Bike Network Flow Prediction

Group5: Yue Feng, Luoyao Chen, Hang Yang,You Wang, Cheng Han

1.Introduction & Background

Bike-sharing programs are gaining a lot of traction around the world. There are lots of benefits from bike-sharing programs, such as environmental quality. One of the largest bike-sharing programs in the United States is Citibike which was founded in 2013. Based on the current data from the Citibike website, over 1 million riders use Citibike for business or pleasure every month. Meanwhile, there were 332 stations and 6000 bikes since its opening in 2013, and nowadays, the system has grown to more than 1000 stations and 17000 bikes by 2020 (Stanislav).

As a result, the demand for Citibike goes up dramatically. During high usage hours, the bike distribution throughout the city might become imbalanced, for example, some stations are totally full and others are completely vacant, rendering those stations worthless for future trips.In this approach, unequal distribution of bikes reduces the overall system's availability and reliability.

2. Problem Statement

2.1 Data set

The data set we used in our project is the Citi bike System Data provided by Lyft Bikes and Scooters, LLC (Index of bucket "tripdata"). The data contains information about Citi bikes from May 2013 to April 2022. For each month, the file contains information such as RideID, Rideable type, Started at, Ended at, Start station name, Start station ID, End station name, End station ID, Start latitude, Start longitude, End latitude, End Longitude, Member or casual ride.

In this project, we analyzed the files from 2021 January to 2021 December. We used Started at, Ended at, Start latitude, Start station ID, End station ID for building model and Start latitude, Start longitude, End latitude, End Longitude for clustering. Other information can be analyzed at a deeper level to answer other questions, but for now, we did not consider it in our project.

2.2 Definition of Imbalanceness

Given one specific station, the number of bikes entering the station minus the number of bikes leaving in a given period (2 hours in our project).

2.3 Goal

Our goal was to predict the 'imbalanceness' of stations selected by clustering. Based on these predictions, the system operators can determine the best plans to distribute the bikes for these stations at different times.

2.4 Optimization summary

We demonstrated our understanding of Advanced Python Concepts by including the following techniques:

1. Utilized List comprehension, NumPy array calculation, Counter module, cuda and built-in functions to accelerate computing.
2. Used the Time module for code profiling.
3. Taken advantage of Multithreading for downloading the dataset
4. Applied Multiprocessing for data cleaning and data transformation
5. Leveraged nn.Parallel to parallely run models on multiple GPUs

3. Data Preprocessing

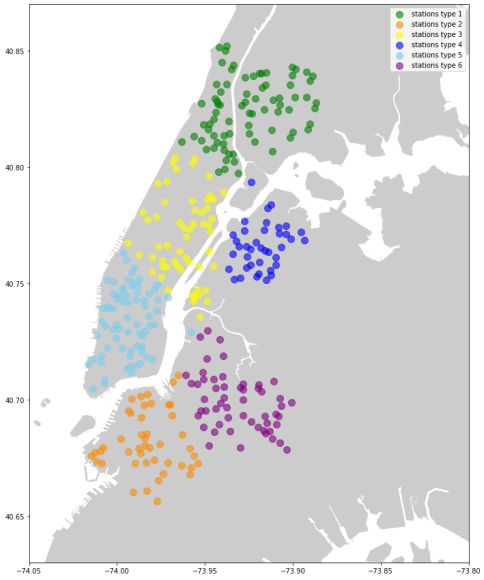
3.1 Download and data cleaning

We first use the requests package in Python to download the data from the Citi Bike trip data website. Once we have

the data we need, we begin to perform data cleaning. The first step of data cleaning is to filter out the records with null value. Then, we extract the date and hour of the trips and filter out the records with a starting hour larger than the end hour on the same day, since these records are obviously wrong and may be resulted from system error. In addition, to simplify our problem, we also filter out the trips with different start dates and end dates. We observed that some of the dates only have trip records in specific hours of the day, which may indicate certain system malfunction. So, we also delete the dates without 24-hour records.

3.2 Clustering

After first-step cleaning, we have trip data come from 342 Citi bike stations. In order to simplify our problem and reduce the computational cost for future analysis, we use the K-means algorithm to cluster the station to 6 groups and later study the interactions between these 6 groups. The clustering (show as below) is performed based on the geographical location of the bike stations.



3.3 Data transformation

We assign each station to a group according to the result of clustering. Having observed that most of the trips have a duration less than 2 hours, we separate the data under every date into 12 two-hour chunks. Then for every two-hour chunk, we create a 6*6 matrix, each entry of this matrix records the number of trips from one specific station to another. For example, the (i,j) entry of this matrix represents the number of trips from cluster i to cluster j.

3.4 Optimization for data preprocessing

In our problem, we are dealing with 11 large csv files and the operations we perform on these files are very similar. Therefore, we can use apply the multithreading and multiprocessing methods during data preprocessing and benefit from the concurrency and parallelism

3.4.1 Multithreading for downloading dataset

When we try to download the 11 csv files, instead of downloading them with a serial function, we implement the threading method with the ThreadPoolExecutor in concurrent.futures. We use threading with 16 threads. This method improved the speed by more than 50% compared with the serial version. Threading method works in this case because this is an I/O bound problem and the majority of the time is spent waiting for the network. The processor can switch between the threads whenever one of them is ready to do some work. So, threading yields better performance in this task.

3.4.2 Multiprocessing for data cleaning and data transformation

In the step of data cleaning and data transformation, the tasks for each file are the same. Unlike downloading data, data cleaning and transformation are CPU-bound tasks. Therefore, we apply the multiprocessing method (with number of processes N = 16) in this case. After using multiprocessing, the execution time for data cleaning decreased by about 45% and the execution time for data transformation decreased by about 75%. We can see that multiprocessing is working very well on our problem because for CPU-bound tasks and truly parallel execution in Python like our problem, the multiprocessing module is a better option.

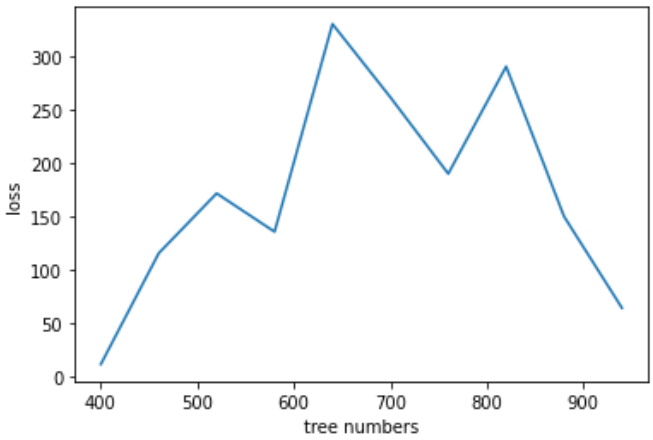
4.Model and Results

4.1 Random Forest Regression Modeling

We first used the random forest regression model to build a baseline model. The model predicted predict the in-node and out-node imbalance (ie. the 'trans_num' variable) using 4 features, 'station1', 'station2', 'day', 'previous_hour'. Meanwhile, in order to improve the training speed, we also employed incremental learning techniques to build the model. We describe the implementation in detail as follows.

In 2011, Wang A P et al proposed an Incremental Extremely Random Forest (IERF) to solve the online learning problem for small sample data streams. Experiments on the UCI dataset demonstrate that the incremental learning algorithm outperformed the greedy decision tree reconstruction algorithm on the mesoscale sample set. Therefore, to improve the training efficiency, we apply the strategy of incrementally growing existing trees as well as adding new trees at the same time for updating random forests.

In the training stage, we first initialized a random forest with 400 estimators, then while literately read in the preprocessed dataset for February to November , we added 60 new trees during each iteration. And the training MSE curve below shows that the MSE loss dropped dramatically after 8 iterations and after finished all 10 iterations, the MSE loss of the model decreased to around 64.83, which indicated the model fitted the data relatively well. After training, we tested the model on the December dataset, and the MSE loss on the test data was around 75.44.



4.2 Neural Network Modeling

The neural network we deployed consisted of 3 parts: GCN, RNN and MLP. We describe the first two in detail as follows.

4.2.1 GCN

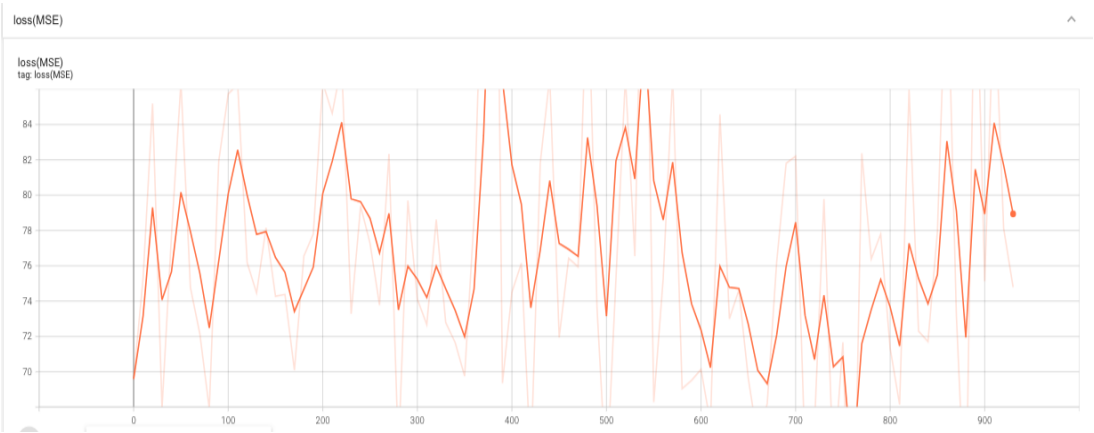
As proposed by Kipf and Welling (2016), Graph Neural Network (GCN) can be used to build a map from graph feature matrix (H) to an output graph with new-generated edges. Specifically, the network structure contains linear layers with matrix multiplications. A primitive version of the formula is $f\ (H,\ A)\ =\ \sigma(A\ H^{l}\ W^{l})$, where $H^{l}{}_{N\times D}$ is a

vertex weight matrix, with each column captures a node feature; $A_{N\times N}$ is an adjacency matrix normalizing the weight of node features, and W^{l} is the weight matrix for the l-th layer in the network. In our specific work, we applied a two-layer GCN, also as we assumed the 6 clusters to be connected with each other, we used random initialization for the adjacency matrix $A_{6\times 6}$

4.2.2 RNN

Since the dataset was formatted into days (333 days in total, 303 days train, 30 days test), with each day containing 12 chunks, we added a RNN model to better capture sequenced information. Some extra manipulations were done in order to construct cluster-wise balanceness. Namely, for the input space $B \times 12 \times 6 \times 6$, we constructed difference between adjacent hours, and obtained $X = B \times 11 \times 6 \times 6$. For the output space, since we tried to predict the cluster imbalanceness, for every 2 hour chunk, we calculated the cluster unbalanceness ($\sum col_i - \sum row_i$), and obtained a vector of size 1×6. Overall, our output space has size $Y = B \times 11 \times 6$. For the RNN part, we mapped $36 -> 36(hidden) -> 6(out)$, and used another MLP as the output layer.

Using the MSE loss, batch size 32, we trained 3000 iterations using the data month 2 - 11, the training MSE curve is shown below, with training average MSE = 78.38563537597656. Testing MSE has an average test MSE = 70.40753936767578.



5.Conclusion

In this project, we explored the structure of the data and used the K-means method to partition Citi bike data into 6 clusters. A new metric 'imbalancedness' was defined and two methods (IREF, RNN) were used to make predictions. The MSE loss of the random forest model on the test set is 75.44. The MSE loss of the neural network model on the test set is 70.40753936767578 . Such accuracy is enough for us to make relatively accurate predictions.

Throughout the project, we mainly used multi-threading and multi-threaded methods to improve the efficiency of data pre-processing. In addition, we also used python modules such as NumPy, time, counter, cuda etc. to help us optimize the speed of our code.

Reference

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Wang, A., Wan, G., Cheng, Z., & Li, S. (2009, November). An incremental extremely random forest classifier for online learning and tracking. In *2009 16th ieee international conference on image processing (icip)* (pp. 1449-1452). IEEE

Rogozhi, S.(2020, December 18). Analysis and prediction of Citi Bike usage in the unpredictable 2020. Towardsdatascience.https://towardsdatascience.com/analysis-and-prediction-of-citi-bike-usage-in-the-unpredictable-2020-3401da26881b