Lyft assignment: estimating travel time

Gaoyang Ye
Jan. 6, 2017

To estimate travel times between two specified locations at a given departure time, I mainly employed two methods: knn estimation (results presented in Part II) and STL analysis (results presented in Part III). The code that generated the results are attached separately. In the last section (Part V), I discussed potential improvements and some ideas for real-time implementation. I'm interested in this assignment, and hopefully, I can have some deeper discussions with Lyft for solving more challenging problems alike.

Part I: Visualization

The first thing I did when I see the task it searched the city in the map and found out that is New York. :P

Figure 1: Map of New York City

Then I plot all the starting points and end points in a graph to get a general feeling of the positions of points (Red points are start points and Blue points are end points).

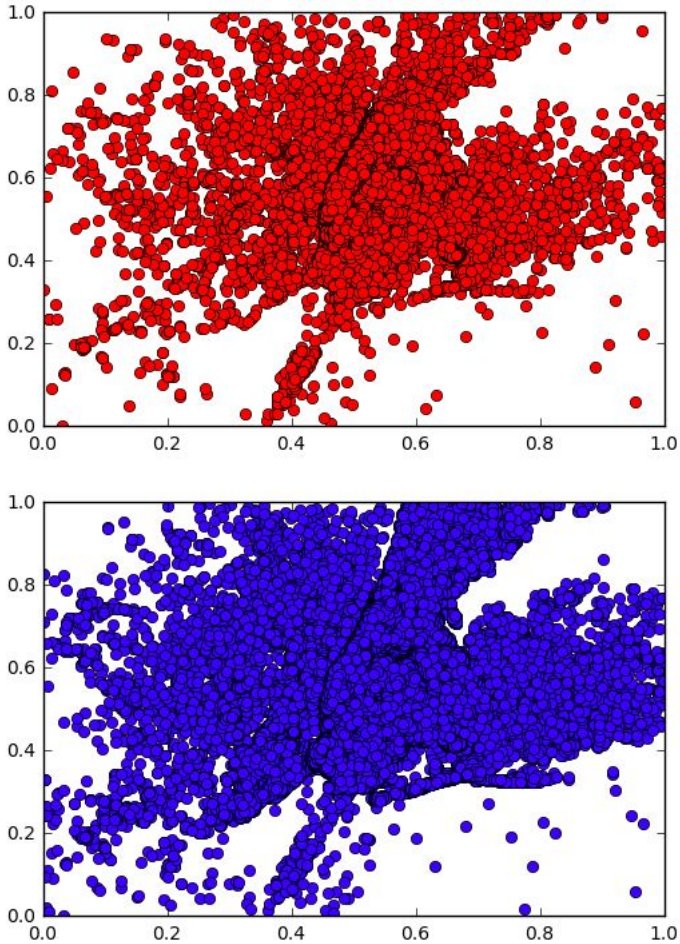

Figure 2: Visualization of the starting and end points with normalization

Part II: knn esimtaion

After seeing the graph, I decided to try knn because knn is a non-parametric lazy learning algorithm and it is easy to set the baseline. Besides, compared to traditional regression methods, knn is more likely to capture seasonal fluctuation.

I implemented the knn by using the scikit-learn KneighborsRegressor and using the random sample(separate the train.csv into 8:2 for training and testing data). The index that I rely to check estiamtion accuracy is R squred. The R squared value for the testing data is 0.50-0.55, and that for the ( ) is 0.90- 0.93. The fluctuation is due to random sample splitting and difference choices of k. The following figure shows the results when k is set to 20 and 200, respectively, and the number of instance is set to 400 and 4000, respectively.

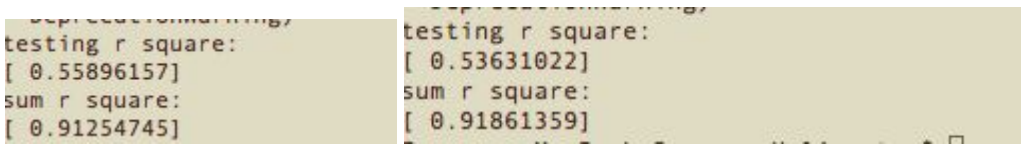

Figure 3: Examples of R squared results from knn estimation

In the end, I attached a csv file which contains all the prediction using knn in the output folder. (The size of my training set was 100,000 since using all of the data in train.csv is so time consuming and I cannot get the result before the deadline of this assignment.)

Part III: STL analysis

After finishing estimating using the knn, I thought a better way to do this is using STL analysis, but I need to do this wisely to fit different types of route. Since it is reasonable that cars from Brooklyn to Manhattan are more likely to have the same pattern than cars from others areas with different population destinies.

So I thought it is necessary to do a clustering on starting points and end points first, instead of separating the dataset and do the STL on each pair of different starting and end points.

I firstly used ***dbscan*** since it is a density based graph, robust to outliers, and does not require an initial of centroids nor a number of cluster. When implementing ***dbscan***, I'm constrained by the laptop capacity, therefore only around 10000 points are plotted as an illustration in the following figures. (I take the first plot as starting points and the second one as end points.)

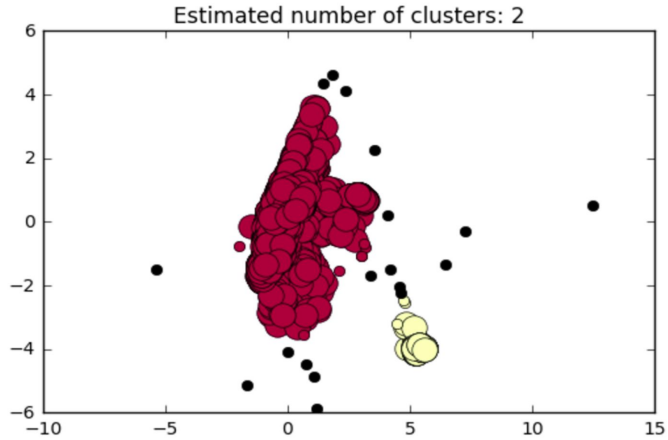

Figure 4A: Clustering results using ***dbscan*** (shown only the starting points)

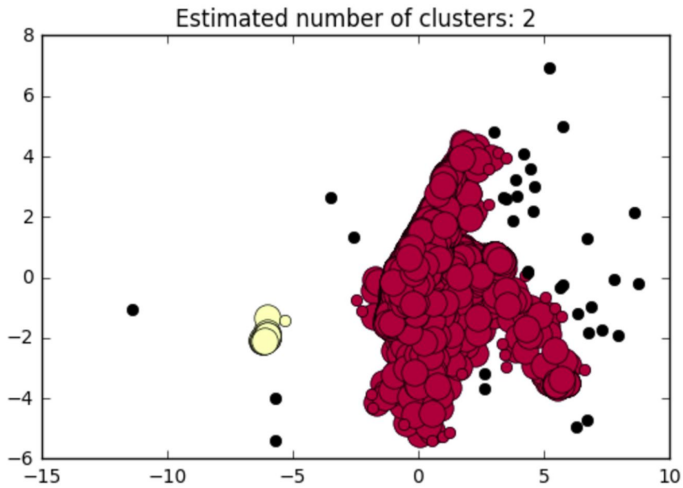

Figure 4A: Clustering results using ***dbscan*** (shown only the end points)

Another drawback for ***dbscan*** is if people want to go from an outlier point to another outlier point, there is no way to tell if he would across the downtown which would cause a huge bias for some user.I believe this is not preferred.

Considering the computational power and the drawbacks I mentioned, I chose kmeans instead. One of the the advantages of kmeans in this case would be that it was easy to choose the initial points since we already know it is in New York. I chose the center of Manhattan, Brooklyn and Newark(or New Port?) as my three initial centriods and I set k = 3. After I did kmeans on the starting points and the end points separately, I have 3 clusters for the starting and end points each. Then, I group them as 9 pairs and map the whole dateset into these 9 sub-dataset. After that, I did the STL analysis on each dataset and predict each one using the group it belongs.

Another big advantage of this approach is that it could be implemented on a cluster since all of the computation is parallel.

For the STL analysis for each route, I present the following figures with time on the horizontal axis and velocity on the vertical axis.

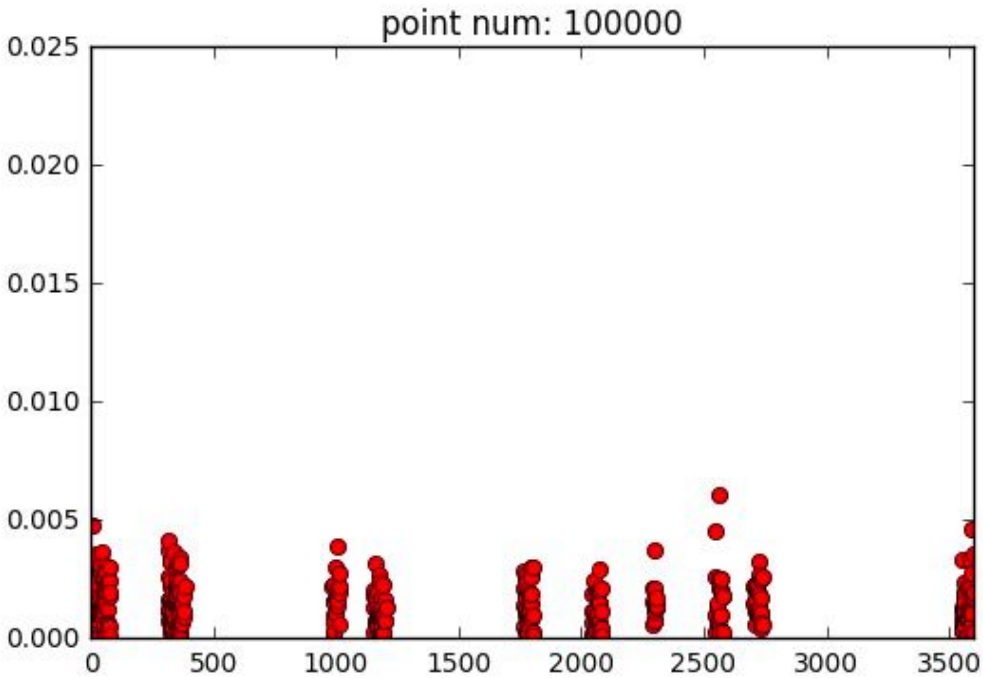

Figure 5: Velocity plot within an hour

I used the mean to represent the value for this hour.
Here is a week, same approach:

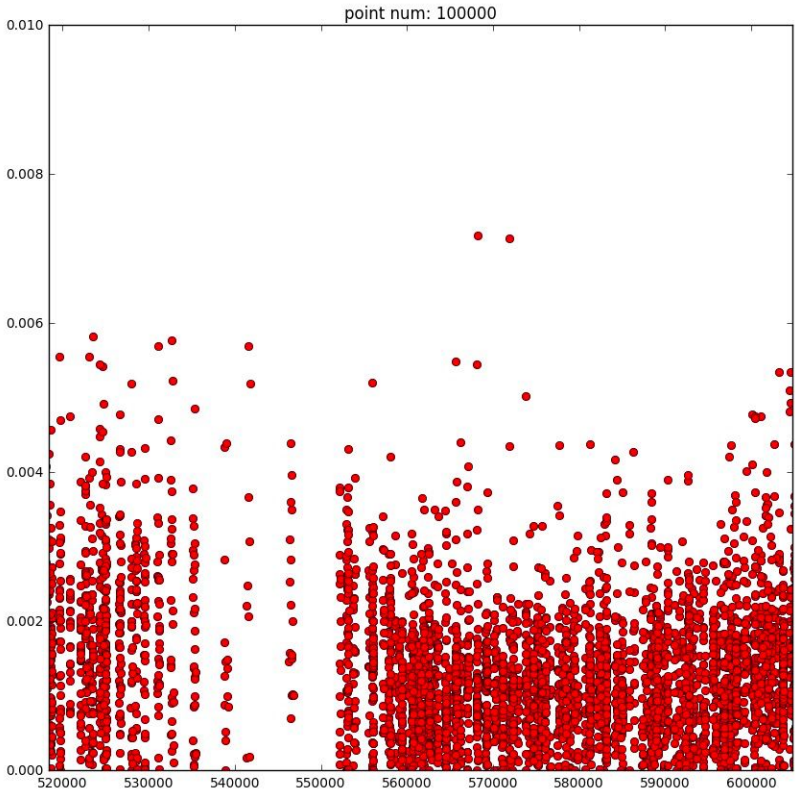

Figure 6: Velocity plot within a week

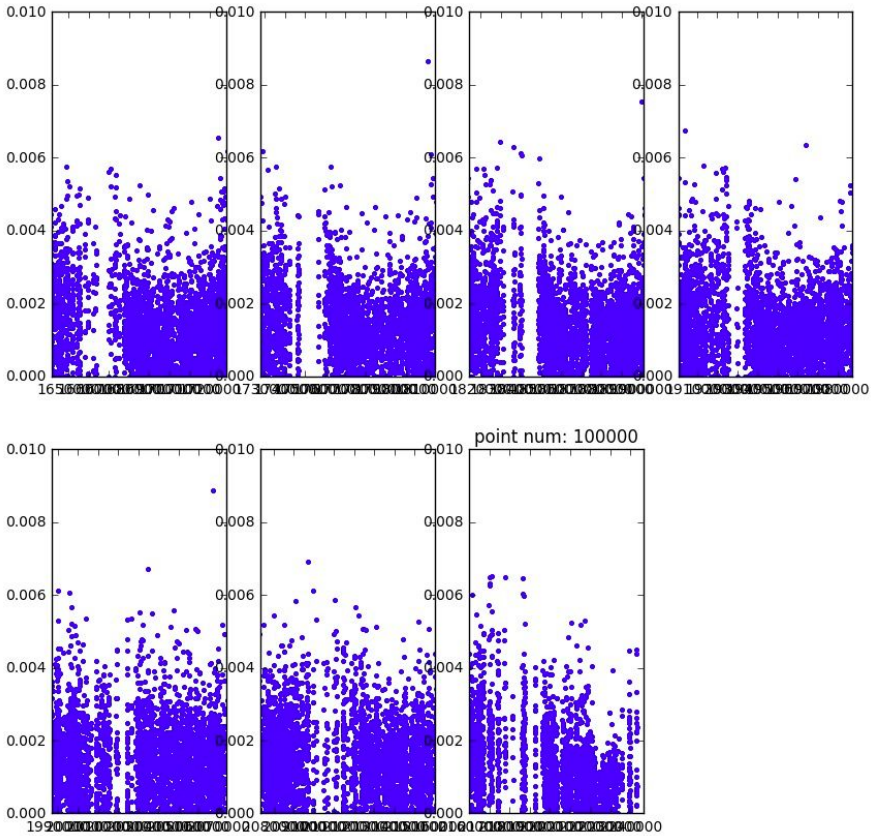

Figure 7: Velocity plot within a week, by days

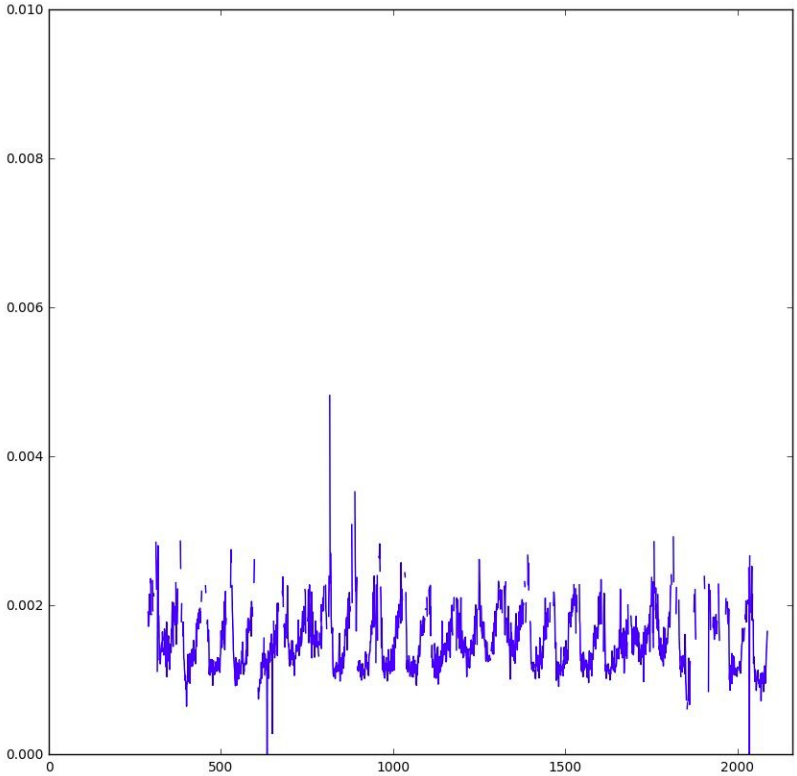

Figure 8: Moving average of velocity (averaged over minute) within 25 days

After I plotted every 10 mins means on the graph, I got the time distribution. Here is one day's:

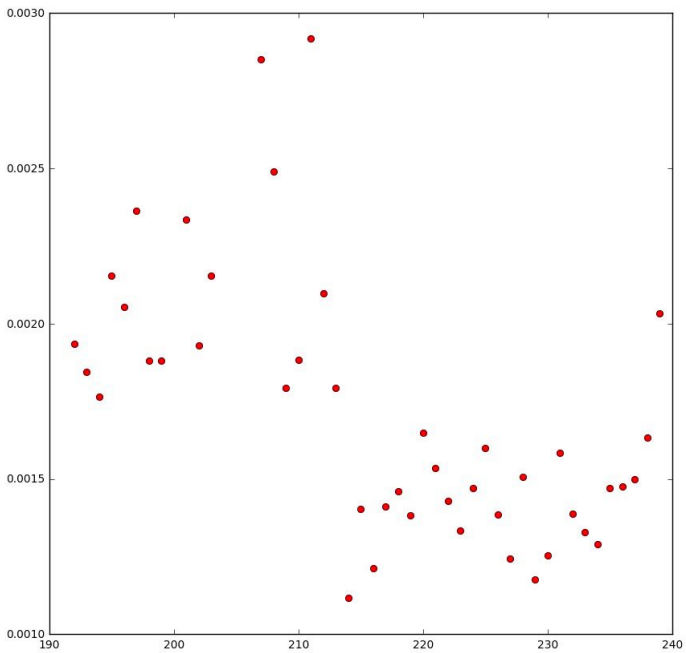

Figure 9: Moving average of velocity (averaged over 10mins) within one day

And here is one month's:

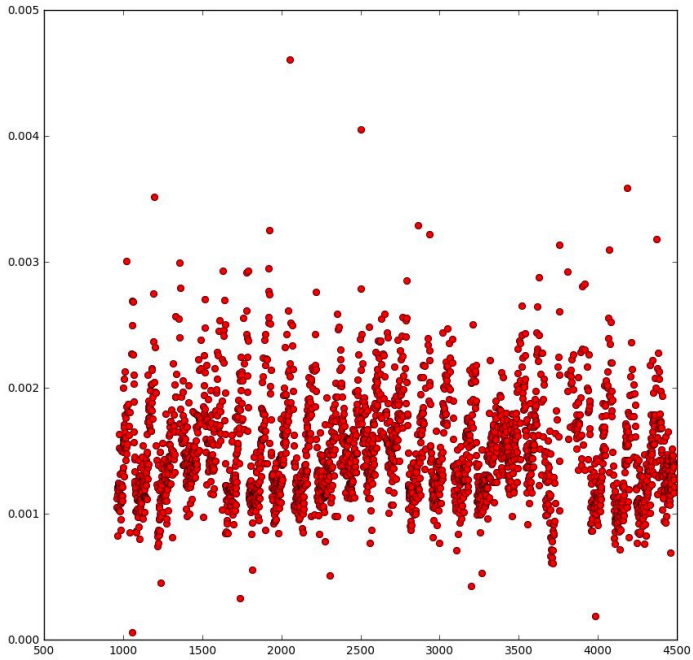

Figure 10: Moving average of velocity (averaged over 10mins) within one month

Part V: Comments and further thinking:

Due to the limited time, I did not finish implementing the STL on each kind of root. (My biggest problem is that most of the package of STL for python could only catch the seasonality of days, but what I want is to the hours even 10-20 mins, I tried to implement myself but I do not have enough time for the massive of coding work.) However I am positive about this approach.

Besides, I believe there are plenty of things that could improve for this task:

1. STL analysis could not be easily parallel-implemented as deep learning. Some deep learning model seems to be better choices too such as HTM(Nupic) algorithm. Or, I believe, approach like Karman Filter may also have a better performance.
2. There should be a better way out there to do the clustering (wiser choice of initials with hierarchical clustering and so on).
3. I could have done a better virtualization, maybe some dynamic changing simulation if given more time.
4. Some ensamble method could be of great fit here such as boosting and RF.
5. I would also like to try out GL-Mix and GAME to solve such a problem.