

Mega Taxi

-- A predictive technology startup

Final Report

Team members and authors: Sharon Cui, Tangyao Zhao,
Hongyue Lan, Lingyun Gao, Yingxin Zhang

1. Introduction:

Our project is a hypothetical new taxi startup aiming to use predictive technology to solve NYC's taxi pick up and pricing issues. The reason that we decide to proceed this topic is that we find the long waiting time and not enough pick up taxis available are two of the huge problems for Manhattan taxi riders and drivers. Due to the un-matching of supply and demand of the taxi, the pricing issue is serious and important for us to tackle. We firstly came up with the most emergent problem, which is that we are missing the accurate prediction of popularity of pick up location for the traditional taxi, Yellow Cab. We think, without considering the popularity of the picking up need, such as popular events happening in the city like the festivals, Broadway shows or Parade, or the rush hours on the weekdays, or the environmental condition such as temperature, and weather, the taxis pricing is not reasonable. Therefore, we used KRR method to predict picking up times in different locations in Manhattan for the Yellow Cab according to the relationship between the historical pickup time and the weather in Manhattan. In addition, we created a mobile app, which embedded Google Map SDK for iOS, so that we could input the pickup and destination locations. We used the same model with cost parameters, in order to predict the cost of the trip.

2. Data

The dataset we used are the NYC Yellow Taxi Data, which is from July.2015 to June.2016, and the Past Weather Data. Both of them are open resources from the following websites:

http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

<https://www.timeanddate.com/weather/@8479493/historic>

There are 12 separate csv files for 12 months and each file has the same format and size, which is about 2GB per file. Due to the slow processing of our own computer and huge dataset, we need to do some data cleaning in order to have a more efficient prediction. The raw data has 19 features(columns), which are "VendorID", "tpep_pickup_datetime", "tpep_dropoff_datetime", "passenger_count", "trip_distance", "pickup_longitude", "pickup_latitude", "RatecodeID", "store_and_fwd_flag", etc.

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag
0	2	2015-08-01 00:00:15	2015-08-01 00:36:21	1	7.22	-73.999809	40.743340	1	N
1	1	2015-08-01 00:00:16	2015-08-01 00:14:52	1	2.30	-73.977043	40.774902	1	N
2	1	2015-08-01 00:00:16	2015-08-01 00:06:30	1	1.50	-73.959122	40.775127	1	N
3	1	2015-08-01 00:00:16	2015-08-01 00:06:18	1	0.90	-73.976624	40.780746	1	N
4	2	2015-08-01 00:00:16	2015-08-01 00:16:28	1	2.44	-73.978592	40.785919	1	N

dropoff_longitude	dropoff_latitude	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount
-73.942848	40.806622	2	29.5	0.5	0.5	0.00	0.0	0.3	30.80
-73.978256	40.749863	1	12.0	0.5	0.5	2.93	0.0	0.3	16.23
-73.980392	40.782314	1	7.0	0.5	0.5	1.65	0.0	0.3	9.95
-73.970558	40.788845	1	6.0	0.5	0.5	1.45	0.0	0.3	8.75
-73.997353	40.756302	1	13.0	0.5	0.5	2.00	0.0	0.3	16.30

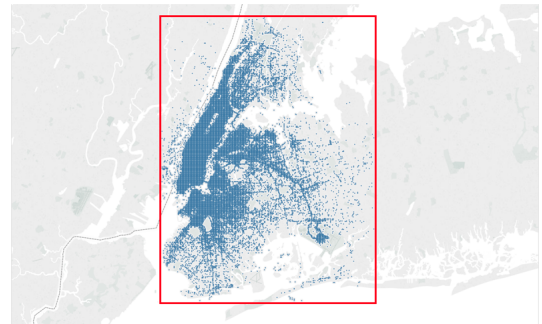
(Figure 2.1: Example of raw data for NYC Yellow Taxi.)

Data cleansing:

- We kept 12 features that will be used for future prediction and delete some unnecessary features.
- We removed some dirty records, such as nulls, negative price. Then, we set a rectangular boundary in order to extract the correct location features, since there are some invalid geographical coordinates (e.g. South America, Pacific Ocean, *Figure 2.2*). *Figure 2.3* shows the correct locations.
- We rounded the correct longitude and latitude to the nearest thousands because 3-decimal points will be accurate enough to show on the map.
- We also added an additional feature “duration”, which records the duration of each trip in seconds for the convenience of future prediction.



(Figure 2.2: South America, Pacific Ocean)



(Figure 2.3: Correction locations)




The graphs below show the result of our data cleaning, which is half of the original data size.

	tpep_pickup_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude
0	2015-08-01 00:00:15	1	7.22	-74.000	40.743
1	2015-08-01 00:00:16	1	2.30	-73.977	40.775
2	2015-08-01 00:00:16	1	1.50	-73.959	40.775

dropoff_longitude	dropoff_latitude	payment_type	fare_amount	tip_amount	total_amount	duration
-73.943	40.807	2	29.5	0.00	30.80	2166.0
-73.978	40.750	1	12.0	2.93	16.23	876.0
-73.980	40.782	1	7.0	1.65	9.95	374.0

(Figure 2.4: Result of data cleaning, half of the original size)

Past weather is the other data set we used. Here are the raw data from the website:

Time		Temp	Weather	Wind		Humidity	Barometer	Visibility
10:51 am		45 °F	Sunny.	13 mph	↘	33%	29.72 "Hg	10 mi
11:51 am		45 °F	Sunny.	14 mph	→	31%	29.75 "Hg	10 mi
12:51 pm		46 °F	Sunny.	15 mph	↘	30%	29.77 "Hg	10 mi

(Figure 2.5: Raw weather data)

In order to apply these data to make better predictions, we used crawler to export the data into csv file and did some data cleansing as well. We extracted following features, Time, Temp, Weather, Wind, Humidity, Barometer, Visibility. We eliminated repeating time and took the average of their humidity, wind, etc. We also added approximated features for the missing time. Below is part of the cleaned weather data:

1	Time	Temp	Weather	Wind	Humidity	Barometer	Visibility
2	01 00:51	28	Clear	10	43	30.14	10
3	01 01:51	28	Clear	9	47	30.14	10
4	01 02:51	27	Clear	7	46	30.14	10
5	01 03:51	27	Clear	3	42	30.12	10

(Figure 2.6: Cleaned weather data)

For the pickup number prediction, we combined the total passenger count per hour and weather data into one dataframe and used for prediction.

For the time series forecast, we extracted “tpep_pickup_datetime” and “passenger_count” from taxi data. Then we created a new csv file, which contains a particular hour and number of passengers travelling in that hour. And we read this data as time series to forecast the hourly pick-up passenger count in New York. Due to limited time, we can only apply time series method to aggregate hours for a broad region instead of hours for each specific location (e.g. Manhattan, Bronx, Brooklyn, Queens, etc.) We are thinking about to set different size of rectangles boundaries and apply time series to each region.

For the cost prediction for a specified trip, we extracted the following features: trip_distance, duration, total_amount from the taxi data and Temp, Weather, Wind, Visibility, Weather=clear, clouds, fog, heavy rain, light rain from the weather data. We combined these features into one dataframe and used for prediction. This graph shows part of the combined data frame.

	trip_distance	duration	Temp	Wind	Visibility	Weather=clear	Weather=cloud	Weather=fog	Weather=heavy rain
Time									
16	2.07	490.0	48	17	10	0.0	1.0	0.0	0.0
16	1.00	494.0	68	5	7	1.0	0.0	0.0	0.0
17	0.79	83037.0	68	6	7	0.0	1.0	0.0	0.0
21	1.04	448.0	57	3	9	0.0	1.0	0.0	0.0
7	0.90	413.0	63	7	10	0.0	1.0	0.0	0.0

(Figure 2.6: Example of the combined data frame.)

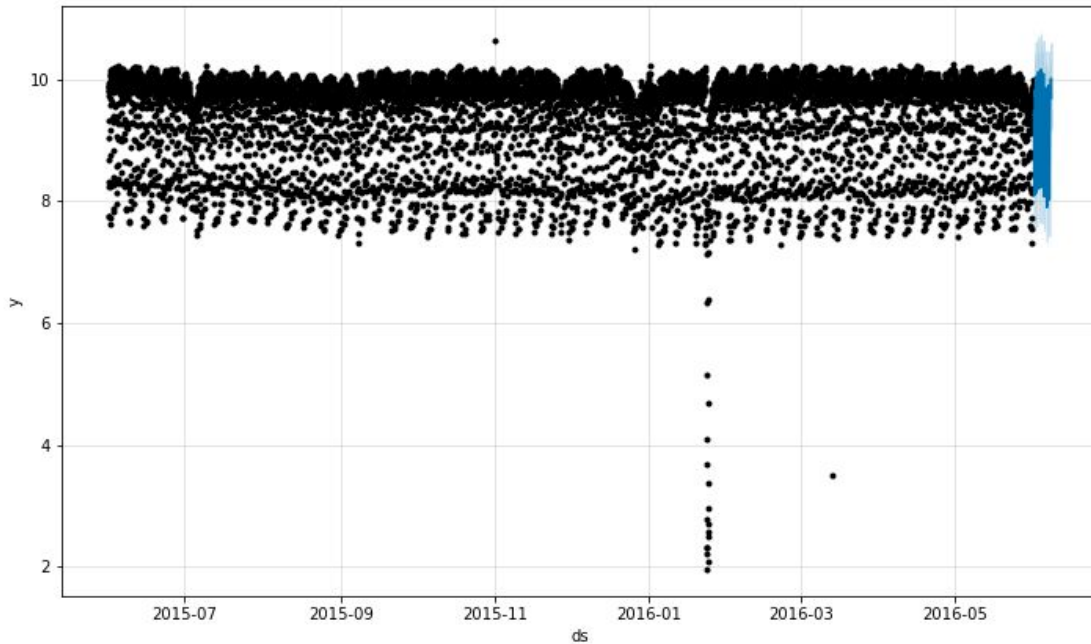
3. Technology

(a) Time Series for Predicting Pickup Number

We use Prophet to forecast the time series data because it combines the factor of trend, seasonality, and holidays. The data we use are from 6.1.2015 to 5.31.2016 because the model works best with at least one-year of historical data.

We first convert the original dataframe to a new one with only two columns indicating the hourly time and the pickup numbers, and then import the data and log-transform the pickup numbers. We fit the model by setting the “changepoint_prior_scale” to be 0.01. This value is default to be 0.05. The smaller it is, the less flexible the trend will be. Since our data is stable during a weekly period, we would want the trend to be stable.

After we fit the model, we use “make_future_dataframe” to get a dataframe that extends into the future for 168 hours. That is, the first 168 hours (7 days) in the June of 2016. Here is the result of the prediction, together with the past data. As we can see, the Prophet model is robust to large outliers, which exists on Jan 23th, 2016. However, the estimated data cannot reach value as small as the real data. We also compare the real data with the predicted data and the results are shown in the next section.



(Figure 3.1: Time series predicted total pickup number during the first week of June with historical data)

(b) Regression models for Predicting Pickup Number

We used the hourly data of New York taxi and weather from 2016-03 to 2016-04 to train and predict models for the total pickup number and also the pickup number for specific locations at a certain time. The last week of 2016-04 is the test set and the rest is the training set.

From our daily experience, we know that there is much difference between weekdays and weekends. So we just built models for weekdays and weekends separately. When predicting the pickup number for a specific time, we will use the corresponding model to predict. We normalized the numeric part of the data, vectorized the 24 hours into ‘Time=0’, ‘Time=1’, ‘Time=2’, etc and vectorized different types of weather into ‘Weather=clear’, ‘Weather=cloud’, ‘Weather=light rain’, etc. The input of the regression models is as follows:

	Temp	Wind	Humidity	Barometer	Visibility	Time=0	Time=1	Time=10	Time=11	Time=12	...	Time=6	Time=7
0	0.400000	0.290323	0.432099	0.540541	1.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0
1	0.363636	0.193548	0.481481	0.549550	1.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0
2	0.381818	0.258065	0.456790	0.567568	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
3	0.363636	0.161290	0.481481	0.612613	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
4	0.290909	0.096774	0.567901	0.639640	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0

(Figure 3.2: Input data for regression models)

We have applied ten regression models to the dataset and used grid search to select the best hyper-parameters for each model. The score we used for grid search is r square score. R square score ranges from 0 to 1 and the higher the score is , the better the model might be. We also analyzed the learning curve of models with higher r square score to find out how much we can benefit from adding more training data and whether the estimator suffers more from a variance error or a bias error.

Below is the table of all the models we used and the corresponding implementation and the r-square score. From the table we know that Kernel Ridge Regression has the highest r-square score and Gradient Boosting regression ranks the second. The r-square score of them is pretty close.

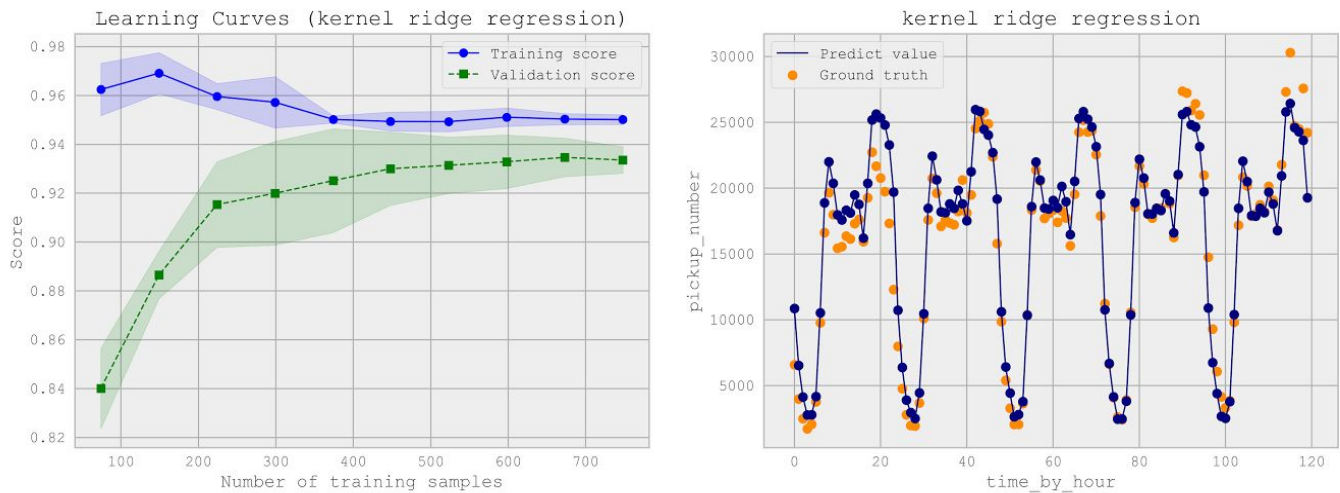
Let's see how these models perform on the test set.

Model	Implementation	r ² _score
SVR RBF kernel	SVR(kernel="rbf", C=100, gamma=0.01}	0.9175
SVR Linear kernel	SVR(kernel="linear", C=1000)	0.9177
SVR Polynomial kernel	SVR(kernel="poly", C=1000, degree=1)	0.9177
Gradient Boosting Regression	GradientBoostingRegressor(learning_rate=0.1, n_estimators=500)	0.9323
Kernel Ridge Regression	KernelRidge(kernel="rbf", alpha=0.01, gamma=0.01)	0.9329
DecisionTreeRegressor	DecisionTreeRegressor(max_depth=8)	0.8534
KNeighborsRegressor	KNeighborsRegressor(n_neighbors=4)	0.88
RandomForestRegressor	RandomForestRegressor(n_estimators=50, max_depth=8)	0.8664
AdaBoostRegressor	AdaBoostRegressor(n_estimators=500)	0.5353
BaggingRegressor	BaggingRegressor(n_estimators=500)	0.9316

(Table 3.1: Best parameters for different models using GridSearchCV)

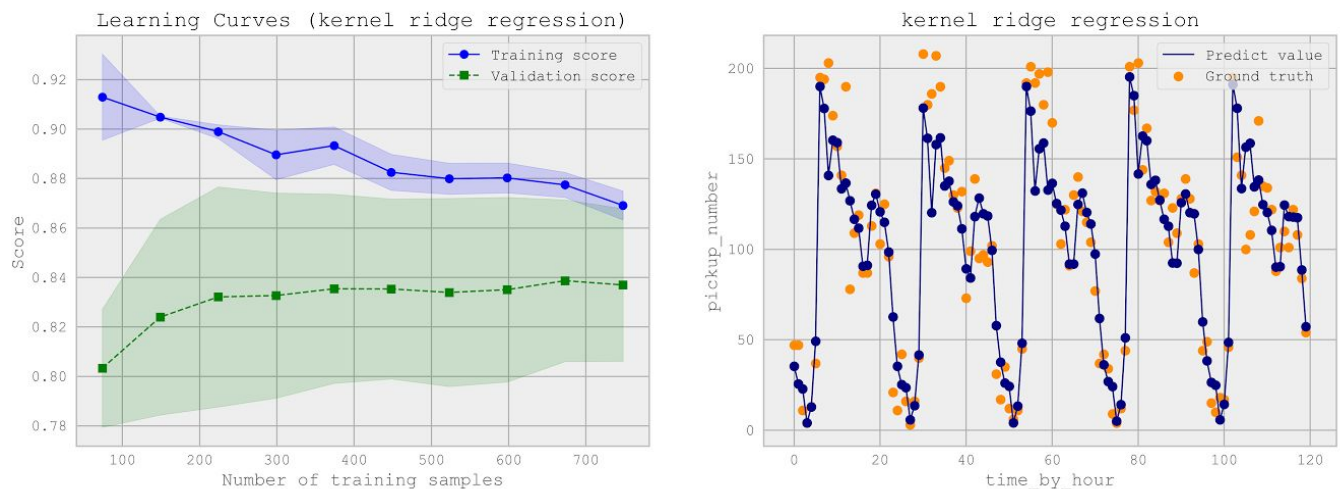
Below is the learning curve of the Kernel Ridge Regression model and the graph of our prediction for the total pick number of New York. The graph on the right side shows the predicted values, that is the blue points, and the ground truth, that is the orange points. From the graph we know that KRR model has a good performance on the test set. The left graph is the learning curve of KRR model. The validation score and the training score are both very high which indicate a small error. However, there is a gap between two curves. This indicates a high variance, that is the model is overfitting. The model might not have a good performance on a

large test set. However, we could benefit from adding more training examples, reducing the size of features or increasing the regularization parameter to address overfitting.



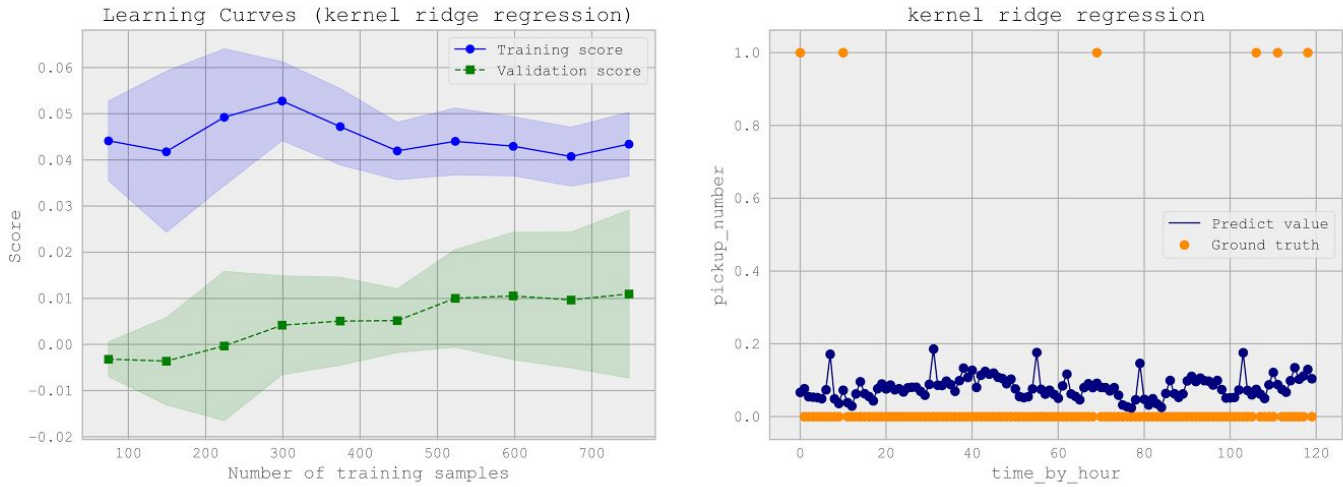
(Figure 3.3: Total pickup number of New York City throughout 24 hours (the last weekday of April 2016))

We used the same method to predict pickup number for specific locations. The graph below is one of the locations we have predicted. It is near Madison Square Garden. The model has a good performance on predicting its pickup number. Furthermore, our model might have a good performance for hot places..



(Figure 3.4: Pickup number of 73.994W 40.751N throughout 24 hours (the last weekday of April 2016))

However, another specific location, which is near Dyckman Street Subway Station, cannot be predicted well, since the pickup number is really small and most of them are zero.



(Figure 3.5: Pickup number of 73.925W 40.866N throughout 24 hours (the last weekday of April 2016))

(c) Regression models for Predicting Trip Cost

For predicting trip cost, we use the same method as predicting pickup number. We used the data of New York taxi and weather in 2015-12 to train and predict models for the trip cost at a certain time. We split the data into training set and test set using `train_test_split` in Sklearn. Since there are almost 10 million pieces of taxi data in December 2015 and we are not able to deal with so much data, we randomly selected 0.1% of data to train out model. We built models for weekdays and weekends separately. Since we want to predict trip cost, we take trip_distance, duration and weather into account. The input of the regression models is as follows:

	trip_distance	duration	Temp	Wind	Visibility	Weather=clear	Weather=cloud	Weather=fog	Weather=heavy rain
Time									
16	2.07	490.0	48	17	10	0.0	1.0	0.0	0.0
16	1.00	494.0	68	5	7	1.0	0.0	0.0	0.0
17	0.79	83037.0	68	6	7	0.0	1.0	0.0	0.0
21	1.04	448.0	57	3	9	0.0	1.0	0.0	0.0
7	0.90	413.0	63	7	10	0.0	1.0	0.0	0.0

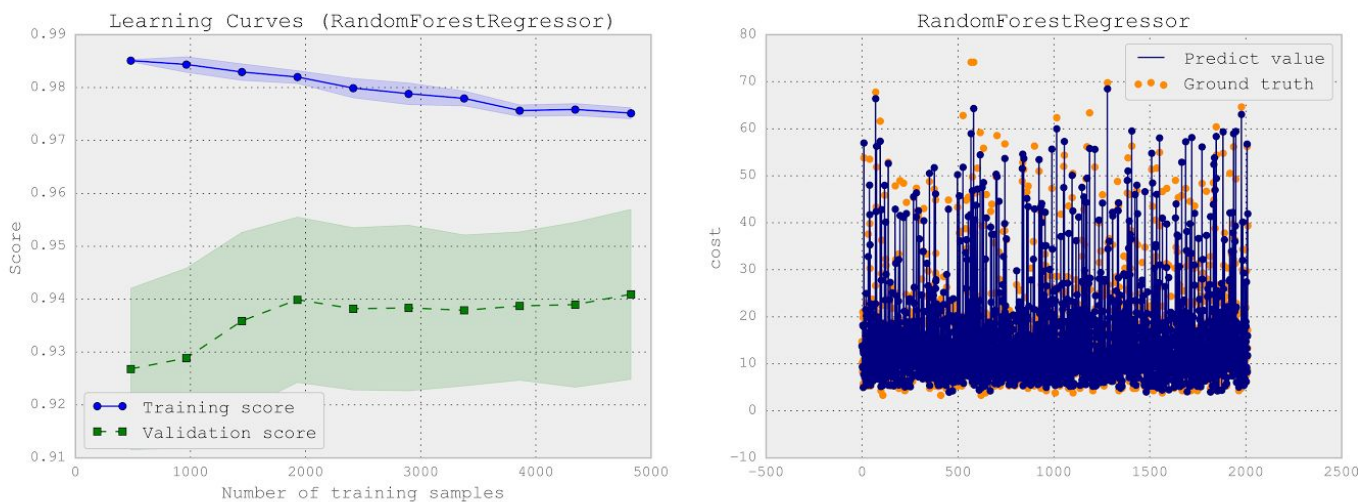
(Figure 3.6: Input data for regression models)

We have applied seven regression models to the dataset and used grid search to select the best hyper-parameters for each model. The score we used for grid search is r square score. Below is the table of all the models we used and the corresponding implementation and r-square score. From the table we know that RandomForestRegressor has the highest r-square score.

Model	Implementation	r ² _score
SVR RBF kernel	SVR(kernel="rbf", C=100, gamma=0.01}	0.762
Gradient Boosting Regression	GradientBoostingRegressor(learning_rate=0.01, n_estimators=900)	0.9524
Kernel Ridge Regression	KernelRidge(kernel="rbf", alpha=0.01, gamma=0.01)	0.4185
DecisionTreeRegressor	DecisionTreeRegressor(max_depth=6)	0.9433
KNeighborsRegressor	KNeighborsRegressor(n_neighbors=4)	0.8165
RandomForestRegressor	RandomForestRegressor(n_estimators=500, max_depth=8)	0.955
AdaBoostRegressor	AdaBoostRegressor(n_estimators=20)	0.9262

(Table 3.2: Best parameters for different models using GridSearchCV)

Below is the learning curve of the RandomForestRegressor model and the graph of our prediction for the trip cost. The graph on the right side shows the predicted values, that is the blue points, and the ground truth, that is the orange points. From the graph we know that RandomForestRegressor model has a good performance on the test set. The left graph is the learning curve of RandomForestRegressor model. By analyzing the learning curve, we could benefit from adding more training examples, reducing the size of features or increasing the regularization parameter to address overfitting.



(Figure 3.7: Predict value and ground truth for trip cost)

(d) iOS Application for Displaying Cost

In order to take advantage of our model in the real world, we have made an iOS app in Swift to display the route between two locations input by a user and the corresponding estimated trip cost based on Google Maps SDK for iOS. Several screenshots of our app will be shown in the next section. In our app, we used HTTP requests to get the route from Google Maps server directly and we used Google Cloud Platform to build our server to respond the request of the trip cost from the app.

Specifically saying, we used socket to establish a TCP connection between our app and the server to get the predicted trip cost. Whenever a user clicks on the button to display cost in our app, the TCP connection will be established, then the latitude and longitude of the two locations will be sent to our server. After the server receives the location information, it will first initiate an HTTP request to the Google Maps server to get the distance and duration of the trip between these two locations. Then it will input the distance, duration and weather into the corresponding model (depends on whether it is weekday or weekend) to predict the trip cost and send the result back to our app. Therefore, we can display the cost in our app. Moreover, we run a crawler on the server to fetch the weather forecast for the next 24 hours from a weather website every hour, which will be used as the input of our model. All the scripts on the server is written in Python.

A screenshot of the log on the server is as follows:

```
connection established
('160.39.38.105', 54274)
client_data: 40.6413111,-73.7781391|40.758895,-73.985131
[ 1.81000000e+01 2.59900000e+03 4.80000000e+01 8.00000000e+00
 0.00000000e+00 0.00000000e+00 1.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00]
/root/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:386: Deprecat
ionWarning: Passing 1d arrays as data is deprecated in 0.17 and will raise ValueError
in 0.19. Reshape your data either using X.reshape(-1, 1) if your data has a single fe
ature or X.reshape(1, -1) if it contains a single sample.
  DeprecationWarning)
43.32
58.85
connection closed
```

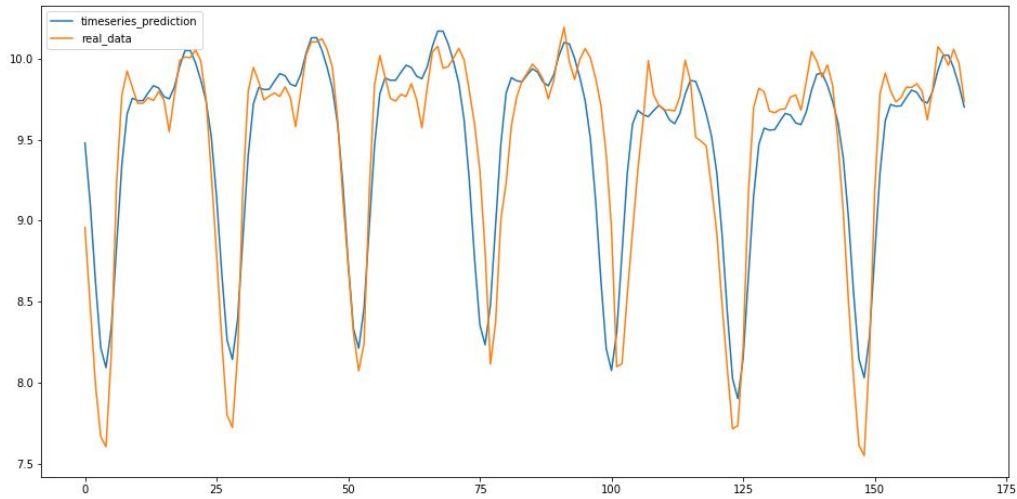
(Figure 3.8: Part of the log on the server)

4. Results

In this part, we will mainly discuss the key findings of our project. Our achievements include making prediction about total pickup number per hour in the Manhattan area using time series method, making prediction about cost per trip and pickup number in every coordinate using cross validation and grid search method, and developing an iOS app which provides route suggestions and corresponding estimated cost.

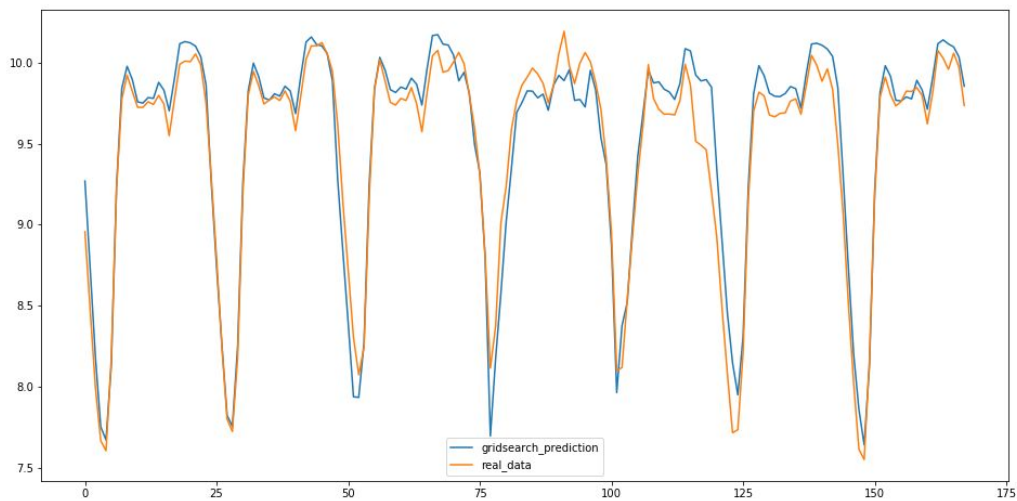
(a) Prediction of Total Pickup Number Hourly

We make prediction of total pickup number per hour in the Manhattan area using both time series and grid search method. They both have a high accuracy. Here is the prediction of the first week in June 2016, which has exactly 168 hours, using time series.



(Figure 4.1: Real data and predicted data using time series of total pickup number in the first week of June 2016)

The figure below is the prediction using grid search combining both the factors of weather and date time.



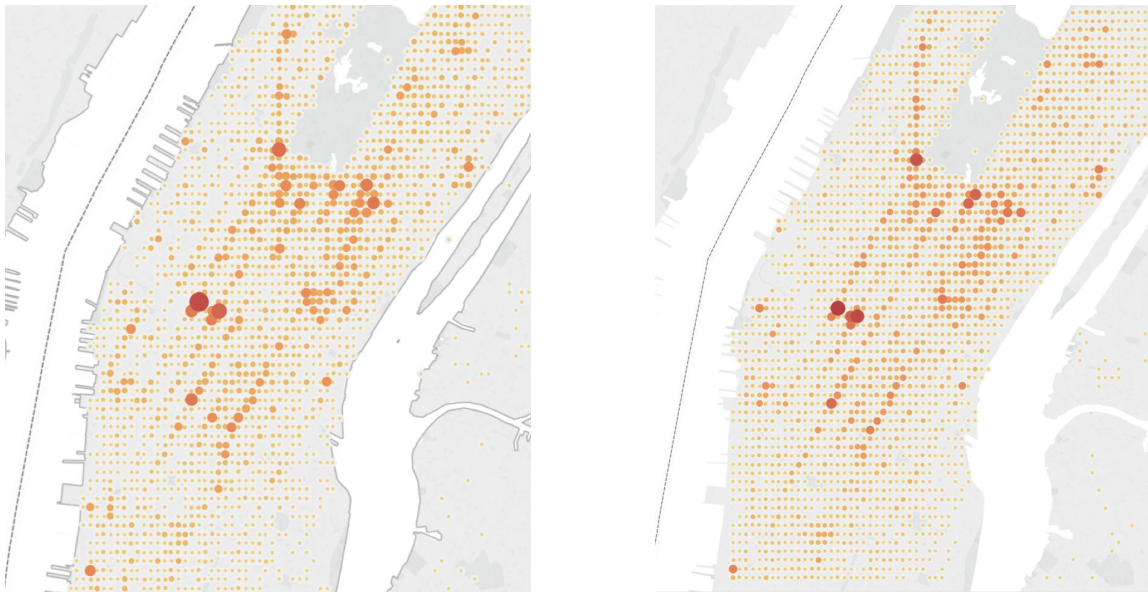
(Figure 4.2: Real data and predicted data using grid search of total pickup number in the first week of June 2016)

As we can see from the results, both predictions are accurate, while the result of grid search is more flexible than the result of time series. Also, the result of grid search is more accurate. It is interesting, because by using grid search, we do not consider seasonality or period. We only separate the weekday data and weekend data, and use “hour” and “weather” as variables. The model would not need to know what day is a specific row of data. But by using time series, the model considers weekly seasonality, holidays, etc. This might mean the factor of weather plays a more important role in the pickup number.

Although the grid search method is relatively more accurate and flexible, performing time series is much faster and requires smaller number of data. And Prophet model is fast because it fits models in Stan.

(b) Prediction of Pickup Number Hourly on Specific Coordinate

We also use grid search to estimate the hourly pickup number on every specific coordinate.



(Figure 4.3: Predicted (left) vs. real (right) pickup number distributions at 6pm on 5.3.2016)

The result turns out to have low accuracy for most of the coordinates. However, for popular places, such as Time Square, Madison Square, etc., the estimations have higher accuracies. Also, the relative pickup numbers for different places per hour could still be observed on map. It means that during certain time we could have a rough idea about which places need more taxis.

(c) Prediction of Cost for Each Trip

Still using the method of grid search, we make predictions about each trip. Here, we use duration, distance, and weather as variables. To make our estimation more accurate, we use the duration returned by Google Map SDK. The cost we predicted is for traditional taxi service, to make our company more competitive, we could reduce our cost by one dollar or two. As seen in the Figure 3.7 (right), our predicted values and the true values are close.

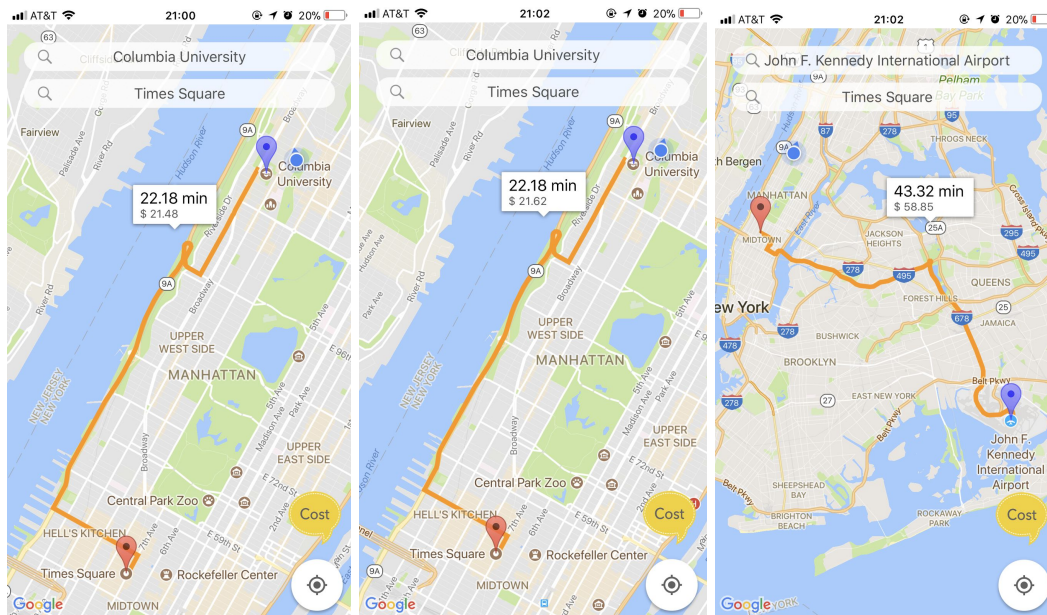
(d) iOS Application

For the convenience of customers, we also develop an iOS app to give users an estimation of duration and cost of each trip.



(Figure 4.4. Icon of our app)

Here are some screenshots of our app. We let users choose their origin and destination, and by tapping on “Cost”, an orange route will display on the screen, which is the route returned by Google Map SDK for iOS. The duration shown in the information window is also returned by Google. However, this value will not change according to real-time traffic condition, which might be improved in the future. But as we can see on the screenshots, the costs at 21:00 and at 21:02 are different. This is because the weather information we crawl on the web changes within the two minutes.



(Figure 4.5. Screenshots of iOS app)

Interesting Finding

Extreme Weather

By observation, during the year from June 2015 to May 2017 in Fig. 3.1, we find that there are some large outliers on Jan. 23, 2016. The pickup number was only 8 at 8pm on that day! It turns out there was a historic blizzard and the New York City was buried in snow.

5. Conclusion:

In our project, we are to solve the yellow taxi pickup inefficiency problem (un-matching demand and supply across time and locations) in New York city area. We used predictive technologies to help provide greater convenience for people to travel, and improve our competency against competitors like Uber, Lyft, Juno, and Via, etc, in the rideshare and taxi transportation market.

The technologies that we used include Time Series method and Machine Learning data science. We have gathered NYC Taxi and weather historical data and used these models to predict pickup number and cost. Among machine learning models, Kernel Ridge Regression and Random Forest Regression were chosen based on our grid search r-square score result. The results of these prediction turn out to be positive. Although for specific locations, the pickup number by hour is not accurate enough, we think that it might due to insufficient amount of training data for each coordinate, and it can be improved by incorporating more data in future model training.

Using the prediction plans that we have, we therefore can schedule the fleet by time, locations, and number, accordingly, as well as provide recommendations for taxi drivers during the day. Additionally, on the customer side, we have also built an iOS app where users can check route and estimated trip price, in such a way that they can have basic information as our competitors can provide as well, rather than going on a “blind trip”.

We believe that our technology can bring convenience to people and improve efficiency to traffic planning, and we look forward to seeing it in the future.