Statistics Capstone Project Final Report

Ruonan Hao
Taxi Data Team

May 10, 2017

1. Problem Description

In our daily life, taxi is a fast, convenient and comfortable transportation for people to go out. However, unlike other public transportation such as subway, taxi is untimed and could distributed anywhere in the city. These characteristics caused some potential problems which lead to inefficient transportation and sometime even energy waste. For example, taxi drivers could not find passengers resulting in high empty driving rate and vice versa. Passengers is always waiting for the taxi and hard to catch one. Therefore, the exact and rational planning dispatch of taxi routing will reduce the rate of empty driving which can raise efficiency of matching drivers and clients. If we know the request in a certain area and the travel time from one location to another, we can send the nearest taxi driver to the passenger in shortest time which will also lower the transportation cost.

There are mainly three problems we are interested: Travel Time Distribution Prediction, Request Distribution Prediction, and Taxi Optimal Dispatch. The reason why we take travel time into consideration lies in that we need to use travel time as a measurement to estimate how long it takes drivers from one place to another. It is true that we could use the real distance as the measurement, but usually the travel time varies from weekdays to weekends, from peak time to off-pick time, which will be inaccurate if we use distance as a measurement. Therefore, travel time might be a good point to start with. We could predict the travel time given the departure time, pick up and drop off location by analyzing large amounts of records of them. The second problem is to predict request distribution. This metric is useful to do the dispatch problem because it gives us an estimate on how many cars we need to dispatch in the city. Intuitively, the most important features that will affect the number of requests are location and time. Therefore, we plan to analyze the connection among them to predict the number of requests given the departure time and location. The last question we are interested is taxi optimal dispatch. This question is based on the first two parts. Since we have already known the number of requests at each place and the time needed to travel from one place to another, then the question becomes that how we could dispatch taxis to save time and energy if we were given the distribution of available taxis, which means to minimize total travel time of all cars in the city.

We have taxi trip records from January 2009 to June 2016, and each month contains more than 10 million trips. The data contains 19 variables with no missing value. However, we will not use all of them. The most important factors we are going to use is time, location and distance. The features that we want to focus on are pick up date and time, drop off date and time, pick up location (in longitude and latitude), drop off location (in longitude and latitude), travel time and trip distance. We will also transform the data into different format to predict request. In this kind of format, we will separate the data according to the time and location, then count the number of request within in each area. The details about how we manipulate the data will be given afterwards.

2. Data Description

The data were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP). The yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. The CSV file for each month is around 1-3 gigabytes containing more than 10,000,000 trips per file.The features that we want to focus on are pick up date and time, drop off date and time, pick up location (in longitude and latitude), drop off location (in longitude and latitude) and trip distance. After conducting data preprocessing steps such as removing outliers and spliting date into day and time, we get a clean format data as follow:

Table 1: Data Preview

pickup_ date	pickup_ time	dropoff_ date	dropoff_ time	week	travel_ time	pickup_ longitude
1/28/16	13:50:40	1/28/16	13:58:38	5	00:07:58	-73.97525787
1/28/16	21:24:48	1/28/16	21:40:01	5	00:15:13	-73.99943542
1/15/16	18:44:00	1/15/16	18:50:53	6	00:06:53	-74.00698853
1/9/16	02:18:11	1/9/16	02:33:21	7	00:15:10	-74.00422668
pickup_ latitude	dropoff_ longitude	dropoff_ latitude	passenger_ count	trip_ distance	total_ amount	
40.76118088	-73.98313904	40.75004959	2	0.88	8.76	
40.73001862	-73.97425079	40.74696732	1	2.5	15.35	
40.74871063	-73.9901123	40.76707077	6	1.67	9.3	
40.72189713	-73.95005035	40.68475723	1	4.48	18.8	

We have separated pick up and drop off date and time into “date” and “time”. Since travel time might differ from weekdays to weekends, we labeled the date to represent whether a day is weekday or weekend by using functions (e.g. IDateTime, wday) in “data.table” package. The relationship among different features and travel time can be considered like this: “pick up and drop off time” features can represent the change among one day. “pick up and drop off location” features can represent the change of locations; either is near to metropolis or is far away from downtown. “Weeks” features can represent the influence of weekends or weekdays on the travel time. With regards to predicting the taxi request, the influence of these features are basically same as predicting travel time.

The following figure shows the distribution of car in New York city. From the plot, we can see that the area in the middle is quite intense with cars while the rest are kind of sparse.

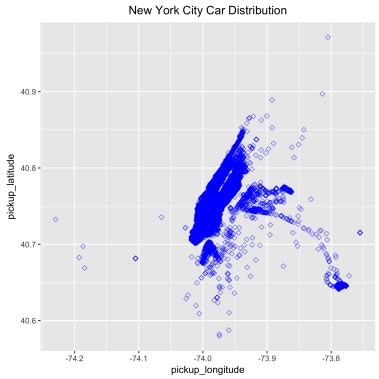

Figure 1

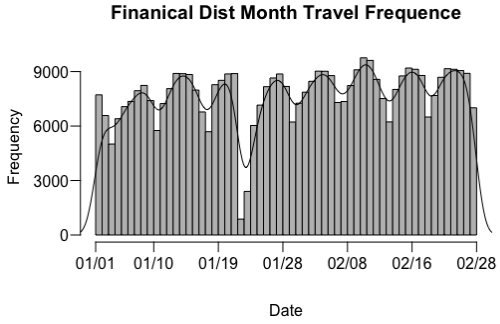

Figure 2

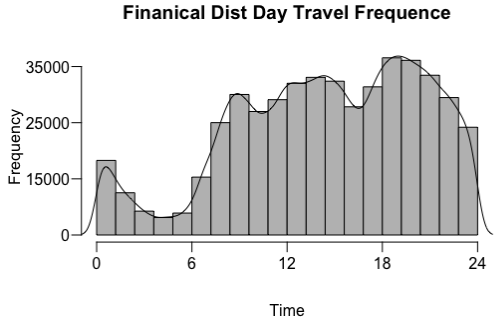

Figure 3

Figure 2 and Figure 3 are the travel frequency of Financial District. We used the data from January and February to tract the regular pattern of travel frequency.The plot on the left is monthly travel frequency. The y-axis of it is the travel frequency for each day. From the plot we can see that The pattern follows a periodic variation with high frequency during weekdays and lower frequency during weekends. There is an extreme drop on January 23 and January 24. We serched for the news and found there is a heavy blizzard on that weekend which we deduce might cause this phenomenon. The plot on the right is daily travel frequency in financial district area. The y-axis of it is the total travel frequency for each hour in two months. We calculate it by counting trip records in one month according to hours(i.e. counting how many trip records do we have from 12pm to 1pm each day and add them together). It also has a regular pattern: a low frequency during midnight to early in the morning and a high frequency during the evening.

Figure 4 and Figure 5 are the travel time scatterplot from financial district to LaGuardia airport, monthly and daily. The pattern for travel time is almost the same as travel frequency. Monthly travel time also follows a periodic variation every week: weekdays has a relatively longer travel time and weekends has shorter travel time. Daily travel time has longer journey during evening and shorter time during midnight to early in the morning. This seems reasonable because the more cars, the easier there will be traffic jams according to our common sense. The travel frequency and travel time sort of has a positive correlation.

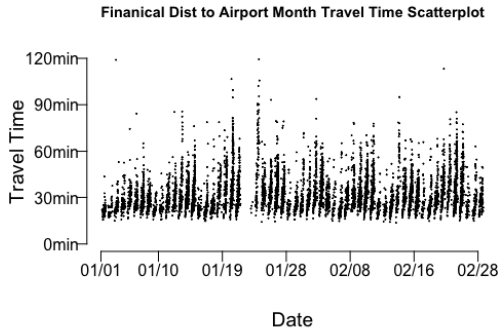

Figure 4

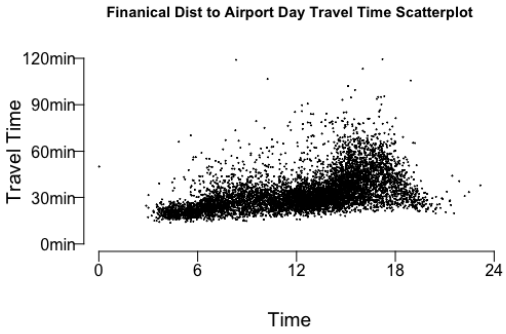

Figure 5

3. Methods

The variables we choose from the original dataset are pick-up date, pick-up time, drop-off date, drop-off time, week, pick-up latitude, pick-up longitude, drop-off latitude, drop-off longitude, passenger count, trip distance, total amount and travel time. In total there are 13 variables. The detailed mathematical terminology is given below.

Table 2: Data Variables

x_1	x_2	x_3	x_4	x_5	y
pick-up time	drop-off time	week	pick-up latitude	pick-up longitude	travel time
x_6	x_7	x_8	x_9	x_{10}	
drop-off latitude	drop-off longitude	passenger count	trip distance	total amount	

3.1 K-Nearest Neighbors

For travel time and request prediction, the model we have applied is K-Nearest neighbors. K-Nearest neighbors can calculate the distance between different data points and find the nearest k points. It can be used both for classification and regression. In our situation, it is a regression problem. I will use travel time as an example to illustrate further.

What we want to obtain is the travel time given the pickup time(x_1), pickup location (x_4, x_5) and drop-off location(x_6, x_7). We assumed that data points in a certain region is almost the same (e.g. if they have same pickup time, pickup location and drop-off location, they might have similar travel time and requests). When we are given a new data point, it only need to go through for all possible combinations, search for data points in training set that have the most similar features with it and then predict the travel time with the mean of k points.

K-Nearest neighbor makes a lot of sense in this situation but it can cause a long time to predict as training data size is increasing. So we also considered ridge regressions with Gaussian kernels as a substitude to predict travel time because it can give us instant prediction after training. However, since this method could not handle large dataset and also R-squared is not ideal enough compared with KNN, we decided to go with K-Nearest Neighbors to do the prediction. Consider the amount of data we have (i.e. around 30 gigabytes for one year), we need to come up with a better solution to optimize our searching algorithm.

There are basically two ideas to optimize searching algorithm. The first solution is subset searching algorithm (our name). This algorithm first separates the location into different subsets according to features excluding those points that are far away from the prediction point, and then run KNN in the relatively small subset. The inspiration of this method comes from the principle of KNN that is to find the nearest point. As given above, one factor that influence the speed of KKN is the size of training set. Therefore, it is natural to come up with the idea that we can subset training data before apply KNN. If we can exclude irrelevant data points, the time of searching could be reduced heavily.

This optimizing algorithm contains the following 3 steps: First, we first sort the data by pick-up time and distance, then cut the data uniformly into 729 small subsets. The distance is computed by pick-up longitude and latitude and drop-off longitude and latitude.The second step is to locate the prediction points. When we want to predict a point, we first locate it into the corresponding subset. We choose the nearest 9 subsets and combine them into one training set. The last step is to apply KNN with the training set build in previous step. Since we only use about 1% of the total data to predict one point, the prediction is much faster than before.

Another solution for optimizing searching algorithm is by using bagging and parallel computing. What we did is to separate the data into equal length subsets (each have same rows of data), searching the given new point in each subset and then take mean of all predictions as final result. However, since the result for this method did not improve speed too much nor did the accuracy for prediction, I plan not to demonstrate into details how it works. We also learned from another course that bagging usually did not work well on K-Nearest Neighbors method. It would be great if we could learn this before we try in our dataset.

3.2 Optimal Dispatch

After we got the travel time and request by using K-Nearest Neighbors, we can do the optimal dispatch with these two predictions. There are basically two goals for optimal dispatch. The first one is to minimize

total travel time of all cars in the city. The second goal is to find the minimum waiting time for longest-waiting customer. The solution to the second goal involves some advance methods in graph theory such as floyd algorithm and ISAP algorithm. It is so complicated that the algorithm takes forever to run if sample size is increasing. Thus, we decided to choose the first method to do the optimal dispatch which actually gives us pretty decent results.

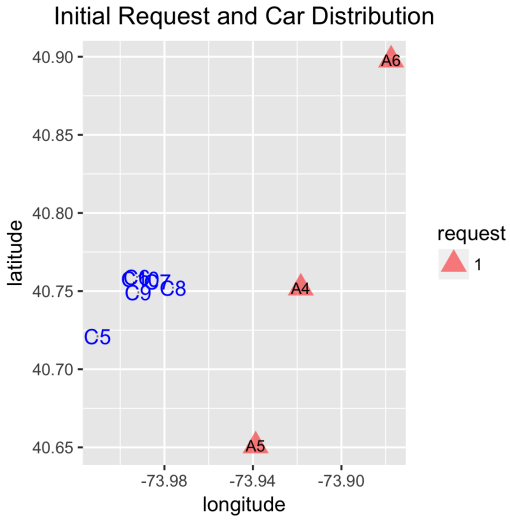

Figure 6

The distribution of requests and cars are shown in Figure 6. Blue points represent cars while red triangles represent the location that drivers need to go. The size of the red triangles represents the number of requests. The requests in Figure 6 are all one for each place.

The mathematical formulation for minimizing total travel time is like this: There are k areas in the city with different requests. The requests for each area are r_1, ...,r_k. Then the total number of requests is $N = r_1 + ... + r_k$. Because the number of cars equals to total number of requests, so we will dispatch N cars to k different areas. Assuming we have cars (blue letters) randomly distributed in the city. Treat the location of cars as pick-up locations while treat areas(red triangles) as drop-off locations. We can use KNN model we built before to predict the travel time between two points (given specific day and time). Then this problem becomes a traditional transportation problem in operations research. What we need to solve is only a linear programming problem. We need to get the minimum travel time under the restriction of two equations given in equation (1) and (2).

$$\text{Objective Function:} \quad \min \mathrm{f} = \sum_{i=1}^{k}\sum_{j=1}^{n} t_{ij}X_{ij}, \tag{1}$$

$$\text{subject to} \quad \begin{cases} \sum_{j=1}^{n} X_{ij} = r_i \quad i = 1, ..., k \\ \sum_{i=1}^{k} X_{ij} = c_i \quad j = 1, ..., n \\ \quad X_{ij} = 1 \text{ or } 0, i = 1, ..., k, j = 1, ..., n \end{cases} \tag{2}$$

t_{ij} in the objective function means travel time from car i to area j; X_{ij} is a binary variable represent whether car i will be dispatched to area j. r_{ij} means the number of requests for each area (in total there are k areas). c_i means the number of cars or drivers for each car location. It is possible to set more drivers at each car location, but here we only set one car at each point. So actually c_i equals to 1.

The matrix for our transportation problem is shown in Table 3. It deals with sources where a bunch of cars are available and destinations where the requests the area is demanded. The algorithms for solving the problem are based on this matrix representation. The travel time of taxi from sources to destinations are indicated by the entries in the matrix. All entries are non-zero because we can predict travel time between each two points on the map. The total number of cars and requests are shown along the margins of the matrix. As I stated before, the total number of cars equals to the total number of requests. In our example, we have 6 requests for 3 locations. So we need to dispatch 6 cars to these 3 areas.

Table 3: Matrix Representation for Dispatch

	C5	C6	C7	C8	C9	C10	request
A4	t_{54}	t_{64}	t_{74}	t_{84}	t_{94}	$t_{10,4}$	1
A5	t_{55}	t_{65}	t_{75}	t_{85}	t_{95}	$t_{10,5}$	1
A6	t_{56}	t_{66}	t_{76}	t_{86}	t_{96}	$t_{10,6}$	4
cars	1	1	1	1	1	1	6

Table 4: Dispatch Output

	C_5	C_6	C_7	C_8	C_9	C_{10}	request
A_4	0	0	1	0	0	0	1
A_5	0	0	0	0	1	0	1
A_6	1	1	0	1	0	1	4
cars	1	1	1	1	1	1	6

The optimal solution usually looks like Table 4. The number in the table is X_{ij} which is a binary variable represent whether car C_i dispatch to area A_j. For instance, from the table we can know that C_7 will dispatch to A_4 whereas C_8, C_{10} will dispatch to A_6.

4. Results

4.1 Data Preprocessing

Before we apply our model to the data, we need to conduct data preprocessing step. The procedures of data preprocessing for travel time and request prediction are illustrated as follow.

For travel time, we transformed the "date" and "time" data into numerical which can be used for plotting and predication. Then we got travel time by using drop off time subtract pick up time from the original data as our y-label. Another important work we have done is to detect outliers and wrong data. When we conducted data preprocessing, we found that there is a bunch of data that have drop off date is earlier than pick up date, which we guess was caused by falsely manual input. Because we have enough data and the outliers and wrong data do not account for a large proportion, we delete them all. The method of detecting outlier is to delete all travel time less than 1 minute and larger than 3 hours. Generally, we deleted around 1% of the original dataset size. An example of data is given in Table 5.

Table 5: Travel Time Data Preview

pickup time(x_1)	week(x_3)	pickup longitude(x_4)	pickup latitude(x_5)	drop-off longitude(x_6)	drop-off latitude(x_7)	travel time(y)
0	3	-73.97674561	40.76515198	-74.00426483	40.74612808	475
0	3	-73.98348236	40.76792526	-74.0059433	40.73316574	666
0	3	-73.78202057	40.64480972	-73.97454071	40.67576981	1866
1	3	-73.78877258	40.64775848	-73.82920837	40.71234512	963

The data preprocessing procedure for request is a little bit complicated because we need to count the trip records according to time and location. The following steps demonstrate how we count requests. First, we divided one day into 48 time intervals, each lasts half an hour. Then separate one-month data into 1440 intervals. This number could vary from month to month. If there are 30 days in one month, then the total intervals will equal to $48 * 30 = 1440$. Second, divided pick-up longitude and pick-up latitude into 10000 subareas and delete areas that are not in New York. Then count request numbers given time and location. After data preprocessing, we got data looks as Table 6.

Table 6: Request Data Preview

time	longitude	latitude	request	week	day	time_withinday
899	51	70	530	2	19	35
71	53	72	96	6	2	23
611	50	64	125	4	13	35
323	50	65	222	4	7	35

The "time" column represents which time interval in one month the trip record belong to. For example, time equals to 899 means it belongs to the 35^{th} time interval on the 18^{th} days of a month. "time_withindate" column stands for which time interval the trip record is in within one day. The "longitude" and "latitude" columns represent which box the trip records happened on the $2-d$ map. "request" column is calculated by counting the number of trip records within each area.

4.2 Training and Application

We trained the data with K-Nearest neighbors in R. The result for travel time is given in Table 7 whereas Table 8 gives the prediction accuracy for request. Figure 7 also shows the request distribution in New York city at 8am on July 1^{st} 2015.

Table 7: Travel Time Prediction Accuracy

Training R^2	Validation R^2	MAPE	Time for one point
0.8908039	0.897633	0.192743	10.8s

Table 8: Request Prediction Accuracy

Training R^2	Validation R^2
0.9528431	0.9256047

Figure 7

For optimal distpatch, we pick up 25 areas from our dataset. In total, there are 87 requests for all 25 areas. Thus, we randomly distributed 87 cars on the map and predict travel time between each car and each request location by using the KNN model we built before. Formulate the predictions in matrix representation. Then we used "lpSolve" package in R to solve this linear programming problem. The minimum total travel time we got from the program is 91147.98 seconds (around 25 hours) which means each car will spend around 17 minutes to pick-up passengers on average. The detailed optimal dispatch is shown in Table 9.

For display purpose, this matrix is only part of the output. The entire matrix has 25 rows and 87 columns. Most of the matrix entries are zero. From the matrix, we can know that Car 1 will dispatch to Area 10 whereas Car 3 will dispatch to Area 11. We visualize the result in Figue 8. The left plot shows the initial

Table 9: Optimal Dispatch Output

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{19}	C_{22}	C_{23}
A_{10}	1	0	0	0	0	0	1	0	0	0	0	0
A_{11}	0	0	1	0	1	0	0	0	0	1	1	0
A_{15}	0	0	0	0	0	0	0	0	1	0	0	0
A_{17}	0	0	0	1	0	0	0	0	0	0	0	0
A_{18}	0	1	0	0	0	0	0	0	0	0	0	0

distribution whereas the right plot shows the distribution of cars after dispatch. We set the pick-up time at Wednesday 6pm. The legend in the plot is only a reference to match the size of red triangles with the number of requests in that area. The average travel time could be shorter if we use more cars to solve the linear programming problem. For instance, if we have 870 cars, then we can always find a new car go to a certain area with shorter travel time. In that case, the average travel time would be shorter.

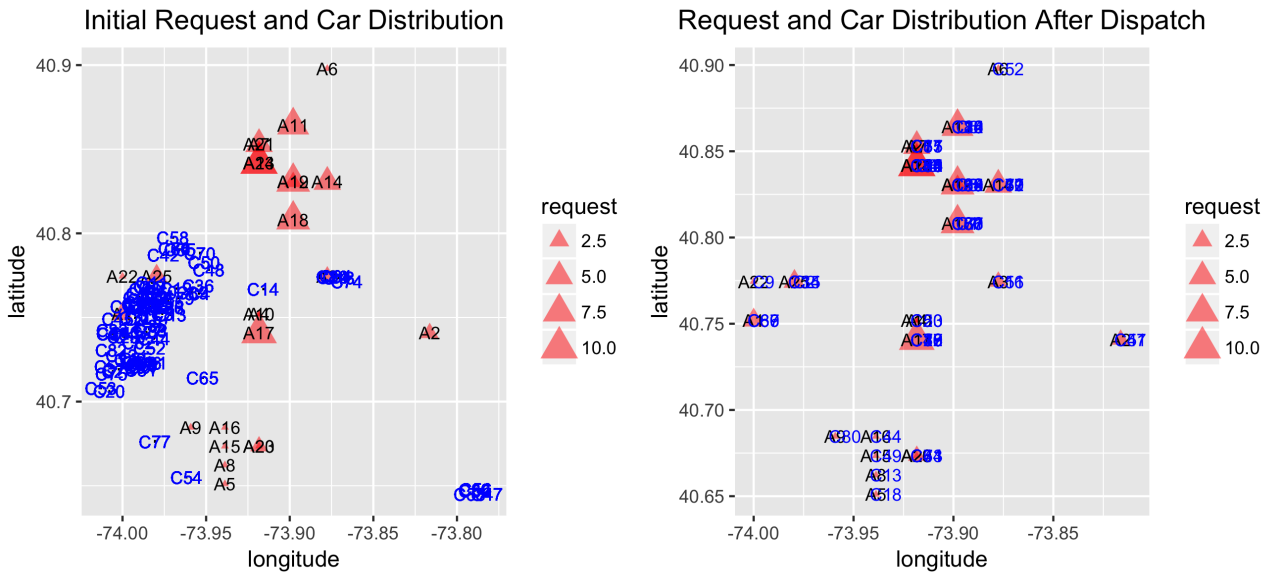

Figure 8

5. Conclusion

We have finished our inital goals for the entire project which are travel time prediction, request prediction and optimal dispatch. For travel time prediction, we used K-Nearest Neighbors method to predict travel time given pick-up time, pick-up location and drop-off location. The accuracy of the model is around 0.9. The time for predicting one point is around 10 seconds. It does not reach our best expectation but it is still good enough for dispatch. For request prediction, we also built a K-Nearest Neighbors model to predict request given departure time and location. We achieved an accuracy at around 0.95 which is quite ideal. The time for predicting one point is around 6 seconds. For optimal dispatch, we fomulated the problem into transportation problem in operations research. The goal for this solution is to minimize the total travel time in the city. It is a sub-optimal solution compared with the solution that minimize the longest waiting time, but it gives us a decent results for optimal dispatch which can clearly show which direction the car should go as well as provding the total minimum travel time. We pick up 25 areas in New York city as an example to illustrate the result. The minimized total travel time is 91147.98 seconds (around 25 hours) which means

each car will spend 17 minutes to pick-up the passenger on average. The average travel time could be shorter if we use more cars to solve the linear programming problem.

The problems we encountered during the project are basically in three parts: data preprocessing, optimize searching algorithm and problem fomulation for dispatch. For data preprocessing, even though there is no missing values in our data, we still need to determind which features should we use as well as creating new features that could reduce the bias of our model. In addition, apart from the regular data manipulation on date type data, we also need to detect and remove outilers for each feature, which is kind of time consuming. The second challenge (i.e. optimize searching algorithm) costs us most of the time. The methods we have tried including ridge regression with Gaussian kernels, bagging KNN with parallel computing and subset searching algorithm. We choose subset searching because it gives us the best accuracy and the shorest predicting time. For the last challenge, we have come up with two solutions for diffent optimal dispatch goals which is minimizing total travel time versus finding the minimum waiting time for longest-waiting customer. Either solution makes sense under different situations. I personally prefer the solution with minimizing total travel time in the city. It can generalize to larger data set and can give more accuracy result as sample size increasing. As long as we have already optimized the KNN searching algorithm, solving dispatch problem would be very quickly.

If we have more time, there are indeed a number of things we could explore future. The first thing is that we want to visualize our result in Shiny app and make all process automatically. Ideally, if we are given a specific time and location, our program should first predict the distribution of requests and cars in the city, then update the plot to show which car should go to which area. After this, we probably will try to figure out how to optimize the second optimal dispatch solution which is to find the minimum waiting time for longest-waiting customer. We hope this algorithm could at least work with the same amount of data we used for the first solution. Another exploration worth to try is to find the how large dataset the optimal dispatch solution of minimum total travel time could handle. It should not have too much problem if we add more data into our linear programming system. However, adding too much data might slow down the entire system which is not what we expect. If we could figure out the limit of our linear programming system, maybe we could improve the accuracy of the result even more. Lastly, since we also have other variables in our dataset such as passenger count and trip distance, we could also explore their relation with location and time. Companies like Uber might care about it because they can use the results to fix price.

These are all the accomplishments for our project. Clearly, the problem in the real world would be more complicated because we always want to improve users' experience and make their life more convenient. All efforts are made towards this ultimate goal. Our solution to optimal dispatch is only a simple version of the real world problem, but it does not simple the question too much. What we need to consider in the future are some optimization problems like providing real time prediction and dispatch. It will require more time and knowledge for us to polish our model, but overall we are satisfied with the results we have obtained so far. Many thanks to my great teammates who we work together for the entire semester as well as the instructors who provide us with brilliant guidances and resources!

6. Reference

K-nearest neighbors algorithm, wiki.
Kernel Smoothing Methods, The element of statistical learning, Chapter 6, Trevor Hastie et.al.
Kernel ridge regression, sklearn-documentation.
Kernel-Based Learning and Multivariate Modeling, Saul Garcia.
An Investigation of Dijkstra and Floyd Algorithms in National City Traffic Advisory Procedures, Arun Kumar Sangaiah , Minghao Han , Suzi Zhang, Int. Journal of Computer Science and Mobile Computing, Vol.3 Issue.2, February- 2014, pg. 124-138.
Algorithms for the Assignment and Transportation Problems, James Munkres, Journal of the Society for Industrial and Applied Mathematics 1957 5:1, 32-38.