Scott Shi

UID: 204 422 750

Professor Miodrag Potkojnak

Lab Report CS 199

**I. Problem Formulation**

**I.1 Data and Tools**

For this lab, we are given data for taxi trips in Porto, Portugal. However, some of the data of a few of the recorded taxi trips are missing, specifically the GPS location of the taxi after a certain point in its trip (i.e. A taxi will have a few ticks designating its coordinates as it progresses along its trip, and then its consequent coordinates will be missing, including its end destination) The data is presented in a \*.csv file and each trip contains the following information:

-unique id for the trip

-how the trip was requested (from taxi service, taxi stand, on the street)

-unique id for the number which was used to call

-unique id for which taxi stand it was requested from (if it was from a taxi stand)

-unique id for the taxi

-timestamp for when the trip started

-kind of day trip was requested on (holiday, before holiday, regular day)

-boolean – is there missing data for this trip?

-list of GPS coordinates of the trip taken every 15 seconds of the trip

The main tools we used to tackle this project are the programming languages R and Python and the libraries that are available to either of the languages.

**I.2 Tasks (TO BE EDITED)**

We first want to be able to predict where a taxi will end up with a fair amount of accuracy even with missing data.

**II. First Task: Taxi Endpoint Prediction**

**II.1 Attempt 1 – Iterative Similarity Comparison Approach**

To predict where a taxi would end up, we focused mainly on the “polyline” section of the data provided that described the coordinate locations of each taxi on a certain trip every 15 seconds. We aimed to predict the endpoint by trying to group together all similar paths and narrowing down our choices of similar paths until the endpoints of all similar paths seem to all converge to a single endpoint (algorithm explained more in detail below).

The first challenge was to differentiate between the coordinate points. In the figure below (Figure 1), we see the main clusters of endpoints of the taxi trips that do not have missing data. It was hard to do this because the coordinate points given to us were originally used to describe locations on a global scale and Porto is geographically small. Therefore, we outlined a geographical box using the outermost coordinates that the data gave us and *normalized* all other data points to be scaled within this box. Now it is easier to differentiate one point from another.

The second challenge was to determine if two paths were similar or not -- if they were headed in the same direction. To be able to do this, we had to create a similarity function. To do this, we took two sets of coordinates from two different time ticks (i.e. beginning tick and 7th

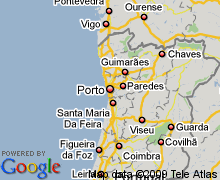




Figure 1. Endpoints of Taxi Trips in Porto (Python), and Porto’s Size on the Map (Google)

tick) and created a vector consisting of all those points for a single path. We did the same thing for the path we wanted to compare against, and determined the *cosine similarity* for both of these vectors. If these paths outputted a cosine similarity greater than a certain threshold that we defined (close to 1, as 1 means there is an angle of 0 between the vectors), we would return that the two inputted paths were similar.

Lastly, we wanted to use the above tools to attempt to predict the paths given limited information on the ticks. The algorithm goes as follows:

1. First we created a vector out of out of the ticks 1 and 4 (arbitrarily chosen) from the path, *p*, whose endpoint we wanted to predict. This vector contained the *normalized* coordinates of *p* at ticks 1 and 4.

2. Next, we grabbed all other paths that had similar 1st/4th ticks as path *p* using the *cosine similarity* function and stored them in a set S. We looked at the endpoints of all paths in set S and evaluated how spread apart the endpoints were. We did this so we could determine if the endpoints of the paths in S seemed to converge to a single endpoint.

3. To mathematically evaluate if the paths in S met this criterion, we used the *coefficient of variation*. If the coefficient of variation was below a certain threshold (which we defined), we claim that there was not much relative variability between the endpoints’ coordinates and that we could safely predict the endpoint of path *p* as the mean of all the endpoints.

\*\*However if the coefficient of variation was too high, we would now consider the 4th and 7th ticks (also arbitrarily chosen) of path *p* and compare the vector created from those normalized coordinates against the 4th and 7th ticks of all paths in set S (back to step 1). This would narrow down my set of possible similar paths to S’ with |S’| |S|. We would repeat this process until we found a set of paths that we could claim converged to an endpoint or we had no paths left in which case we would determine the prediction inconclusive.

To conclude, we found that this iterative approach was simply not robust enough. If there were few ticks for the unknown paths, our predictions were less than 50% accurate. Additionally, in the simple counterexample in which all destinations end up in two places, this algorithm would at best return the midpoint between these two endpoints. This pushed us to change our focus to not output a single prediction that could either be right or wrong but multiple answers with probabilities associated with each of them. This led us to our next, k-means clustering approach. Additionally, we kept the *normalization* and *cosine similarity* algorithms we developed earlier for future use.

**II.2 Attempt 2 – K-Means Clustering Approach**

This approach centered around using the *k-means clustering* method. The goal was to create clusters of all known endpoints and associate with each cluster a probability that the unknown trip will end up in that cluster. We would update these probabilities with each successive tick of the unknown path and in the end output a list of all possible clusters the path could end in with their probabilities. Therefore, when given all the coordinate points, an unknown path, and the ticks to compare, it is possible to tell based on the data how likely the unknown path is to end up in any number of the given clusters that we created.



Figure 2. An example of clustering on a map.

This approach is better not only because it does not depend on predicting the exact endpoint of the trip but rather a local region. This is okay because since clusters are fairly local regions, there is not much loss of precision if we fail to pinpoint the exact endpoint of an active taxi trip. Additionally, this will be able to give us more accurate predictions as it considers all possible clusters of endpoints and gives us a probability of the unknown taxi trip ending up there for each of the clusters.

Another important point to bring up is determining the number of clusters to create. The algorithm essentially works by taking away the largest edges in a connected graph until you have the desired amount of clusters. The way that we determine how many clusters to use is through the *elbow method*. By using this method, we plot the average dispersion between the clusters against the number of clusters. By visually looking at the graph, we can see a point where there is diminishing returns in creating new clusters. Although there is not always a distinct “elbow” in the graph, we simply set a threshold on when there is not much change in dispersion when going from (c) clusters to (c+1) clusters and we choose “c” to be our number of clusters.

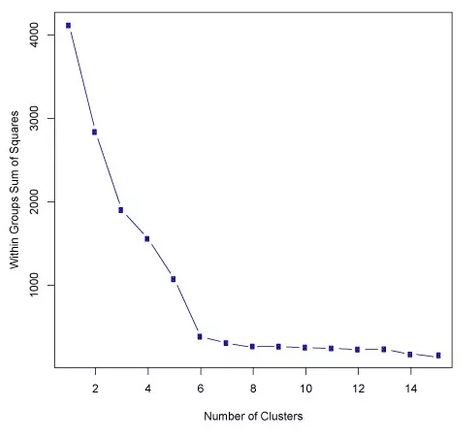


Figure 3. Elbow method – find the “joint” in the graph to determine point of diminishing returns.

(TODO: Conclusion about using the clustering approach to estimate how many anticipated active taxi trips will end up in any given cluster. i.e. for one unknown taxi trip the predicted clusters are 0.21, 0.10, 0.40, etc…, I will assume that many taxi drivers will be available at each cluster. Given a large amount of taxi drivers, trained on data several times, I’ll add up all those fractions and get a 95% confidence interval and have that estimated number of predicted drivers)

\*repeat many times (pick a random sample of training data)

\*of all active trips, run clustering algorithm and get fractions of trips ending up in

a given cluster (.21 in cluster A, .14 in cluster B, etc…)

\*Add up all these fractions from all the unknown trips and get a certain amount of

predicted taxi drivers in each cluster (3.45 in cluster A, 4.6 in cluster B, etc…)

\*construct a range of different predictions from each of the iterations

\*We construct a 95% confidence interval from these trials

**III. Second Task: Customer Prediction**

(TODO: IMPLEMENT AND FILL THIS OUT)

**IV. Fourth Task: Bipartite Matching (Free Driver to Customer)**

Now that we can predict where the taxi drivers are going to end up and where customers are likely to pop up at given time intervals, we need to tackle the following optimization problem:

\*\*Given a list of free/to be free taxi drivers, waiting customers, and an upper time limit on how long a customer can wait for, match customers to drivers in order to minimize the total amount of gas used by taxi drivers.

Since we are trying to minimize the amount of gas used by taxi drivers, we are essentially trying to minimize the distance they have to travel to pick up customers. We encounter the problem of whether a customer should be paired with an immediately free taxi driver at a farther distance or if the customer should wait for a short period of time for a currently active taxi driver to arrive at a closer location. To set up this problem, we use the parameters given (i.e. time limit on how long a customer can wait) to find all currently free drivers and predicted to be free drivers at any given cluster. Additionally, we use the information on where customers are predicted to pop up. With these two groups available, we perform *bipartite matching* to try to match each customer to each driver. To create the graph, we connect an edge between every driver and every customer with the weight being the distance between the two. We prefer lower distances / weights over higher ones. So after performing a matching between customers and drivers, we are able to solve the above question of minimizing gas usage.

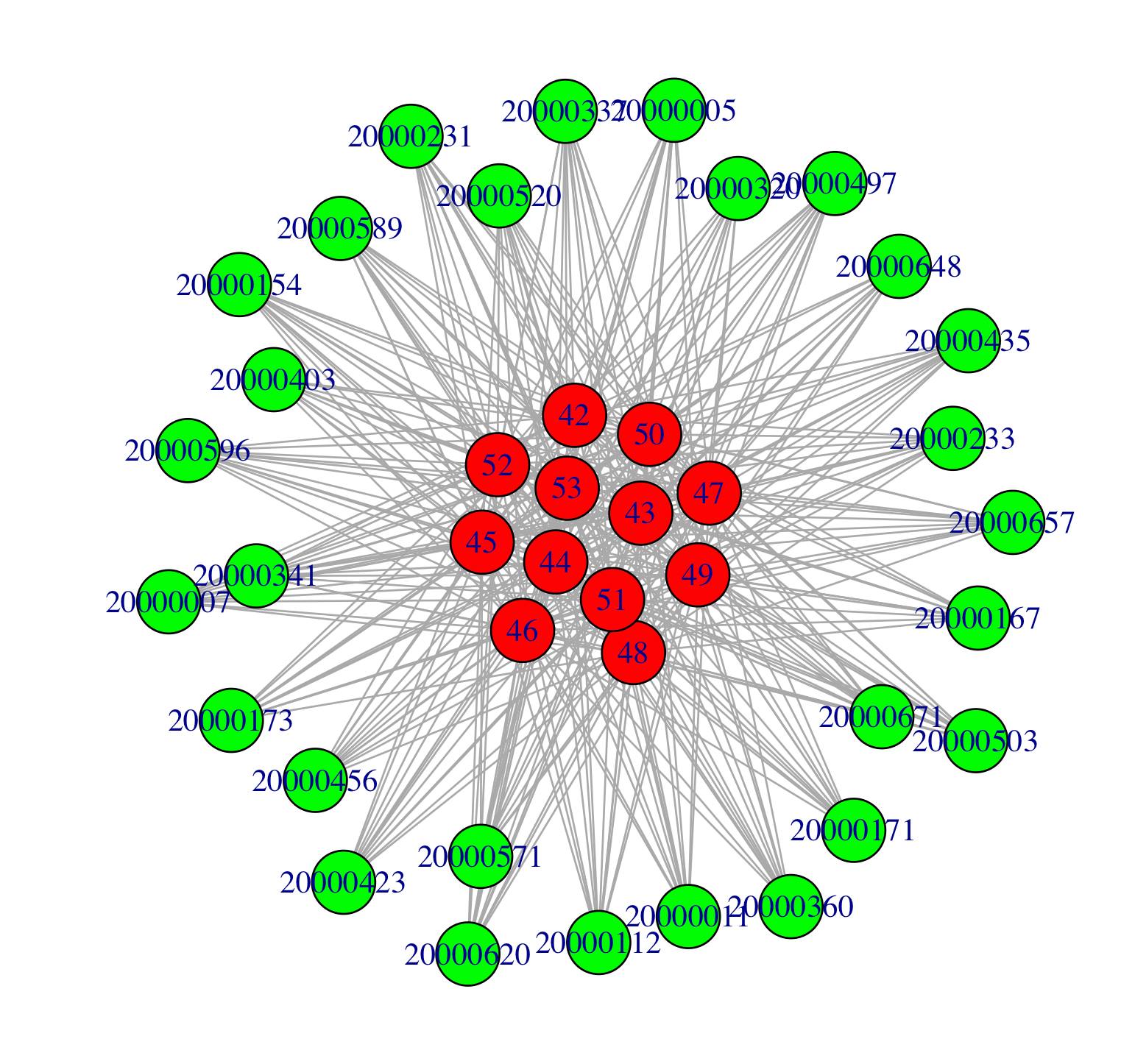


Figure 4. Bipartite graph of free drivers and customers

**V. Fourth Task: Bipartite Matching (Free Driver to Customer)**