Bike Network Flow Prediction

Yiju Hou
Stanford University
Dept. of Computer Science

Stuart Sy
Stanford University
Dept. of Computer Science

Christopher Yuan
Stanford University
Dept. of Computer Science

Abstract

Recent network analysis research has made remarkable progress on studying complex networks in a social context with machine learning techniques. One example of such a complex network is the system data of the New York Citibike bike sharing program, which consists of the data from 49 million bike trips in New York City since July 2013. By inspecting the dataset, we find out that significantly more demand could be satisfied if we were able to predict the bike flow between stations and rebalance the bike supplies accordingly. In this paper, we formalize this task as predicting the change in number of bikes at each station, and develop algorithms to predict this change based on snapshots of the network from the past. We find that clustering the network into smaller parts results in more accurate predictions, and propose a novel algorithm for rebalancing the network based on the prediction results from our model.

1. Introduction

Bike-sharing systems are becoming incredibly popular all over the world. One of the largest such systems in the United States is the Citibike network in New York City [1]. Every month, over 1 million riders take Citibike trips for work or leisure. During peak usage hours, the distribution of bikes around the city can become unbalanced, with some stations completely filled up and other stations completely empty, leaving those stations unusable for ending/starting a trip in the future. In this way, an unbalanced distribution of bikes leads to lower availability and reliability of the entire system.

In this project, we want to predict the "balancedness" of each station, which is defined as the difference of the number of inbound bikes and outbound bikes for any station in a particular time interval, given snapshots of the entire system for a previous time interval as the input. Based on these predictions, system operators can plan optimal redistributions that will keep the system balanced during peak usage hours. Given the trips took place between time t and $t + d$, the network is defined as $G(V, E)$, and each edge $E(u, v)$ represents an trip between station u and station v. By definition, in a directed graph, a node is balanced if the indegree equals outdegree, and a node is semi-balanced if indegree differs from outdegree by 1 [4]. Therefore, we propose to model this problem as a node-balancing prediction problem over a network. We seek to accurately predict if a node if balanced, semi-balanced or unbalanced from a future time t' to $t' + d'$. We also want to quantify the degree of "unbalancedness" of a node so that we can provide real-time recommendations on how to restore the unbalanced stations back to a balanced or semi-balanced state.

2. Related Work

Before we discuss our work, we will review some related work that will inform our approach.

We have review related research on the problem of predicting missing links from an observed network: a network of interactions is constructed based on observable data and then the algorithm tries to predict additional edges that, do not exist yet, are likely to occur in the future. For example, Nowell et al. provides several node proximity based methods to approach to assign a $score(u, v)$ metric for each pair of unconnected nodes u and v in the graph, which indicates how likely new edges will be formed [5]. The methods are evaluated on two collaboration networks between authors, and the researchers found that several of the methods significantly outperform the random predictor on certain networks, but there was no one method which was uniformly better across all networks.

Additionally, the link prediction problem is relevant to current applications of machine learning algorithms. The research conducted by M.Hasan et al applies and evaluates 8 classification algorithms, including decision tree, SVM(linear kernel), SVM(RBF kernel), k-nearest neighbors, multilayer perceptron, RBF Network, Naive Bayes and bagging, on the task of predicting links co-authorship networks [6]. The researchers analyze the algorithm performance and conclude that SVM outperforms other models with narrow margin in all performance measures.

As we will discuss in the Methods section, high dimensionality will constrain much of the analysis that we do. In

order to reduce the dimensionality of our model, we can segment the network into several clusters, or communities. Girvan and Newman propose an approach to clustering that operates over edges instead of nodes [2]. The research formulates a new measure called edge betweenness centrality, which they define as the number of shortest paths between pairs of vertices that run along it. The edge betweenness measure is intuitive and produces good results, as the authors show by analyzing several graphs with both known and unknown community structures. However, the relatively slow runtime of the algorithm is a large roadblock to analyzing large networks. The entire algorithm runs in time $O(m^2n)$, where m is the number of edges and n is the number of nodes in the network.

3. Methods

In this paper, our ultimate goal is to make predictions on the difference of indegree and outdegree of every node in a given time interval so that we can make real-time recommendations to rebalance the bike stations in the system. If we can show that our predictions are accurate, then the Citibike operators can use them to redistribute bikes accordingly. In this section, we will describe several ways we attempted to solve this prediction problem.

3.1. Supervised Learning: Trip Prediction

The most intuitive way to approach a network flow prediction problem is to predict individual future trips on the network. Given a graph that represents all trips taken in the network in a particular time interval, we could output a new graph representing every trip that we predict will happen in the next time interval.

This edge prediction problem naturally suits itself to a supervised machine learning framework. Given the input in the form of an adjacency matrix for the bikeshare network between time t and $t+k$ and labels in form of an adjacency matrix for the network between time $t+k$ and $t+k+1$, we could train a model to learn the statistical correlation between past and future. We considered two models: a linear regression and Recurrent Neural Network.

3.2. High Dimensionality Requires More Data

High dimensionality (which directly leads to a large number of parameters in the models) is inherent in edge prediction problem that we have formulated. With the number of stations in the network (804) and the architecture that we proposed for edge prediction, each of our training examples would have a feature matrix between size of $(804 \times 804 \times 1)$. This dimensionality will require at least $600,000$ parameters even for the simplest single-layer neural network.

Machine learning theory suggests in order to train a hypothesis class that has d parameters, generally we need to use on the order of a linear number of training examples in d [8]. Meanwhile, there are just over $35,000$ training examples in our dataset, one for each hour that the network has operated between 2014 and the present day. We most likely will need more data in order to meaningfully train our machine learning models.

Therefore, we will propose a modeling formulation and two new input dimensionality reduction techniques.

3.2.1 Reformulate edge prediction to node-balance prediction

Given the adjacency matrix for a current time period in the Citibike network, the edge prediction model tries to output a predicted adjacency matrix for the next time period. In order to achieve the same goal with reduced dimensionality, we propose to predict the balancedness of a given node, which represents the change in the number of bikes at each station. More formally, this is

$$\Delta B_i = \text{indegree of } S_i - \text{outdegree of } S_i$$

where S_i is a station, and the equation is parameterized by time. Given the adjacency matrix A, the Δ of the given station s can be calculated as $\Delta_s = \sum_{i=0}^{n} A_{si} - \sum_{i=0}^{n} A_{is}$. With this transformation, our model can still predict which bike stations are more likely to have overflow or underflow problems.

3.2.2 Input Dimensionality Reduction

We hypothesized that the Citibike network could be partitioned into several clusters, with each cluster representing a physical neighborhood or a ridership community in New York City. These clusters would have a relatively high number of trips within the cluster, and fewer trips between clusters. If we can validate this hypothesis, then the system-wide bike rebalancing problem could be decomposed as several smaller rebalancing subproblems, one within each of the clusters.

Next, we will talk about several ways to do this partitioning and examine the quality of the partition results.

3.3. Graph Partitioning

If we compose all trips for a given time period into one multi-edge graph, then we can start partitioning the bike stations into several communities. If we are able to partition the graph into subgraphs with similar sizes, we can use a divide and conquer strategy for our node balancing problem in each subgraph. Additionally, these communities can provide insights into the structure of the data and provide more useful features in our flow prediction task.

Girvan Newman (Node Clustering):
The first community-detection algorithm we evaluated

was the Girvan Newman algorithm. Girvan Newman community detection operates on the edge-betweenness centrality metric [2]. The authors calculate a hierarchy of communities over the network by repeatedly removing the edge with highest edge betweenness and re-running the edge betweenness algorithm. The order in which edges are removed creates the hierarchy.

Unfortunately, the Girvan Newman community detection algorithm did not give good results. The majority of our stations ended up in a single community. This may be because the graph is highly connected, with any station almost certainly connected to any other station. Because there is one large community, it was not possible for us to partition the graph in a meaningful way.

Spectral Clustering

Second, we tried spectral clustering, a more advanced graph partitioning technique. Normalized spectral clustering algorithm has demonstrated outstanding performance at partitioning graphs with complex structures [9]. In order to partition the graph with the normalized spectral clustering algorithm, we applied a technique proposed by M. Newman to convert the multi-edge graph into a graph with weighted edges by using the count of edges between any pair of nodes as the weight [7]. We applied the algorithm to partition the graph into 0, 2, 4, 6, 8, 10 clusters (0 implies no partitioning). With each partition configuration, we are going to train machine learning algorithms independently on each cluster and evaluate the overall performance by taking a weighted average. We later decide the most optimal graph partitioning by evaluating the quality of the partitioning and the performance on the prediction task.

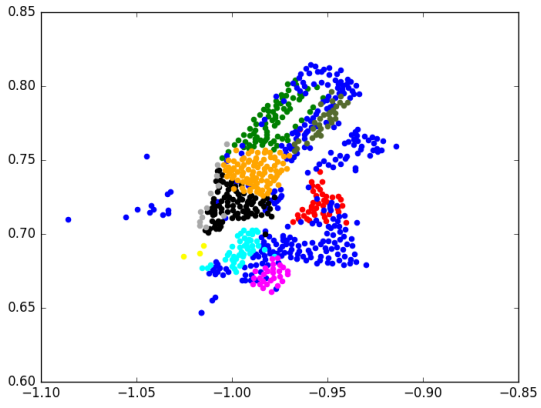

Figure 1: Spectral Clustering with $k = 10$.

By examining the partitioning result, we can clearly see the shapes of some geological neighborhoods, such as Lower Manhattan, Midtown, Brooklyn, and Hudson River bank. The result validates our previous assumption that the ridership clusters are similar to the geographical neighborhoods.

K-means Clustering:

In order to further verify this hypothesis, we ran k-means clustering based on the geographic distance of the bike stations and compared the clusters generated by k-means with the ones generated by spectral clustering (which takes edges/weights into account). The k-means algorithm alternates between the following two steps until converge:

Step 1: For each point $i = 1, ..., n$, assign i to cluster in closest centroid:

$$z_i \leftarrow arg \min_{k=1,...,K} ||x_i - u_k||$$

Step 2: For each cluster $k = 1, ..., K$: Set u_k to be the average of the points assigned to cluster k:

$$u_k \leftarrow \frac{1}{|\{i:z_i=k\}|} \sum_{i:z_i=k} x_i$$

Here in Figure 2, we can see the results of k-means clustering on the bike stations' geographic data with $k = 10$.

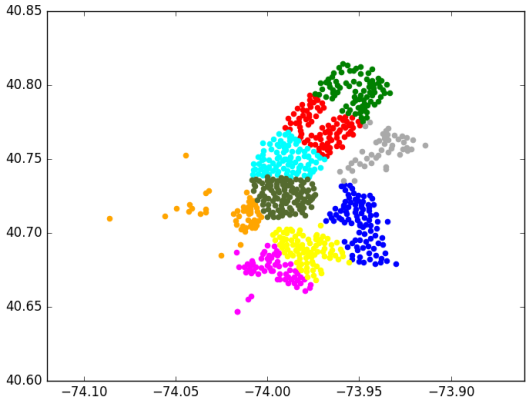

Figure 2: K-means Clustering with $k = 10$.

By comparing the K-means Clustering graph in figure 2 with the Spectral Clustering graph in figure 1, we can see that the clusters generally correspond to each other – even though the K-means clusters were generated with static station location data while the spectral clusters were generated with trip data. The fact that the clusters are similar means that our assumption that people travel within geographic neighborhoods is validated.

3.4. Solving Prediction Problem within Clusters

Our clustering analysis led us to conclude that the nodes on the Citibike network could be partitioned into several

clusters. Each cluster represents an approximate geographic neighborhood where there are relatively more trips within the neighborhood than there are trips between different neighborhoods.

Because the spectral clustering partitions take trip history into account and also represent geographic features, we decided to split the graph into several sub-graphs using the normalized spectral clustering algorithm. Then we removed the edges that do not belong to the same cluster. Because of the nature of spectral clustering, the sub-graphs do not have any overlap in nodes or edges, so we would then be able to independently run the rebalancing algorithm on each sub-graph. By doing this, we could dramatically reduce the dimensionality of our input data and the quantity of computation while preserving the accuracy of our model.

After training the node-balance prediction model on each of the sub-graphs, we take a weighted average (based on the number of trips) of the error metrics for each of the sub-graphs to obtain the overall prediction error over the entire graph. We believe that this reformulation correctly captures the dynamics of the real-world bike rebalancing problem. Since each cluster is approximately mapped to a geological neighborhood, then each neighborhood can be efficiently rebalanced locally and independently of the other neighborhoods.

By treating each cluster as a sub-graph, we leave out the edges that originally spanned nodes that are now in different clusters. In order to build an accurate and realistic model, we want to preserve as many trips as possible. Therefore the portion of trips that are lost is used as an important metric in the evaluation section.

3.5. Evaluation

We plan to train and evaluate our models based on different graph partitioning. To evaluate the performance of the different models and graph partitioning, we will use a mean squared error on the difference between the predicted number of edges between each station, and the actual number of trips that were taken.

$$L = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

4. Dataset

4.1. Description

We use the aforementioned Citibike data [1] detailing the movement of shared bikes in New York City from the period of July 2013 to September 2017. Each row of data represents a single bike trip taken, with attributes such as the timestamp, start/end station name and location, and user age and gender. There are approximately 1 million bike trips recorded each month, with the usage of the bike share system increasing over time, as seen in Figure 3. We can see that the usage of the bike share system generally increased over time, but that there are seasonal dips in the winter months. There are a total of 804 unique bike stations represented in the dataset, though not all are used in each month.

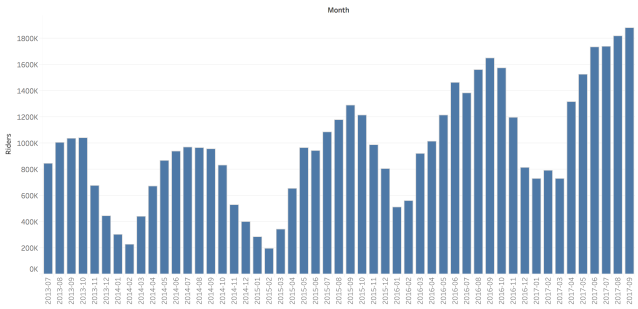

Figure 3: Bike share usage over time.

4.2. Network Representation

We represent this dataset as a multigraph network where each bike station is a node, and each bike trip is a directed edge from one station to another. Figure 4 is a visualization of these different bike stations spread throughout New York City. The stations cover most of Manhattan, and also spread through the Brooklyn borough.

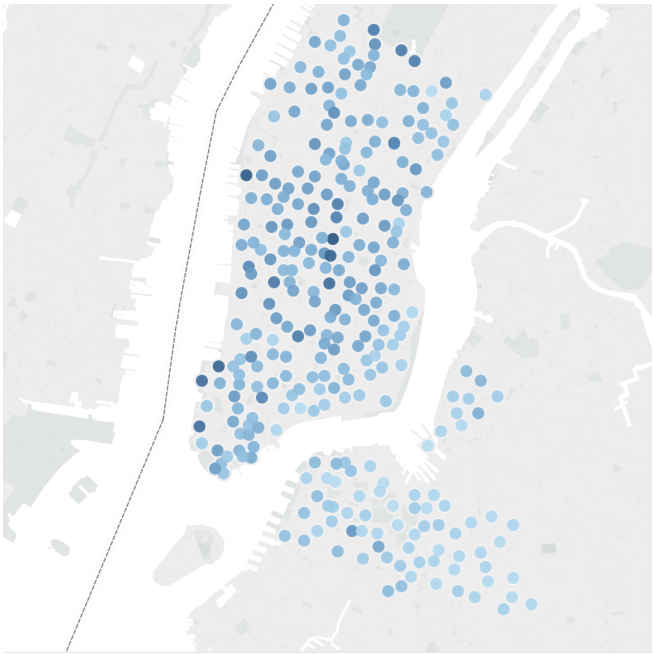

Figure 4: Visualization of bike share network.

4.3. Preprocessing

Our initial feature set only consists of the start time, end time, start station and end station of a trip. We first filtered the data to extract those fields. We then aggregated all trips taken on the network into one-hour chunks. The aggregated trip data for each hour is represented as a $n \times n$ adjacency matrix, where n is the number of stations in the network. Each entry in the matrix corresponds to the number of trips taken from station A to station B in any given hour. For any station S_i, if we sum over row i, we can get bikes starting at station S_i; similarly, if we sum over column i, we can get bikes ending at station S_i. Using this scheme, we divided our dataset into discrete chunks of data by hour.

Because the bike flow depends on the trips have taken place in the past, our algorithm will aim to predict one hour of station deltas (ΔB_i = # bikes ending at S_i − # bikes starting at S_i where S_i is a station) from several previous hours of trips. If $\Delta B_i = 0$, then the node is balanced; if $\Delta B_i = 1$, then the node is semi-balanced; otherwise, the node is unbalanced. We can implement our model to predict one hour of trips from the previous hour of trips. In data preprocessing, we will use a sliding time window that slides over one hour for each training example. For the task of using the previous hour to predict the next hour of station deltas, our first training example will use hour 1 as x and hour 2 as y. We then slide the window over one hour. The next training example will use hour 2 as x and hour 3 as y. We will then slide the time window again.

In the case where we are using a 1-hour prediction, the input to our model will be a $n \times n$ adjacency matrix where each entry represents the number of trips between two stations during the previous hour. The output of our model will be an $n \times 1$ matrix where each entry represents the predicted number of bikes that each station has gained or lost during the next hour.

We have one training example per hour that data was collected, for a total of around $37,000$ examples.

4.4. Dataset Analysis

In order to gain more insights into the properties of the Citibike network, we conducted aggregate analysis using SNAP and other network visualization tools.

We discovered that the frequency of trips is highly correlated with the time of the day. As shown in Figure 5, during the first week of September, 541 trips took place during 8:00 - 8:05am on Thursday, while 61 trips took place during 12:00 - 12:05am on the same day. However, the bike flow pattern is completely different during the weekend. On Saturday of the same week, 88 trips took place during 8:00 - 8:05am while 96 trips took place during 12:00 - 12:05am. This roughly correlates to the pattern seen in this dataset by Schneider [10].

Because the trip pattern varies significantly based on the day of week and time of day, we conclude that an accurate model will need to incorporate these external factors as training features.

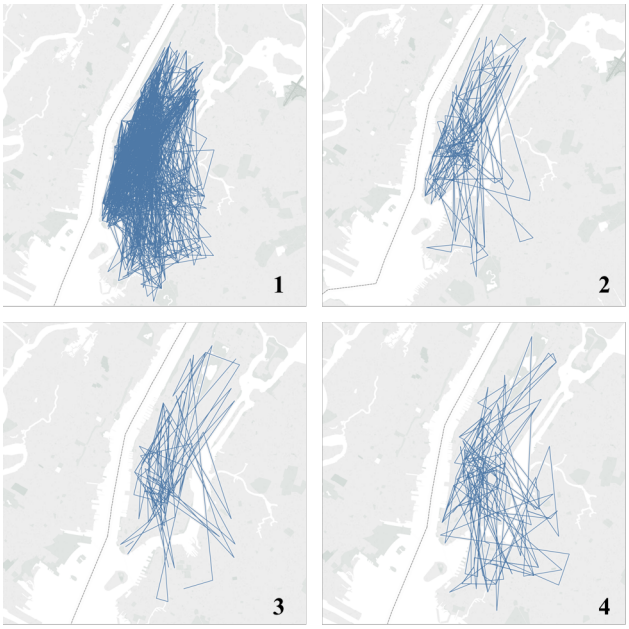

Figure 5: **1.** Bike trip during 8:00 - 8:05am on Thursday.**2.** Bike trip during 8:00 - 8:05am on Saturday. **3.** Bike trip during 12:00 - 12:05am on Thursday. **4.** Bike trip during 12:00 - 12:05am on Saturday

We also discovered that self-loops, which indicates a trip started and ended at the same station, only count as about 2% of all trips. Therefore, we decided not to include the likelihood of having a self-loop for each node as a feature in our machine learning algorithm because it will not help us increase the performance of our model significantly.

We hypothesized that in most cases, the bikes are used for moderate-distance commuting, so the distance between the start station and end station should fall between a certain range. In order to verify this hypothesis, we visualized the relationship between station distance and trip count. In Figure 6, we can see that the distribution is centered at 300 meters and falls off on both sides. The result implies that when the distance between two stations are within a certain (short) range, a trip is more likely to take place. Therefore, distance between stations should be added to the feature set to improve the performance of our machine learning algorithm.

4.5. Dynamic Visualization

As you can see from figure 5, visualizing such a large dataset is difficult, as capturing even a 5-minute interval cluttered the visualization to the extent that it was difficult to meaningfully represent the original data. In or-

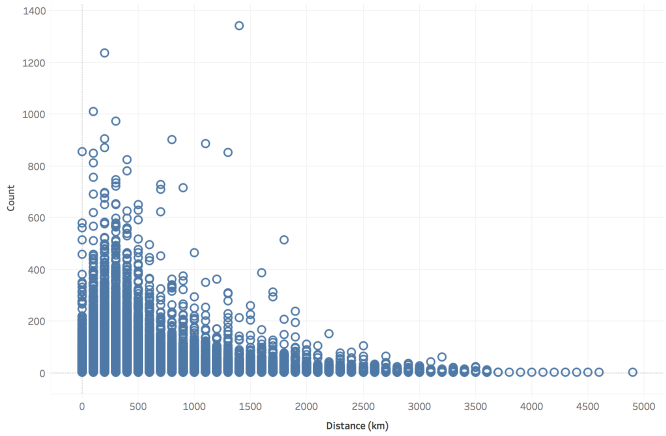

Figure 6: Most Citibike trips occur over relatively short distances. As the distance between stations increases, the number of trips decrease.

der to see and understand the flow of traffic, we built a custom visualization tool using D3.js and Leaflet.js. The custom visualization animates trips between two stations over time. You can see an example of the visualization at http://stanford.edu/∼cqyuan/animated_trips/.

Using this dynamic visualization tool, we observed several months of bike trips on the network. We noticed that a large number of trips happen between a small number of stations in clusters, which reinforced our earlier decision to reduce the dimensionality of the dataset by partitioning the network.

5. Models

This section details the architectures of the models we implemented to solve the node-balancing prediction with the dimensionality reduction techniques we have previously mentioned. The architectures that we have implemented and evaluated are linear regression, feed-forward neural network and recurrent neural network. Moreover, we trained and tested the best architecture on the 6 different graph partition configurations, aiming to find the most optimal graph partition and perdition model combination.

The implementations used a combination of the scikit-learn and Tensorflow packages as the main libraries for machine learning. We divided the dataset into train and test sets, with a 85% to 15% split. Only the train set is used for developing over models so that the performance of the models can be validated on unseen data.

5.1. Architecture

We created the following models to tackle the node-balancing prediction task. All the models use an $n \times n$ adjacency matrix representing an hour time slice of trips between any pair of stations as input and the $n \times 1$ matrix for

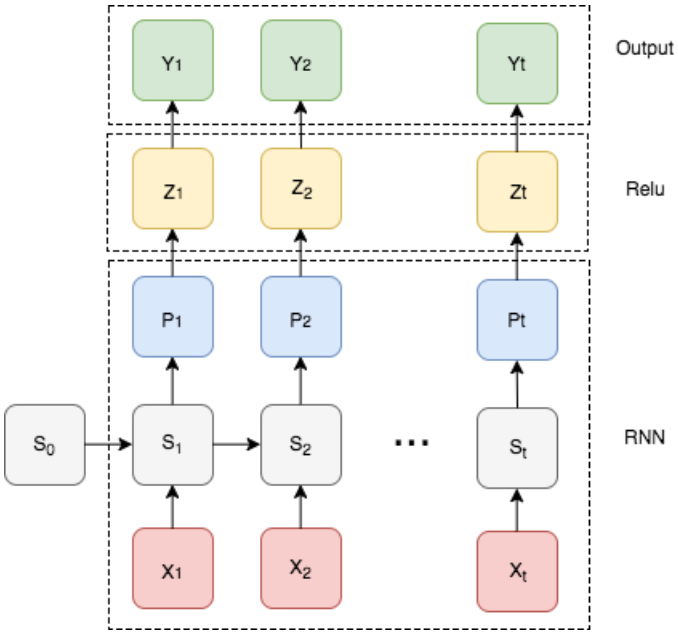

Figure 7: RNN architecture.

change in number of bikes of each station as output, where n is the number of stations after dimensionality reduction.

5.1.1 Linear Regression

Our first implementation was a linear regression model that extracted a linear relationship between the adjacency matrix for one hour and the delta bikes matrix for the next hour. Because the linear model could not capture the more complex nonlinear relationships in the timeseries data, the performance was predictably poor. However, the error of the linear model served as a good baseline for evaluating the neural network-based approaches that we later developed.

5.1.2 Feed-Forward Neural Network

Our next model was a feed-forward neural network containing 2 fully-connected layers with ReLu as the activation function. The increased expressive power of the model led to a significant decrease in both train error and test error.

5.1.3 Recurrent Neural Network

Empirical evidence suggests that recurrent neural network has superior capacity in modeling timeseries data than traditional feed-forward neural networks. Next, we implemented a recurrent neural network model concatenated with a Relu layer to perform the node-balancing prediction task. Figure 7 shows the architecture of the neural network.

6. Results

Table 1 shows the performance of our models on the test set. The linear regression, feed-forward neural network, and recurrent neural network models were trained on train set and then evaluated on the test set.

Model	Training Loss	Test Loss
Linear Regression	1.1302	1.7180
Neural Network	0.8325	1.5223
RNN	0.7110	1.3941

Table 1: Test set performance of models on entire dataset.

The results indicates that RNN is the best model for the prediction task on our dataset, so we decided to choose to further fine tune RNN with different graph partitions. As previously mentioned, we were able to group the stations in the Citibike network into different clusters in order to decompose the prediction problem to predicting the trips within individual clusters. We trained and tested a RNN model independently on each cluster and weighted averaged the losses between clusters for a varying number of clusters k. Figure 8 shows the results.

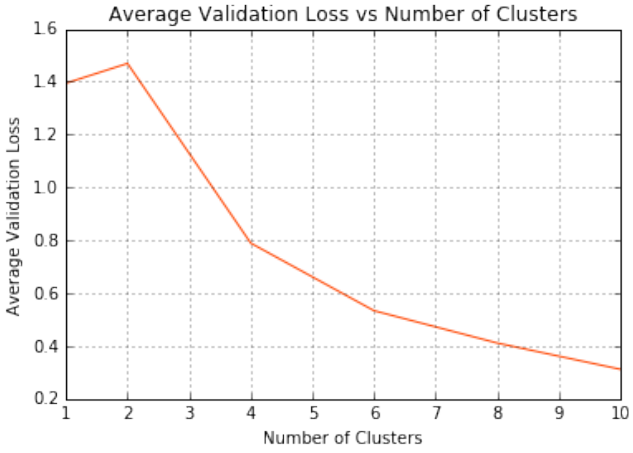

Figure 8: Graph of Weighted Average Test Error vs Number of Clusters. The model was evaluated on each of k clusters, and the losses were averaged based on the number of edges in each cluster.

The plot shows that when $k \geq 2$, as the number of clusters increases, the average loss of the model decreases. One possible explanation is that as the number of stations in each cluster decreases as the number of clusters increases, which allows our model to achieve a tighter fit given the same amount of training.

However, the model will start losing generality if we choose a value of k that is too high . If we divide the graph into more clusters, we are more likely to omit trips that happen between stations that fall into different clusters. We evaluated the percentage p of trips captured for each value of k. Given the adjacency matrix A, p is defined as:

$$p = \frac{\sum_{c=0}^{k} \sum_{i \in S_c, j \in S_c} A_{ij}}{\sum_i \sum_j A_{ij}}$$

Table 2 shows the p value for each k.

k: Num Clusters	% of trips captured
2	60.5
4	61.3
6	57.7
8	55.2
10	44.4
20	28.5

Table 2: As the number of clusters increases, the partitioned clusters capture a lower proportion of trips. The sharp decrease between $k = 8$ and $k = 10$ led us to choose $k = 8$ as the final number of clusters in our model.

Taking both test error and the percentage of trips captured into account, we decided that the optimal number of clusters for the Citibike network is $k = 8$. This value of k allowed us to achieve an ideal test error of 0.4, while avoiding the steep loss in the percentage of captured trips between $k = 8$ and $k = 10$.

After deciding that the RNN with $k = 8$ was the best model to predict node imbalance, we ran our model on several time points to compare its prediction with the ground truth. We have visualized the predictions to understand the results in social and geographical contexts. We will show these results in the next subsection.

6.1. Visualization of Prediction Results

After we trained our RNN model on $k = 8$ clusters, we wanted to get an intuitive sense of the model's performance. We did this by making several predictions on high-traffic time intervals. The resulting visualization gave an intuitive sense of the model's performance.

Here, we highlight two comparisons between our model and the ground truth, with the aim of highlighting the sociological and geographical interpretation of our model. Both of these examples were taken on Friday, September 1, 2017. Figure 9 is a snapshot of the network at 8am. The heat map on the left is our model's prediction. The prediction takes the adjacency matrix for this particular cluster from 7am to 8am as input, and outputs the predicted node balance from 8am to 9am. The heat map on the right is the ground truth – the real node balance from 8am to 9am.

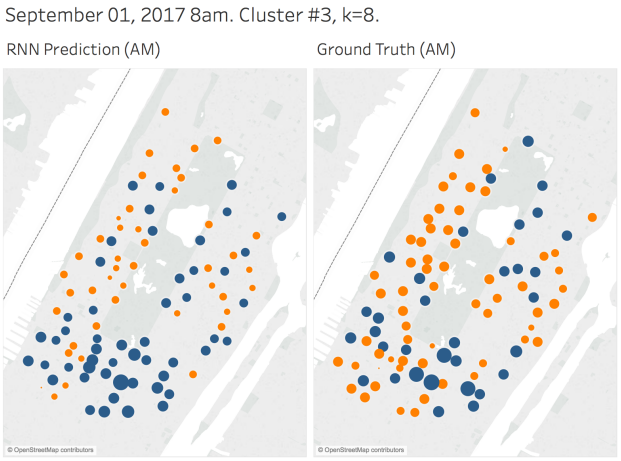

Figure 9: Our model (left) and the ground truth (right). Stations that lost bikes are in orange, and stations that gained bikes are in blue. The Friday morning commuter traffic towards downtown can clearly be seen.

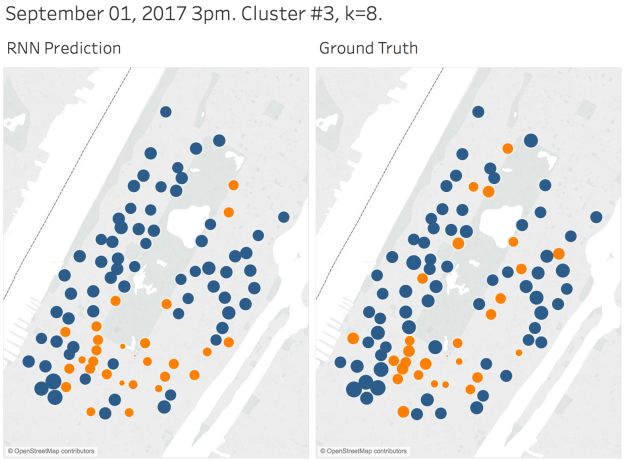

Figure 10: Our model (left) and the ground truth (right). Stations that lost bikes are in orange, and stations that gained bikes are in blue. The Friday afternoon commuter traffic towards the northern residential areas can clearly be seen.

We can see that our model's prediction clearly captures a flow. Specifically, the majority of trips seem to flow from the uptown residential areas towards the downtown Times Square business district. This flow is reflected in the ground truth data.

In figure 10, we see that our model captures the opposite flow. On a Friday afternoon at 3pm, the majority of bike traffic is moving away from the downtown area towards the residential neighborhoods uptown.

With these accurate predictions, the Citibike operators would know to rebalance bikes towards uptown in the morning, and towards downtown in the afternoon.

6.2. Bike Redistribution Algorithm

After the model is trained and we successfully predict the balance of each station, we would like to give the Citibike operators an explicit set of instructions with which they can dispatch trucks around the city to move bikes from the most overflowing stations to the most empty stations. We built this algorithm as follows:

For each cluster in the graph, we will predict at each time point the change in node balance for the next hour. We assume the bike system was balanced in the initial state. The rebalancing algorithm will show the operators a good way to reset the node balance back to what it was at the beginning of the predicted time period.

For example, consider some station that at $t = 0$ had 5 bikes. Between $t = 0$ and $t = 1$, this station gained 5 bikes and now has 10 bikes. We would like to output a series of instructions so that the operator can 'reset' this station back to 5 bikes.

We implemented the greedy algorithm below that chooses the station with the largest change in bikes at each time step until the truck holding the bikes is full. At that point, it will redistribute the bikes to the stations with the largest loss of bikes until the truck is empty. We fist find a number N, so that repeat this process N times the fraction of balanced and semi-balanced nodes is over 50% of all the nodes in the cluster. The algorithm below demonstrates the the truck operates in each cycle and outputs the number of legs traveled, (where a leg is any route between a pair of stations) and the number of bikes that were redistributed. Pseudocode for this algorithm can be seen in Figure 11.

7. Conclusion

In this paper, we were able to explore and visualize the structure of the data from the Citibike bike share system, and use our insights to predict the bike flow in the system. The large amount of sparse data posed a computability problem that we tackled through several dimensionality reduction techniques. We employed both graph theory and machine learning techniques to process the data and create several predictive models. We also present a novel algorithm for the redistribution of bikes in the network based on the output of our models.

The partition-based RNN model we have developed can predict the future change in the number of bikes at each station fairly accurately. The weighted average test error of 0.4 means that on average our predictions are within ± 0.6 bikes range of the ground truth of each station. Such accuracy is sufficient for determining if a station is balanced or not, and deploying rebalancing operations. This result also demonstrates that the dimensionality reduction technique we used

```
def redistribute(bikeDeltas, N, T):
  out = []
  curBikes = 0
  curIndex = 0
  inputStations = []
  outputStations = []
  for i in range(N):
    if bikeDeltas[curIndex] == 0:
      break
    availableBikes = bikeDeltas[curIndex]
    capacity = T - curIndex
    inputStations.append(curIndex)
    # take all bikes
    if availableBikes <= capacity:
      curBikes += availableBikes
      bikeDeltas[curIndex] -= availableBikes
      curIndex += 1
    # take some bikes
    else:
      curBikes += capacity
      bikeDeltas[curIndex] -= capacity

    # if truck is full, go redistribute
    if curBikes == T:
      for j in range(len(bikeDeltas), 0):
        bikesMissing = -bikeDeltas[j]
        if bikesMissing == 0:
          continue
        outputStations.append(j)
        if curBikes > bikesMissing:
          curBikes -= bikesMissing
          bikeDeltas[j] = 0
        # we can use up all the bikes at this station
        else:
          bikeDeltas[j] += curBikes
          curBikes = 0
          break

      out.append((inputStations, outputStations))
  return out
```

Figure 11: Pseudocode for our bike redistribution algorithm.

is successful because it captures the underlying graph structure of the data.

8. Future Work

There are several areas where we could continue working to improve our models and algorithms to improve our recommendations in rebalancing the Citibike network:

8.1. Better Redistribution Algorithm and Metrics for Success

There are several changes we could make to improve our redistribution algorithm.

8.1.1 Minimize Distance

The greedy redistribution algorithm we created minimizes the number of legs driven, but does not keep track of the total distance driven. In order to optimize for fuel consumption or time spent on the road, we could take into account the distance between stations in the rebalancing algorithm.

8.1.2 Baseline Balance

Our current algorithm attempts to redistribute bikes to 'cancel out' one hour of change in the node balance. However, we don't keep track of whether the stations were balanced at the beginning of the time period. It is possible that the stations were more balanced at the end of a time period than at the beginning. For example, if the operator fails to rebalance a station during the previous timestep, then the initial state of this station is not a good baseline for the rebalancing algorithm. In this case, our redistribution algorithm would make the network less balanced.

In order to solve this problem, we need to run real-world experiments to find the balance point b_s for each station s. When station s has b_s bikes, it is least likely to have overflowing or underflowing problems in next timestep. We could use the balance point values for each station as a baseline for redistributing the bikes.

8.2. Discount incentive for overflow stations

Aside from giving directions to trucks that ferry bikes between stations, another potential way to rebalance the bike network would be to weight the price of the bike rentals, giving discounts depending on whether there was a large surplus of bikes in a particular area.

8.3. Model Architecture and Tuning

Another area of improvement would be to experiment with different model architectures and to tune our models for increased performance. For the purpose of our experiments in this paper, we used relatively small scale simple models to iterate quickly and reduce the amount of parameters trained/computing resources needed to work with our large dataset. Given more time and computing power, we could tune the model hyper parameters, train for longer periods of time, and add additional layers and training techniques such as Dropout[11] and Batch Normalization[3] which have been shown to further improve model performance.

8.4. Time as a Feature

As we have showed in the Dataset section, the volume of the bike traffic is highly correlated with the time of the day. Therefore, adding time of the day as a feature to the model can help the model learn the expected traffic pattern through the day. Additionally, other time-related feature s, such as weekday, weekend, month, are likely to improve the performance of the model.

References

[1] Citibike raw data, 2017.

[2] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *arXiv:cond-mat/0112110v1*, 2001.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[4] P. P. Jones, N. C. *Introduction to Bioinformatics Algorithms*, volume 8.9. Fragment Assembly in DNA Sequencing. MIT Press.

[5] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth Annual ACM International Conference on Information and Knowledge Management (CIKM03)*, page 556559, 2003.

[6] S. S. M. Z. Mohammad Al Hasan, Vineet Chaoji. Link prediction using supervised learning. In *2006 SIAM Conference on Data Mining*, 2006.

[7] M. E. J. Newman. Analysis of weighted networks. *Phys. Rev. E*, 70:056131, Nov 2004.

[8] A. Ng. Stanford university cs229 lecture notes. Part VI Learning Theory.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[10] T. Schneider. A tale of twenty-two million citi bike rides: Analyzing the nyc bike share system.

[11] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.