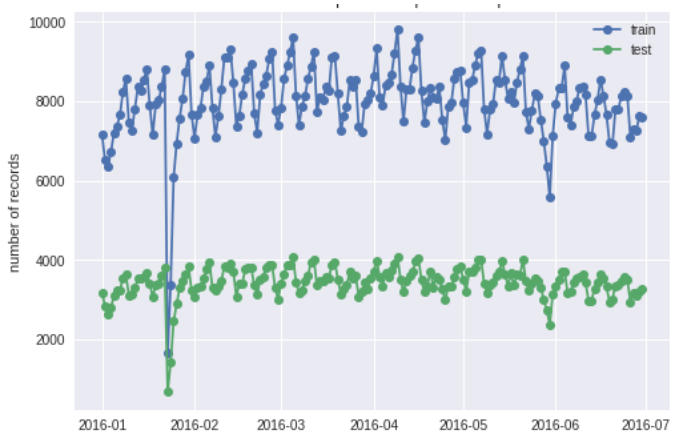

Figure 1. The time periods for train and test dataset overlap.

III. Tag the neighborhood by K-means clustering

A. Overview

Although we are given the longitude and latitude of pick-up and drop-off location, it is more useful if we can label the location information with major neighborhoods in the New York City. Despite the fact that Google Map API can reversely

Advisor: Professor Matthew Jacobs, Math 156 Project

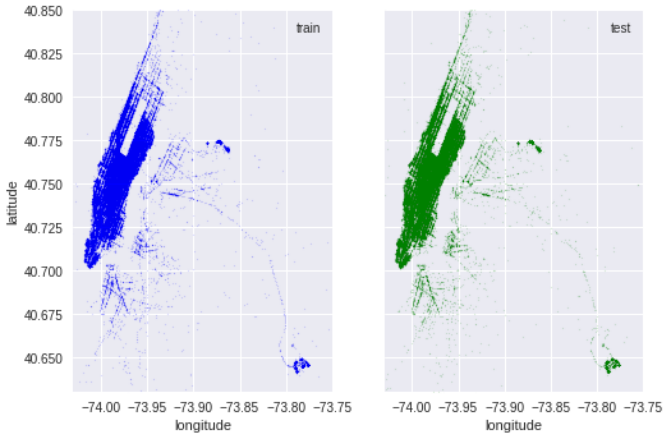

Figure 2. The geographical area for train and test dataset overlap.

geocode the longitude and latitude, it is not feasible to utilize Google API to label more than 2 millions entries of data, due to the daily threshold of allowed interaction with the API and computational speed. Therefore, we decided to make use of k-means clustering to label the pick-up and drop-off location information. We aggregated the pick-up location and drop-off location and chose to divide into 16 neighborhoods based on the official information released by the Census Bureau: Midtown-East, East Harlem, JFK Airport, LaGuardia Airport, Upper West Side, Brooklyn, and Financial District, etc. To verify the accuracy, we randomly selected 10,000 samples, and compared the labels generated by k-means algorithm and the true neighborhood given by Google API.

B. Pseudocode of k-means clustering

The following Pseudocode is intended for pythonic style implemention, since the coding part is all completed through Python.

- **First initialize the centroid for each neighborhood.**

```
pickupLocation = pickupGetloc()
dropoffLocation = dropoffGetloc()
pickupCentroids = getCentroids(pickupLocation, 16)
dropoffCentroids = getCentroids(dropoffLocation, 16)
iterations = 0
oldCentroids = []
maxIteration = N
```

- **Implement k-means, stop when reaching max iteration or satisfy convergence property.**

while iterations is smaller than maxIteration or !checkConvergence():

oldCentroids = currentCentroids
iterations += 1
pickupLabels = getLabels(pickupLocation, currentCentroids)
dropoffLabels = getLabels(dropoffLocation, currentCentroids)
pickupCentroids = getCentroids(pickupLocation, pickupLabels, 16)
dropoffCentroids = getCentroids(dropoffLocation, dropoffLabels, 16)
return currentCentroids

def checkConvergence():
If the norm $<$ certain threshold, then return true. Otherwise, return false.

def getLabels:
For each point, calculate the distance to the current centroid and assign the label to the closest centroid.

def getCentroids:
Calculate the mean of that class

C. Clustering Result

The above graphs display the result of neighborhood labeling through k-means algorithm. It is interesting that both JFK Airport and LaGuardia Airport have been identified as an individual cluster, probably because the total number of riders are significantly large enough to form a cluster.

Compared to the result given by Google API, the classification accuracy is 82.45%. Also, after we obtained the neighborhood labels, we were able to observe the most frequent inter-neighborhood travel paths (figure 6). Most inter-neighborhood trips occurred in Manhattan area. Neighborhoods that had frequent interactions were: Midtown-East and Williamsburgt, Midtown-East and Upper East Side, East Harlem and Upper East Side, and Soho and Chelsea. Therefore, a majority of trip in New York City belonged to short trips.

It

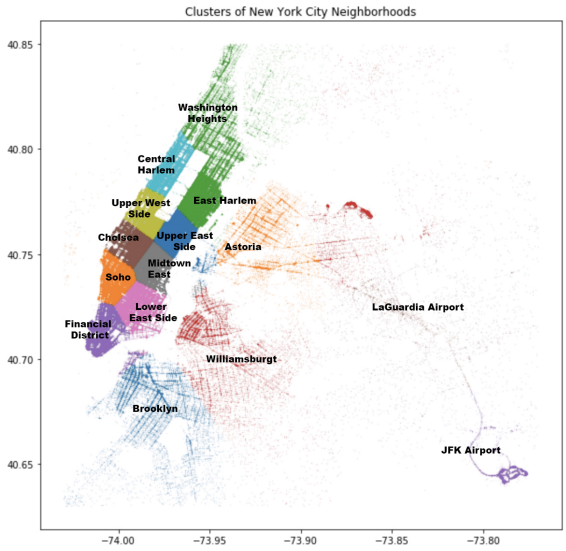

Figure 3. Neighborhood of NYC generated by k-means

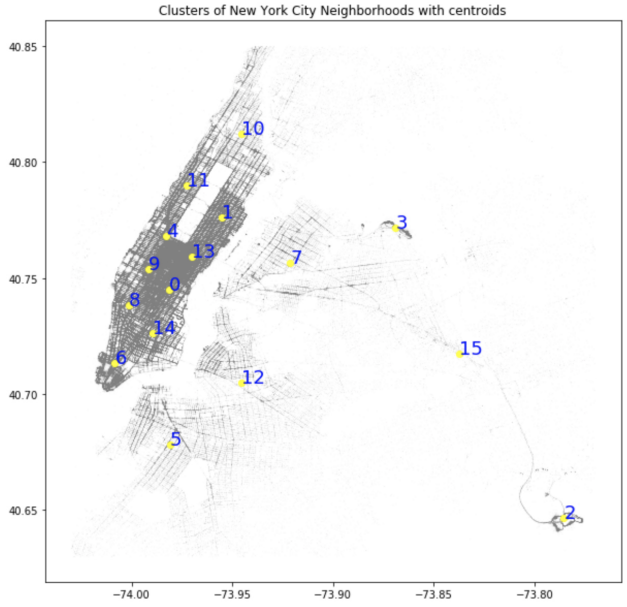

Figure 4. K-means centroids

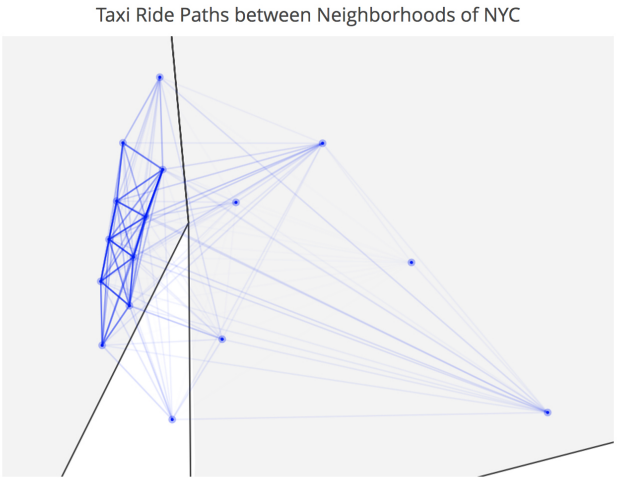

Figure 5. Inter-neighborhood taxi ride paths

	Cluster_ID	Latitude	Longitude	Neighborhood
0	0	40.744760	-73.981282	Midtown-East
1	1	40.775994	-73.955308	East Harlem
2	2	40.646615	-73.785425	JFK Airport
3	3	40.771528	-73.868996	LaGuardia Airport
4	4	40.767980	-73.983087	Upper West Side
5	5	40.678305	-73.981069	Brooklyn
6	6	40.713440	-74.008350	Financial District
7	7	40.756574	-73.921281	Astoria
8	8	40.738328	-74.001227	Soho
9	9	40.753845	-73.991803	Chelsea
10	10	40.812025	-73.945474	Washington Heights
11	11	40.789963	-73.972717	Central Harlem
12	12	40.704832	-73.945266	Williamsburgt
13	13	40.759267	-73.970153	Upper East Side
14	14	40.726322	-73.989362	Lower East Side
15	15	40.717222	-73.837548	Queens

Figure 6. K-means centroids with longitude, latitude and neighborhoods

IV. Training Neural Net to Predict Trip Duration

A. Data Preprocessing

In addition to clustering the pick-up and drop-off locations by k-means, we applied Principal Component Analysis(PCA) to the geographical coordinates, for feature scaling purpose. After PCA, we see that the new coordinate system is concentrated around the origin without removing important information, which facilitates further neural net training. See Figure 7.

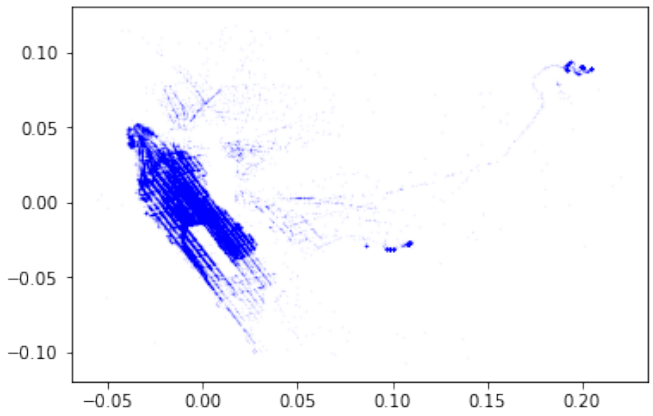

Figure 7. Coordinate after PCA

For each trip, we calculated the haversine distance and Manhattan distance between pick-up location and drop-off location. Based on pick-up datetime, we add new features such as rush hours and holidays, etc.

B. Algorithm

Input: D, all training examples ;
U_I, U_H, U_O, the input, hidden and output units ;
$v_{j,k}$, input value for k outputted by j ;
σ, activation function on each unit ;
η, the learning rate for gradient descent;
Output: w, weights of the units
Initialize random weights to w;
while training loss improvement $>$ delta **do**
  foreach $(X, y) \in D$ **do**
    /* compute output of layer 1 */
    foreach $u \in U_H$ **do**
      $y_u \leftarrow \sigma(\mathbf{w}_u^T x)$
    end
    /* compute output of layer 2 */
    $y_o \leftarrow \sigma(\mathbf{w}_o^T \mathbf{y}_H)$;
    /* back-propagate layer 2 & 1 */
    $\delta_o \leftarrow y_o \cdot (1 - y_o) \cdot (y - y_o)$;
    foreach $u \in U_H$ **do**
      $\delta_u \leftarrow y_u \cdot (1 - y_u) \cdot w_{u,o} \cdot \delta_o$
    end
    /* Update weights */
    foreach $w_{j,k} \in (U_I \times U_H) \cup (U_H \times U_O)$ **do**
      $\delta w_{j,k} \leftarrow \eta \delta_k \cdot v_{j,k}$;
      $w_{j,k} \leftarrow w_{j,k} + \delta w_{j,k}$;
    end
  end
end
return *w*

	pickup_datetime	pickup_latitude	pickup_longitude	dropoff_latitude	dropoff_longitude	dist	manh	weekend	trip_duration	pred
0	2016-03-14 17:24:55	40.767937	-73.982155	40.765602	-73.964630	1.949977	2.020368	0	455	621.643186
1	2016-06-12 00:43:35	40.738564	-73.980415	40.731152	-73.999481	2.132178	2.347206	1	663	619.208530
2	2016-01-19 11:35:24	40.763939	-73.979027	40.710087	-74.005333	3.359039	4.575113	0	2124	1404.521226
3	2016-04-06 19:32:31	40.719971	-74.010040	40.706718	-74.012268	0.475519	0.653590	0	429	357.913917
4	2016-03-26 13:30:55	40.793209	-73.973053	40.782520	-73.972923	0.328461	0.342567	1	435	307.966978
5	2016-01-30 22:01:40	40.742195	-73.982857	40.749184	-73.992081	1.047816	1.239953	1	443	461.983248
6	2016-06-17 22:34:59	40.757839	-73.969017	40.765896	-73.957405	1.314693	1.538761	0	341	387.959614
7	2016-05-21 07:54:58	40.797779	-73.969276	40.760559	-73.922470	5.328986	6.350783	1	1551	1015.633505
8	2016-05-27 23:12:23	40.738400	-73.999481	40.732815	-73.985786	1.532387	1.694105	0	255	537.227992
9	2016-03-10 21:45:01	40.744339	-73.981049	40.789989	-73.973000	1.662582	2.296472	0	1225	1112.298923
10	2016-05-10 22:08:41	40.763840	-73.982651	40.732990	-74.002228	2.373515	3.122257	0	1274	868.025103
11	2016-05-15 11:16:11	40.749439	-73.991531	40.770630	-73.956543	3.944541	4.541733	1	1128	932.828845
12	2016-02-19 09:52:46	40.756680	-73.962982	40.760719	-73.984406	2.385393	2.506101	0	1114	957.312143
13	2016-06-01 20:58:29	40.767941	-73.956306	40.763000	-73.966110	1.100643	1.241851	0	260	378.449795
14	2016-05-27 00:43:36	40.727226	-73.992195	40.783070	-73.974655	2.596011	3.664566	0	1414	1308.907217
15	2016-05-16 15:29:02	40.768593	-73.955513	40.771545	-73.948761	0.756255	0.841567	0	211	262.127277
16	2016-04-11 17:29:50	40.755562	-73.991165	40.725353	-73.999290	1.293849	1.829410	0	2316	852.539227
17	2016-04-14 08:48:26	40.745804	-73.994255	40.723343	-73.999657	0.913702	1.289062	0	731	634.619512
18	2016-06-27 09:55:13	40.713013	-74.003983	40.749924	-73.979195	2.979645	3.889025	0	1317	1172.423688
19	2016-06-05 13:47:23	40.738197	-73.983887	40.727871	-73.991203	0.873049	1.130235	1	251	362.510382
20	2016-02-28 02:23:02	40.742420	-73.980370	40.760635	-73.962852	2.026507	2.507359	1	486	510.004182
21	2016-04-01 12:12:25	40.753361	-73.979538	40.763458	-73.963997	1.755677	2.038250	0	652	781.673539
22	2016-04-09 03:34:27	40.758812	-73.995865	40.740322	-73.993324	0.633381	0.849435	1	423	437.546996
23	2016-06-25 10:36:26	40.747173	-73.993553	40.704384	-74.006142	1.918169	2.710762	1	1163	1014.131858

Figure 8. First 20 lines of the observations combined with predictions. Check last two columns for comparisons.

C. Performance of the Neural Net

The performance of neural network achieves Root Mean Squared Logarithmic Error (RMSLE) 0.46039, which is better than the median score of the Kaggle competition. Therefore, our neural net did perform well. For better visual effects, we randomly select 500 training observations with their corresponding predicted trip durations and plot a graph, see Fig. 9. It should be clear from the graph that the actual duration and prediction usually stay within the same range.

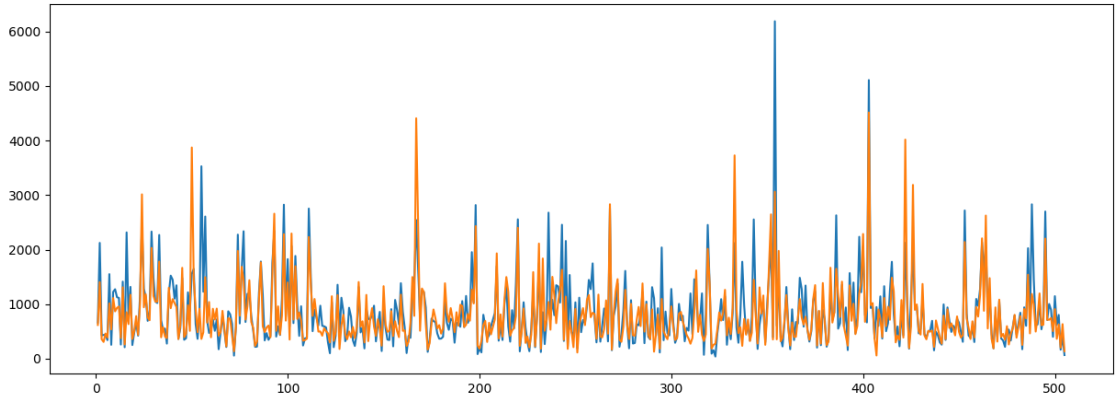

Figure 9. Blue plot for actual trip durations and yellow plot for predictions.