STOR 565 Group 10 Final Project Report

A Study on Manhattan Taxi Trip Duration

Bei An, Yuqing Tang, Han Guo, Courtney Robinson, Tian Xin, Chi Zhang

I. Project Overview

In our project, we use *Multilinear Regression (MLR)*, *Random Forest (RF),* and *Deep Learning (DL)* models to predict the ride durations of taxi trips in the Manhattan area in 2016. The *shortest distance* is calculated based on pick-up and drop-off locations. These are regarded as primary variables, and additional variables are added through merging supplementary datasets. We give a brief explanation of the mechanisms of *MLR and RF* and a thorough elaboration of the methodology of the *DL* model. We evaluate the performance and limitation of each model and then illustrate methods that can potentially improve each model's prediction accuracies.

For **data wrangling**, we first use *Manhattan distance* to approximate the shortest distances of each trip. Then we use *k-means clustering* to generate 250 clusters out of 11,146 roads in Manhattan and summarize the information associated with each cluster including basic road attributes and frequencies of events. We also use Tableau to **visualize the distributions of the trips** based on both pick-up and drop-off locations over a 24-hour period.

II. Data Wrangling

Before constructing models to predict the duration of taxi trips in Manhattan district, we need to create several critical variables based on our existing data for better model performance. They are as follows:

1. *Shortest distance* of the trip based on pick-up and drop-off geo-coordinates.
2. Basic *attributes* (i.e. speed limit, width) of the routes of the trip.
3. *Frequencies* of events that are considered to affect trip duration in the general vicinity, such as accidents, constructions and etc.

1) Shortest Distance Calculation

Note. For the simplicity of this project, we assume that each driver will follow the shortest path between the pick-up and the drop-off locations as well as the general rules, so there will be no detour or shortcut for each trip.

We use both *Euclidean distance* and *Manhattan distance* to calculate the shortest distance, and compare them to the actual distance obtained from Google map. Given the size of this dataset, we randomly choose a sample of 10 for both calculation and comparison.

According to the result, the mean squared difference between euclidean and actual distances is 1.829843910^{7} (m), while the mean squared difference between Manhattan and actual distances is 9.810777510^{6} (m). The difference between Manhattan and actual distances is much smaller than that between euclidean and actual distances, which means Manhattan distance is a much better approximation of the actual distance.

Through researching, we find out that *Manhattan distance*, which only allows traversing in a grid based on a strictly horizontal and/or vertical path, is exactly based on the grid-like street geography of the Manhattan district.

2) Route Attributes & Frequencies of Events

After cleaning up both the **manhat_road** and **manhat_event** datasets (see *data_cleaning* for more details), we first generate the middle points of each road using starting_point and ending_point geo-coordinates. We then create a *distance matrix* that contains the distances between the location of each event and all the roads, and we assign the event to a specific road based on the *minimum distance* (for more details, see **data_wrangling**).

After merging the road and event dataset, we use k-means clustering to cluster all the roads into 100, 250 and 500 clusters respectively.

Based on the graphs, we decide that *k = 250* gives the best clustering result and we then use python to generate a table that contains the 250 clusters along with their summarized information (see **Group_10_Data_Sample_2.csv**). And then we use Tableau to create an interactive map of the 250 clusters.

(see **Tableau_Visualization**)

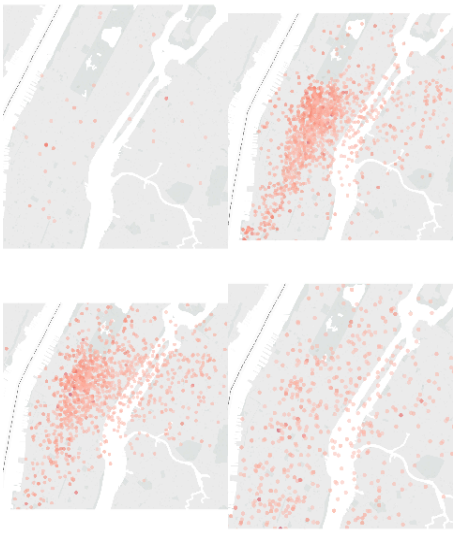
III. Visualization

To discover the patterns in the distribution of taxi trips in Manhattan, we plot each trip based on its **pick-up and drop-off locations** over a 24-hour period using Tableau and obtain the following graphs:

In the graphs on the left, each point represents a single trip. The darker the color of a point is, the longer the duration of that specific trip. At first glance we discover no clear patterns in the distributions of pick-up locations throughout the day, except that there are more pick-ups during working hours and less in the morning and late nights. When we look closer, however, we see that there are more dark points in both the second and third graphs (*9 a.m. & 4 p.m.*), which suggests more trips have longer duration during those hours.

Note. The graphs above correspond to the distribution at *6 a.m., 9 a.m., 4 p.m. and 9 p.m.* (left to right).

Next, we look at the graphs based on drop-off locations:



Note. The graphs above correspond to the distribution at 6 a.m., 9 a.m., 3 p.m. and 9 p.m. (left to right).

With drop-off locations, we see a clear pattern of clusters appearing in the business side of Manhattan in the early hours of the day. As time goes on we see the clusters spread out and disperse to surrounding districts of Manhattan during later hours. Moreover, there are also more darker points in the graph of *9 a.m.* and *3 p.m.*. The similar patterns in both pick-up and drop-off visualizations suggest that there is a correlation between the trip duration and both pick-up and drop-off hours. During rush hours, such as *9 a.m.* and *3 p.m.*, the duration is longer due to both increasing traffic and travelling distance (from surrounding districts to Manhattan).

IV. Model Construction

1) Multilinear Regression

The multilinear regression is one of the most direct and common models to perform regression on a dataset. Given that our dataset has dimension of *1,327,981x25*, the issue of dimensionality is not significant in this case. Therefore, we will construct a basic MLR model without performing variable selections (i.e. *ridge or lasso*).

```
# sampling
size = dim(trip)[1]
set.seed(3)
test.row = sort(sample(1:size, size/4))
test = trip[test.row, ]
train = trip[-test.row, ]
```

For different testing purposes, we generate three MLR models. The first model is the full MLR model that contains all the variables in the dataset, including the shortest distance and the additional road information variables.

```
mod1 = lm(trip_duration ~ month + pickup_hour +
  st_width + st_length + avg_accident_no +
  avg_speed_lim + avg_construction_no +
  avg_event_no + avg_congestion_no + avg_delays_no +
  avg_vehicle_breakdown_no + avg_fire_no +
  avg_signal_problem_no + avg_weather_incident_no +
  shortest_distance + avg_temp + pickup_longitude +
  pickup_latitude + dropoff_longitude +
  dropoff_latitude, train)
```

Considering that the shortest distance is most likely to have the greatest impact on the trip duration, we run the second model with shortest distance as the only predictor in order to compare it with the full model and measure its explanatory and predictive powers.

```
mod2 = lm(trip_duration ~ shortest_distance,
   train)
```

The third model uses only the road information variables, excluding the shortest_distance variable, to evaluate the explanatory and predictive powers of the rest of the predictors, whose influences on prediction might be overpowered by shortest distance.

```
mod3 = lm(trip_duration ~ month + pickup_hour +
   st_width + st_length + avg_accident_no +
   avg_speed_lim + avg_construction_no +
   avg_event_no + avg_congestion_no + avg_delays_no +
   avg_vehicle_breakdown_no + avg_fire_no +
   avg_signal_problem_no + avg_weather_incident_no +
   avg_temp + pickup_longitude + pickup_latitude +
   dropoff_longitude + dropoff_latitude,
   train)
```

The MSEs of the three models are **37.08**, **38.02** and **65.87 (min)** respectively. According to the adjusted R^2s, the full model, that includes shortest_distance and all other predictors, explains **50.1%** of the variation in trip duration; while the second model, that includes only the shortest_distance variable accounts for **48.9%** of the variation in duration; and lastly, the model that excludes the shortest_distance variable only explains **11.6%** of the variation. These results confirm our previous expectation that shortest distance is the primary variable with the largest explanatory and predictive powers, and the additional variables still mildly yet significantly influence the model performance and improve our prediction accuracy. Nevertheless, the MLR model performs rather poorly in terms of predictions in that it is only able to lower the test error to **37.08 (min)**, which is too large given that most of our trips take place in the Manhattan district.

2) Random Forest

The second model candidate is the tree regression. As we aim to accurately predict the trip duration rather than studying the correlation and interactions between variables of interest and the trip duration, random forest is sufficient to fulfill this purpose. Due to the complexity of the random forest algorithm, we use an implementation of RF called Ranger, which is computationally faster and memory-efficient. Moreover, to speed up the computation time, we adopt parallel execution through an R package called foreach. (Note: For the convenience of model comparison, we use the same training and testing dataset as the MLR model.)

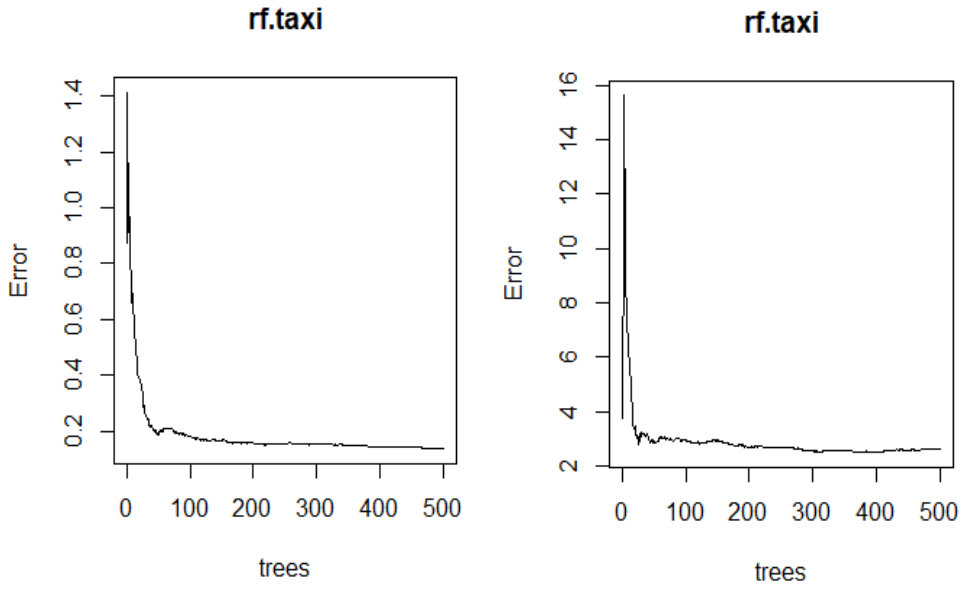
First of all, we train the dataset using the ranger function with the default ntree=500 and mtry = (total number of variables/3).

```
# out of bag error v.s. test error
oob.err <- double(18)
test.err <- double(18)
for(mtry in 4:21)
{
 rf<-ranger(trip_duration ~.-dropoff_hour-id, data = train, mtry= mtry, save.memory=TRUE,write.forest=FALSE, i
mportance = "none",verbose = FALSE)

 oob.err[mtry] = rf$prediction.error #Error of all Trees fitted
 ed<-predict(rf.taxi, test) #Predictions on Test Set for each Tree
 temp<-with(test,(trip_duration-pred$predictions)^2)
 temp<-temp[!is.na(temp)]
```

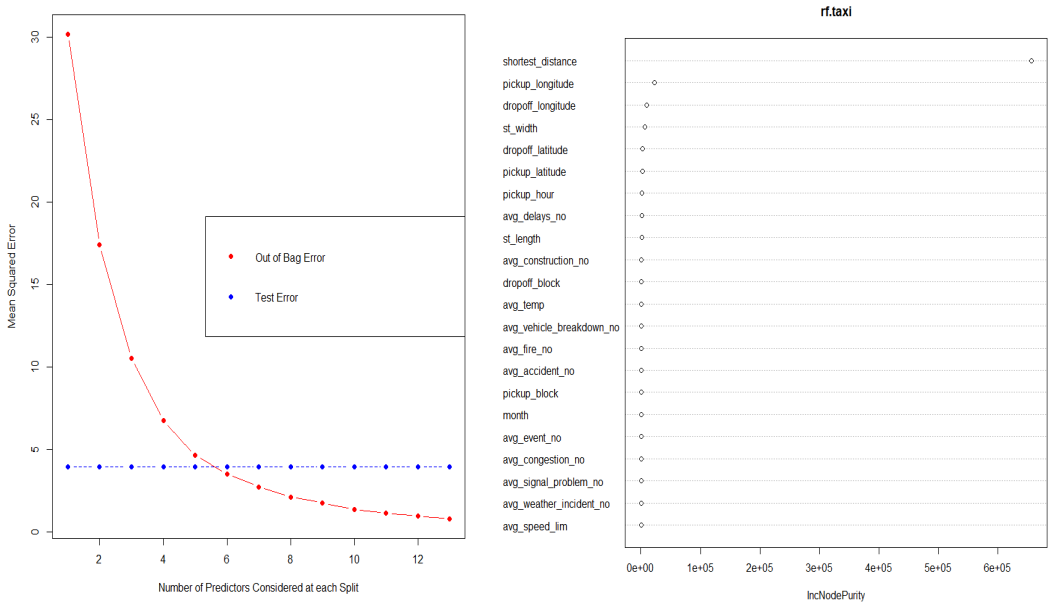
```
 test.err[mtry]<-with(test,mean(temp)) #Mean Squared Test Error

 cat(mtry," ") #printing the output to the console
}
```



Left: Error plot with default mtry. Right: Error plot with mtry of 21.

By plotting the MSE vs. number of branches, we find that the test errors approach a relatively steady state whenever ntree≥300, and therefore is well justified using ntree=500. To find the best estimate of mtry, we then test the out-of-bag error rates based on different mtry values ranging from 1 to 21 and generate the following plot that shows as the number of predictor increases, the OOB error rate exponentially decreases while the test error remains approximately the same.



Left: Out-of-bag error rate with different mtry. Right: Importance plot.

Looking at the plots, we decide to fit the random forest regression with mtry = 21.

```
rf.taxi <- foreach(ntree=rep(500, 6), .combine=combine, .packages=c('ranger', 'randomForest', 'methods')) %dopar% ranger(trip_duration ~.-dropoff_hour-id, data = train, mtry=21, ntree = 300, importance = "impurity", write.forest=TRUE, verbose = TRUE)

# training error
rf.taxi$prediction.error
plot(rf.taxi)

# test error
yhat<-predict(rf.taxi, test)
sq<-with(test,(trip_duration-yhat$predictions)^2)
sq<-sq[!is.na(sq)]
test.mse <- mean(sq)
test.mse
```

The MSE of the new model is **15.55(mins)**, which is surprisingly low and effectively indicates that RF regression has a much better performance than MLR in our case. Nonetheless, the RF model does agree with the MLR model that all variables are significant while shortest_distance is the dominant variable. The low error rate does not necessarily suggest a high practical value. On one hand, RF regression is inappropriate for forecasting because of its unstable nature. The small MSE might result from the built-in variable selection process. On the other hand, RF regression is extremely time-consuming even when pairing with parallel execution. Compare to the deep learning method in the following section, random forest might not be the most satisfying answer for our problem of interest.

3) Wide and Deep Learning Methodology

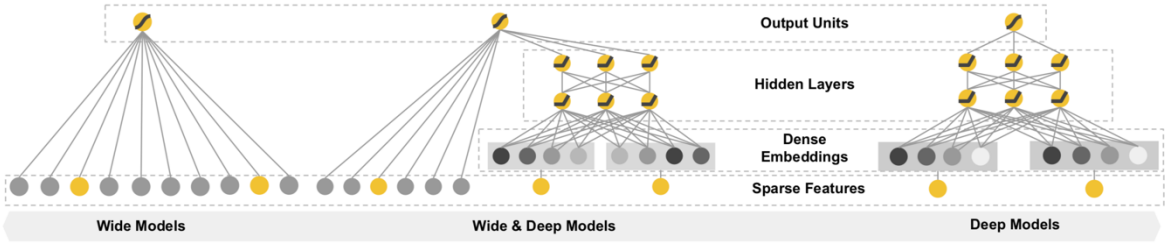


Figure 1 Wide and Deep Model

In this section, we borrow the **Wide and Deep** learning framework from Google Research to achieve both memorization and generalization in one model. Through jointly training a linear model component and a neural network component, we bring together the strength of two models.

1. Wide Model

The wide component is a generalized linear model of the form $y = w^T x + b$, as illustrated on the left of Figure 1 where y is the prediction, $x = \{x_1, x_2, \dots, x_n\}$ is a vector of features, $w = \{w_1, \dots, w_n\}$ are the model parameters, and b is the bias. The feature set include raw input features (in case of numerical inputs) as well as the transformed features (i.e. embedded features for categorical inputs.

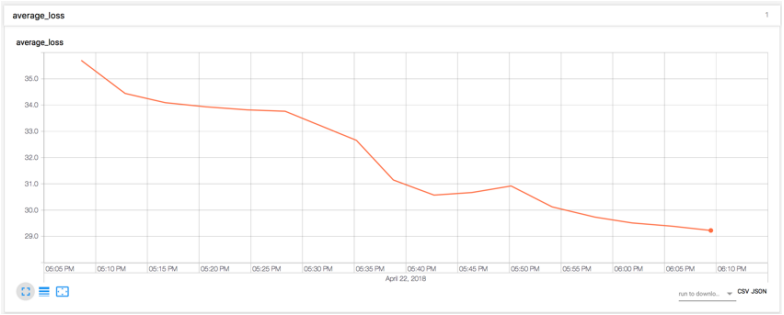
2. Deep Model

For this project, we use the simpler Feed-forward Neural Network (FNN), as shown in the right side of Figure 1. Similar to Wide Model, we convert sparse categorical features into low-dimensional embedding vectors that are jointly optimized with the model. Then all features are fed into the hidden layers of a neural network in the forward pass. Specifically, the FNN does the computation $a^{l+1} = f\,(W^l a^l + b^l)$ where l is the layer number, and f is the activation function, chosen to be $\tanh(\cdot)$.

3. Joint Training and Optimization

The wide and deep components are combined using a weighted sum of their outputs as prediction, which is fed to one common MSE loss function for joint training (i.e. all parameters are jointly optimized). We use two optimization algorithms for different components. Specifically, we use Follow-the-regularized-leader (FTRL) algorithm for the wide component, and Adam, an algorithm for first-order gradient based optimization of stochastic objective functions. We also initially tried AdaGrad, but found this led to a bad local minimum.

Model Construction and Analysis



We use **Tensorflow** for building the model and optimizing the objective. Because of the huge computation and memory cost, we use stochastic gradient descent with mini-batch size 512, and the recommended default learning rates for both optimizers. For the deep model, we use a 4-layer FNN with hidden layer sizes of {256, 128, 64, 32}. The entire model is trained using a single **K40** GPU.

Among all the features, we use numerical features as is, but convert all categorical features into vectors with the hash bucket size 500. Specifically, we use indicator embedding (i.e. one-hot) for the **month** variable and dense embedding to convert all the other categorical variables into R^{16} learned jointly with all parameters. We use the same input features for both the wide and the deep components. In addition, to avoid being bottlenecked by the data, we utilize parallel data reading/preprocessing with 5 parallel calls. After roughly 1 hour of training, the test MSE error is **29.22 (mins)**. From Figure 2 above, we can see that the MSE is decreasing very fast initially. This is due to the use of Adam optimizer, which leverages the history of loss values for automatically adapting its effective learning rates. However, we find the model to be stuck in a sub-optimal minimum, achieving performance only better than regression methods, but worse than tree based methods. We think the reasons for that are: 1) Adam optimizer often works well when the dimensionality of inputs is very large (which is not in this dataset), and is known for being hard to converge around optimum. 2) We didn't explicitly measure the interaction between different variables, which can be a big feature for this dataset. 3) The complexity of neural network makes it hard to interpret the results and make improvements, thus diagnosis is hard. 4) The model is not yet fully converged, and further training can still help.

V. Limitations & Conclusions

Although our models perform fairly well and we are able to obtain a test error as low as *15.7* (mins) using the *random forest model*, there are still several limitations that prevent us from obtaining more accurate predictions:

1. **Shortest Distance Assumption:**
   i) We assume that there is *no detour or shortcut* for each trip, which is unlikely in reality.
   ii) The *Manhattan distance approximation* still falls short of the actual distance. Given that shortest distance is a primary variable in our models, the prediction accuracies are affected by the difference.
2. **Clustering Limitation:**
   i) When adding the additional information from supplemental datasets, we only consider pick-up & drop-off blocks but ignore all the blocks in-between, which is why the additional predictors are statistically significant but contribute little to improve prediction accuracy.
   ii) The additional information is not firsthand which undermines our models' predictive powers for the other data.
3. **Modeling Process:**
   i) We only test the models using the 2016 dataset. This is problematic as the models might be applicable to this specific case but do not bear generalizing significance.
   ii) The random forest model is highly flexible and depends largely on the initial data. It might also be applicable to this case but unable to be implemented for future forecasting.
   iii) For the deep learning model, it is hard to to interpret the model decision. And given that we have an extremely nonlinear optimization, it is easy to be trapped in a local optimum, which explains its relatively poor performance.

In Conclusion, both the *MLR* and *Random Forest* models find shortest distance to be the most important variable. Although the additional variables are statistically significant, they comparatively contribute much less due to our clustering limitations. The *RF* model performs the best in terms of prediction with the **least test error of only 15 minutes**. Even though our *Deep Learning* model's performance falls short of our previous expectation, it is probably more suited to future forecasting than the *RF* model, which is highly flexible. If we have more time to train the *DL* model, we believe we will come up with a much better prediction accuracy.

References:

Manhattan distance. [accessed 2018 Apr 26]. "*https://en.wiktionary.org/wiki/Manhattan_distance*"

Cutler A, Breiman L. Random Forests. [accessed 2018 Apr 27].
"*https://www.stat.berkeley.edu/~breiman/RandomForests/cc_papers.htm*"

foreach: Provides Foreach Looping Construct for R. R-project. [accessed 2018 Apr 25].
"https://cran.r-project.org/web/packages/foreach/index.html"

R: parallel computing in 5 minutes (with foreach and doParallel). 2015 [accessed 2018 Apr 27].
"http://blog.aicry.com/r-parallel-computing-in-5-minutes/"

Wright M. ranger. 2017 [accessed 2018 Apr 25].
"https://www.rdocumentation.org/packages/ranger/versions/0.9.0/topics/ranger"

Cheng, Heng-Tze, et al. "Wide & Deep Learning for Recommender Systems." *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems - DLRS 2016, 2016, doi:10.1145/2988450.2988454.*

Kingma, et al. "Adam: A Method for Stochastic Optimization." *[1412.6980] Adam: A Method for Stochastic Optimization, 30 Jan. 2017, arxiv.org/abs/1412.6980.*