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. We picked up R from scratch. Some of the concepts that we used to implement our code was suggested to us by Potkojnak, others were just googled. For all these concepts, we researched all background information necessary for us to understand how it works, and the tradeoffs involved in using one method of solving an issue over another (i.e. using k-means clustering instead of logistic regression).

**I.2 Tasks**

We first want to be able to predict the region where a taxi will end up with limited information on its trip (i.e. only the first couple of ticks of a taxi’s trip). We also want to be able to predict what regions customers tend to pop up at given times. Furthermore, based on the customer data, we want to see if there’s a trend between where customers leave from and arrive to earlier in the day and where customers leave from and arrive to later in the day (perhaps a group all leave to an area to work in the morning and all come back from work around the same time at night?). Finally, we want to be able to perform bipartite matching on free drivers and waiting customers and match them based on shortest distance.

**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

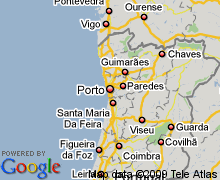




Figure 1. Endpoints of Taxi Trips in Porto (Python), and Porto’s Size on the Map (Google)

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 distance between points within 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.

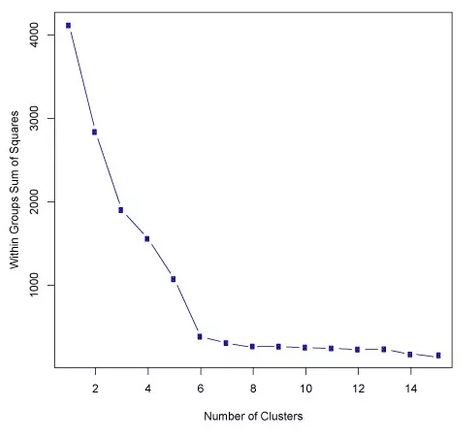


Figure 3. Elbow method – find the “joint” in the graph to determine point of diminishing returns.

I want to use this clustering approach to estimate how many anticipated active taxi trips will end up in each given cluster. To do this, I will assume that in a given taxi trip, if it outputs probabilities of 0.21 of going to cluster A, 0.14 to cluster B, etc…”0.21 drivers” will be available at A, “0.14 drivers” will be available at cluster B, and so on. Given a large amount of in progress taxi drivers, I can add up all those fractions and predict how many taxi drivers are expected to pop up within a given time frame. This will be the data from a single sample. To get a more solid prediction, I will take a random sample many times and construct a *95% confidence interval* for how many active taxi trips will end up in a specific region for every region.

\*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 for each cluster from these results

From here on, I will use these confidence intervals of predicted amount of taxi drivers per cluster at a given time in the following way. I will take the center coordinate point of the cluster, the 25% value of the confidence interval (I want to be more skeptical in my predictions), and claim that many taxi drivers will be around that coordinate point. In the future, I can use this data as amount of “to be free” taxi drivers in this area and consider them in the final step of matching available or to be available taxi drivers to pending customers.

**III. Second Task: Customer Prediction**

In this section, I try to predict where customers tend to pop up at any given time. To do this, I take a similar approach to predicting how many drivers will end up in specified regions at a given time. To do this, I must first gather all coordinates of customers within a given time frame. I simply consider all completed trips with timestamps within my given time range as my training data. I treat the first coordinate tick of each of these locations as the customers’ locations. With this dataset, I run the following algorithm:

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

\*run k-means clustering algorithm on those beginning points

\*record centers of the clusters and how many points belong in those clusters

\*save the data into a list that holds the results of each random sample

\*Construct a 95% confidence interval for each cluster from this list of results

From here on, I can do two things with this data. The first thing I can do is to predict the amount of customers I can anticipate to pop up in specific regions. Again, I will use the 25% mark within the confidence interval to anticipate how many customers will show up around that specific region’s center coordinate points. I will use this data in addition to the “free taxi driver” data from the previous section to match them in the later section.

Additionally, I will use this data to predict trends in customer requests. For example, if a large amount of customers tends to request taxis from a certain “departure” area during a specific time and most end up in the same “arrival “area. Perhaps this could be interpreted as workers all taking taxis to work in the morning. Likewise, we could also see if there were a lot of customers who requested rides from that “arrival” area to the “departure” area in the evenings. In Figure 4 below, we can see that there are some regions in which customers request much more on average at a specific time. This brings us to my next area of research – customer arrival / departure trends.

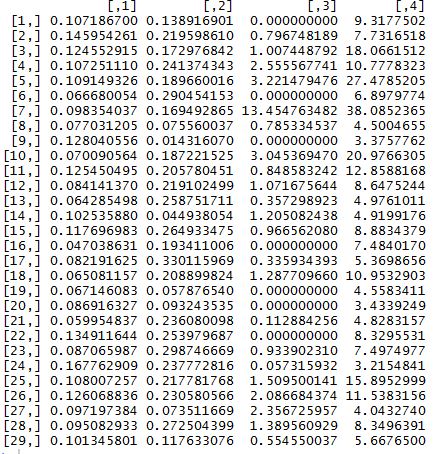


Figure 4. Results of 200 random samples of trips that started within a certain time frame. There are 29 clusters that I found. The first two columns are the coordinate points of the clusters, the third and fourth columns are the 95% confidence intervals denoting the number of customers I expect to come from each region. You can see that I can expect many more customers from some regions than other regions.

**IV. Third Task: Customer Departure/Arrival Trends**

**(TODO: IMPLEMENT AND REPORT)**

**V. 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.

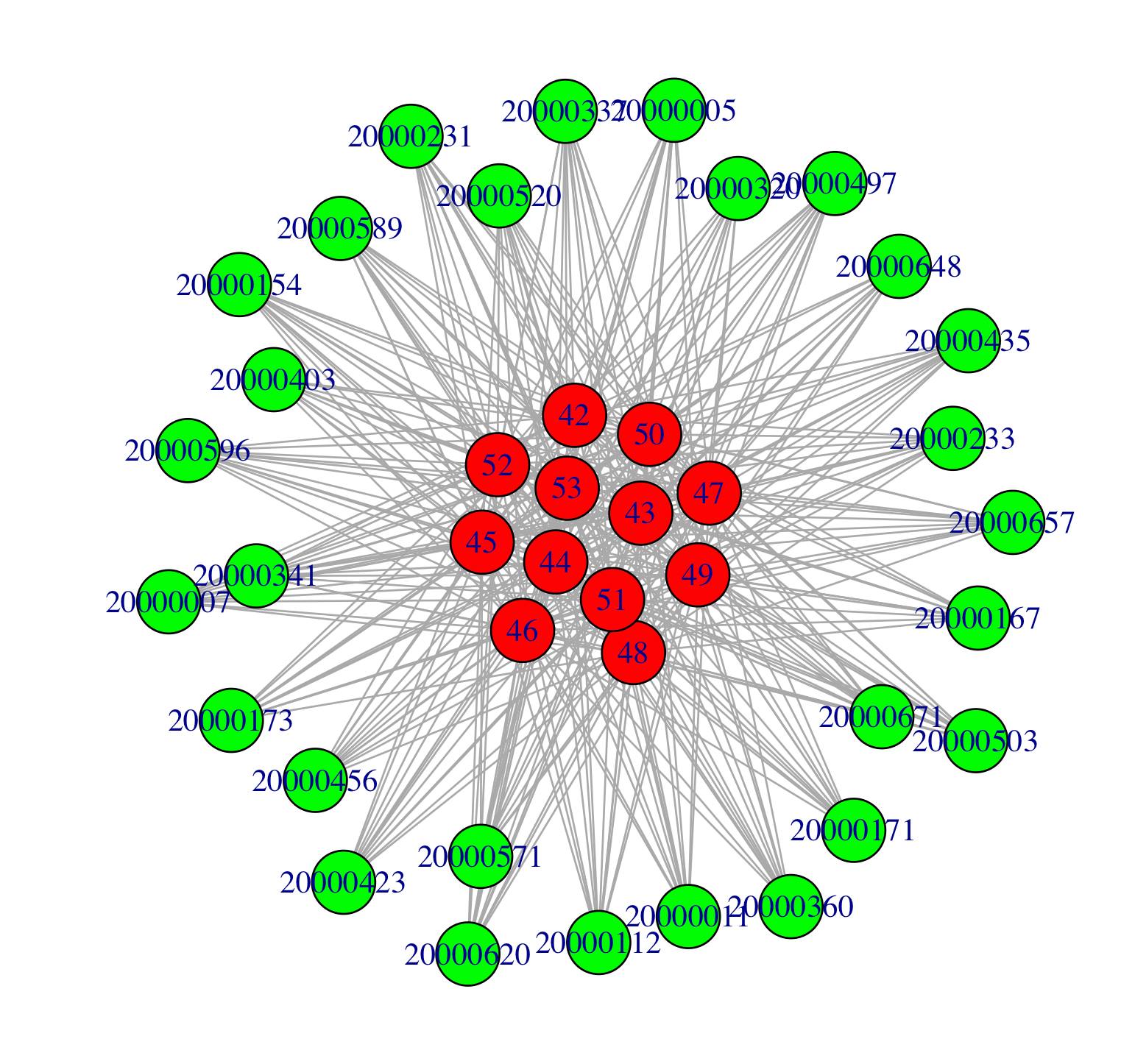


Figure 5. Bipartite graph of free drivers and customers

**V. Conclusion**

**V.1 Summary**

Our overarching goal was to try to minimize gas cost of taxi trips. In order to do this, we would need to try to minimize the distance taxi drivers needed to travel to pick up customers. To tackle the tasks in this problem, we had to create some tools such as a function to see how similar paths were that used *cosine similarity*. We also had to *rescale* the coordinates so that we would be able to differentiate one from another more easily. With these tools, we were able to use many statistical modeling techniques such as *k-means clustering* and *confidence intervals* in order to make some predictions. Our predictions consisted of where we expect customers to pop up and where we expect active taxi trips to end up. We also used this data to try to locate trends in when and where customers requested rides. Finally, we combined our prediction data to try to match

**V.2 Future Tasks**