Transit App Data Analysis: Inferring User Home / Work Locations from Location Data

Danielle Stacy[1]
Candace Brakewood[2], **Kwai Wong**[2], **Cheng Liu**[2]

[1] The University of Alabama
[2] The University of Tennessee

Abstract

The goal of this project is to infer the home and work locations of users of the Transit app by analyzing their location data. This information could be useful to the developers of Transit so that they could incorporate useful features related to users' home and work locations. To achieve this goal, it is necessary to make use of the location data collected on the users of the app and the locations of the home and workplaces of the users who chose to save that information.

As location data is collected on a typical user over a long period of time, it is expected that their data would cluster in two discernible clusters: their home and work locations. Once these two clusters are found, they need to be labeled home and work. To distinguish the home and work clusters, the timestamp of the location data in analyzed. The cluster that contains more location data on the weekend is labeled home and the other is work. This project was completed by use of OpenDIEL, bash script, and Python.

Background

Transit is an app used to collect and map real-time public transit data in 167 different cities across the world[1]. The app has been developed to instantly display any real-time transit data around the current location of the user[2]. People may use the app to determine what bus or train route to take, to plan a trip, or to request a car-sharing or bike-sharing service. As a person uses the app, they generate data such as their current location, features they used, apps they have downloaded, and locations they have saved[3].

All the data has been organized into thirteen categories, and descriptions of each of these categories has been listed below[3]. The data was originally recorded in JSON format, but it has been converted to CSV format for easier analysis. Each user has been assigned a unique

identifier so that their actions may be tracked between the tables. This unique identifier is referred to as the udid, which stands for unique device identifier.

Table Names	Description of Contents
Device	Contains a Transit app specific identification number (device ID), device type, model of device, operating system, version of Transit app, and last date of app use
Favorite	Provides information on user designated favorites in terms of transit routes
Feed Download	Provides a summary of activity on the Transit app by day
Installed App	Reports on other installed apps on the user's device that can impact functionality, such as the Uber app
Location	Includes the location (lat/long) and a unique session ID for each time the app is opened
Nearby View	Contains information about the transit routes presented to a user in each session upon opening the app
Placemark	Includes location data from an optional function that stores places users often go (e.g. home or work)
Session Complete	Provides an event based view of each session, including the beginning and ending location
Sharing System Actions	Provides data on the booking of carshare, bikeshare, and other services, including the location of shared vehicles
Sharing System Purchase	Provides purchase records for shared vehicles, which are primarily bikeshare passes
Trip	Contains information about usage of the trip planning feature, including start and end coordinates (lat/long)
Uber Request	Lists requests for service from Uber, which are then handed off to Uber's app for fulfillment
User Feed Session	Includes the number of times the app is opened and the different transit agency's data accessed by the user

I have been given 418 days of data from 2015 to 2016, organized so that each day of data is split into thirteen files corresponding to the thirteen categories. All of this information totaled to over a terabyte of compressed data. For the results discussed in this paper, I focus on the data from the month of April 2016. Even just within April, I was looking at over 30 gigabytes of data.

The tables that I am interested in for this analysis are the session complete table, the placemark table, and the favorites table. When first presented to me, I believed I only needed the session complete table and the placemark table. The placemark data that I received, however, only contained 52 users who had saved their home or work locations and no user had saved both their home and work locations. This issue was solved when investigation revealed much of this data was saved as a feature of the favorite category in the JSON file and was lost during the conversion to CSV file. The script to convert the files from JSON to CSV was rewritten, and now I have over 4,000 users who saved both their home and work locations just on April 1, 2016.

OpenDIEL is software designed to allow many units of parallel code running seamlessly under a unified executable and to allow these individual programs to exchange data specified by the user[4]. I use the engine to parallelize the code so that I can complete my analysis with the large

dataset in a timely manner. Bash script is used to extract the data from the files that I need, and I run my clustering algorithm using a bash script. Python is the primary computing tool used for this project. It is useful due to its extensive libraries used for data science and its simple syntax.

Objectives

There are three basic steps to completing this project. The first encompasses extracting all the necessary location and time data for each unique user from the session complete table without losing too much time. Next, the location data for each user needs to be clustered to find areas of high density. This clustering algorithm needs to assign each data object to a discrete cluster so no data object should belong to more than one cluster. Additionally, the clustering algorithm should cluster data around two distinct points. Finally the results of the clustering need to be validated so that an accuracy rate may be established. The accuracy rate needs to reflect how close the algorithm-found home and work locations are to the actual home and work locations and fall within a reasonable margin of error. The actual home and work locations can be found in the placemark and favorites tables.

Extracting the Data

From all the data tables given to me, I need tables that contain information about location and time and information about the home and work locations. The session complete table contains location information and timestamps for users while they were using the app, and it was a more reasonable size than the location table. The placemark and favorites tables contain the coordinates of locations saved by users accompanied by the unique identifier of each user. These are the only tables I need to complete my analysis.

To achieve results that can be easily validated, I only want to analyze the location data of users who had saved their home and work locations. Because of this, I extract all the udids from the placemark and favorite tables who had saved their home and work locations. This task is accomplished using an algorithm written in Python. I choose to use Python because there is a module in Python called csv that makes reading and writing csv files easier.

The algorithm opens the file that contains the information of the users who saved a certain location on April 1. The information stored in the file is the user's udid, name of the location, latitude of the location, and longitude of the location. My algorithm simply runs through this file and checks if the name of the saved location is "Home" or "Work." If the name is "Home," then the information for that saved location is written to a file containing the information of saved home locations. If the name is "Work," then the information for that saved location is written to a file containing the information of saved work locations.

Now that I have all the saved home and work locations, I need to know that each saved home and work location is unique. The algorithm described above fails to check if there are duplicates in the file. To remedy this issue, I have another algorithm written in Python, used again for its csv module.

To check for duplicates, the algorithm opens the home and work location files and checks every udid saved. It also opens two new files so that the unique home and work locations are saved to new files. If a udid is unique, then its information is written to either the unique home or work file, depending on if the location is home or work. If a udid has already been seen, however, then the information has already been written to a file so the algorithm moves to the next udid.

At this point, I have every unique home and work location, but it is unknown which users have saved both their home and work locations. To find the users who saved both their home and work locations on April 1, I have another algorithm written in Python, used yet again for its csv module.

This algorithm opens the files that contain the unique home and work locations. To determine which users have saved both their home and work locations on this day, I examine the udids of both files. If a udid exists in both files, I write that udid, the home latitude, the home longitude, the work latitude, and the work longitude to a file that will store this information for all users who saved both their home and work locations.

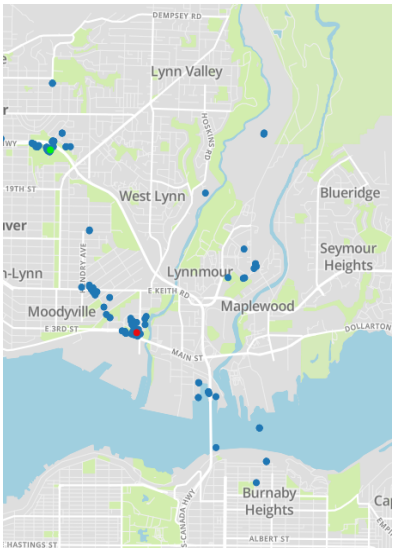
Once I have all the users who had saved their home and work locations, I need all their location data. The location data was saved in the session complete tables. I search the session complete tables from the month of April for the udids who had saved their home and work locations and write all their location and time information to a unique file for each udid using an algorithm written in Bash. I choose to write the algorithm in Bash for its simplicity and speed.

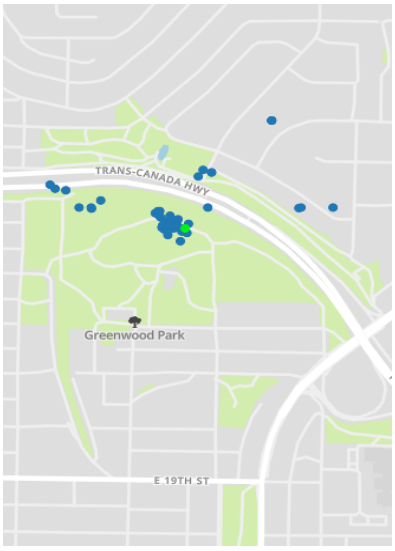
This script takes a session complete file and the file that contains the information of the users who saved both their home and work locations made from the last algorithm. It reads all the udids that have saved home and work locations and searches for those udids in the session complete file. If the udid it is searching for is found in the session complete file, it takes the data from the session complete file and writes it to a new file that stores the location information of that udid for that day. At the end of this process, there will be 30 files for each udid. The 30 files for each udid are then combined into one file by another Bash algorithm.

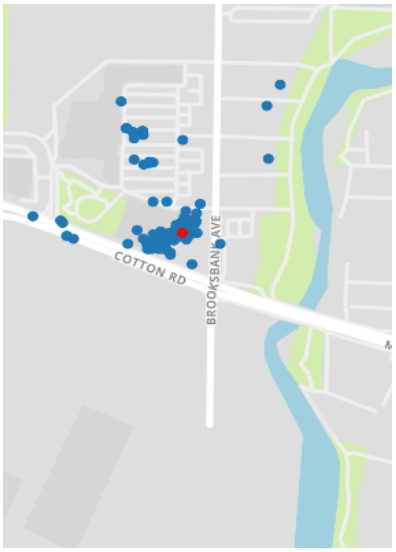
The algorithm that extracts the location information from the session complete table had to be parallelized because I was working with 4000 udids and a month of session complete data. Since each month is split into days, the algorithm has to search through 30 days of data for 4000 udids. If this was computed serially, it would take several days to successfully complete. Thus, the code

is parallelized using OpenDIEL. The workflow is to set up directories for the files that will be created, to search for the udids in the session complete files and write them to files in the directories created, and then to concatenate the files made for a single udid.

Clustering the Data





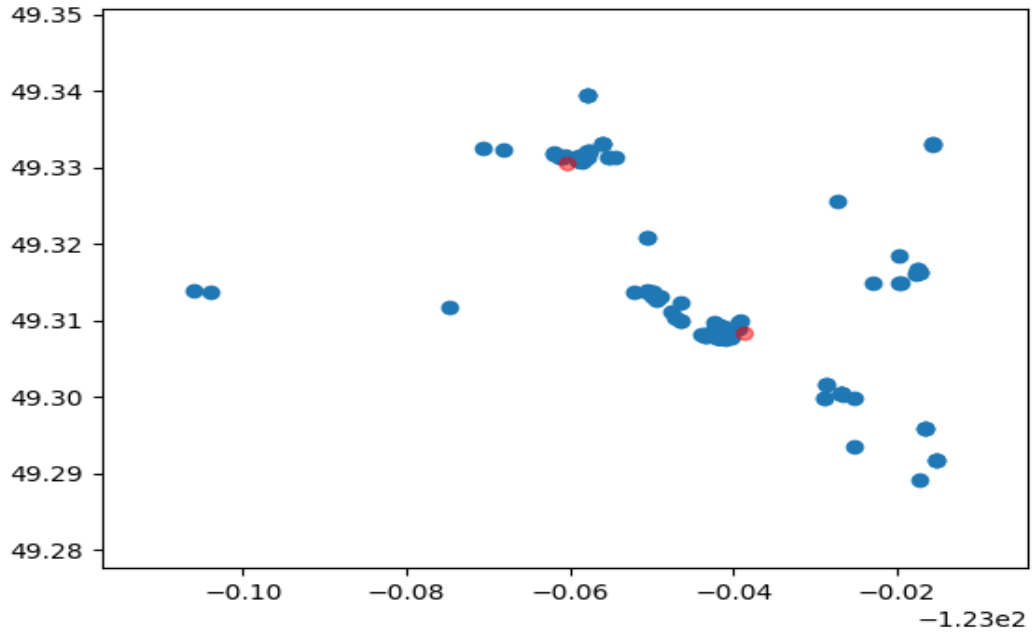


In these images, the location data of this particular user is represented by the blue points, the work location is represented by the green point, and the home location is represented by the red point. The points have been shifted by a small amount in order to protect user anonymity. Two fairly distinct clusters do appear around the home and work locations. The clustering algorithm should detect these clusters and label them as home and work appropriately.

I choose to cluster the location data with the k-means algorithm because it will cluster the data into a specified number of discrete clusters. I use Python for the KMeans algorithm imported from the sklearn.cluster module and specify the number of clusters to be two. Because k-means is sensitive to outliers, I need to account for clusters created by strong outliers. To do this, I set a minimum cluster size. If a cluster does not have the minimum amount of points, then those points are removed from the dataset and the data is passed through the clustering algorithm once again.

Since I only want to cluster the location data of users who have a prolonged period of use, I must filter out files with less than ideal amounts of data. To do this, I run my Python script using a Bash script, specifying to only cluster the data of files with at least 50 lines of information. This way I am not clustering any data that might not even meet the minimum cluster size.

The scatterplot below represents the results of clustering the data from the maps shown above. The blue dots are simply the location data, and the red dots are the centroids of the clusters. The centroids are fairly close to the actual locations.



Once I successfully find my clusters, I need to label them. I choose to label them by selecting the centroid of the cluster with the larger proportion of location data taking place on the weekend as home and the centroid of the other cluster as work. This data is saved by writing the udid, the home coordinates, the work coordinates, the algorithm-found home coordinates, and the algorithm-found work coordinates to a file. The image below shows the comparison of the actual locations with the algorithm-found locations (shifted to protect anonymity) from the scatterplot above. The algorithm-found home and work locations are indeed very close to actual home and work locations.

```
Home: [49.3079726, -123.0410949]
Work: [49.3110115, -123.0583576]
Algorithm-found home: [49.30841905263158, -123.0386882631579]
Algorithm-found work: [49.330624179104476, -123.06033597014925]
```

Accuracy

Now that I have my algorithm found home and work locations, I need to validate my findings. Since I only used udids with saved home and work locations, I can easily check my work against the true locations. I need my results to check the accuracy, so I open the file that has stored the results of the algorithm and the saved locations of the home and work.

Once I have the results, I can begin to check them. To check the accuracy, I simply check the difference in latitude and longitude of my algorithm-determined locations and the actual locations. If the home coordinates are within 0.05 of my algorithm locations, then I consider that home point accurate. I use the same process for the work coordinates.

After all the results have been checked, I establish an accuracy rate. This is done by finding how many home and work locations my algorithm found correctly within the 0.05 margin of error. The number of correct locations I found are divided by the total number of udids clustered. I want to establish separate home and work accuracy rates, so I take the number of accurate homes and divide that by the total number of samples analyzed to establish a home accuracy rate. The same process is used with the accurate work locations to establish a work accuracy rate.

Results and Future Work

With the process I have just described, I am able to establish a home accuracy rate of 65.5027% and a work accuracy rate of 56.1108%. These are the overall best accuracy rates I have been able to achieve. Other labeling methods I have tried include counting Friday as a weekend, checking the location most visited on Sunday, and checking the location most visited at night. Including Friday as a weekend actually yielded a slightly better work accuracy rate, but the home accuracy rate was worse than the work accuracy rate was better.

In addition to establishing the accuracy rate, I am interested in seeing how far away my algorithm-found point is from the actual point. Interestingly, the weekend labeling system did not achieve the points with the smallest average distances from the actual locations. Here are some different results attained by the different labeling systems:

```
Home accuracy rate: 0.6550270921131849
Work accuracy rate: 0.5611077664057796
Average distance from home: 0.09856185921476948, -0.7193104142689638
Average distance from work: 0.1440318463053732, -0.7285809828786194
```

```
Home accuracy rate: 0.6309452137266707
Work accuracy rate: 0.5668272125225767
Average distance from home: 0.055667596132212796, -0.4457047546287139
Average distance from work: 0.18542354841196354, -0.991841823228436
```

```
Home accuracy rate: 0.6503159795365634
Work accuracy rate: 0.5362624134817936
Average distance from home: 0.0018536861337655106, -0.5682967575081299
Average distance from work: 0.24550316712400705, -0.8802563548791372
```

```
Home accuracy rate: 0.629930743751882
Work accuracy rate: 0.5380909364649202
Average distance from home: 0.15320578694916834, -0.9338232058354968
Average distance from work: 0.08721401422151914, -0.5018262899262531
```

The first image is from the labeling system that labels the location most visited on Saturday and Sunday as home. The second image is found by labeling the location most visited on Friday, Saturday, and Sunday as home. Below that is the results from the labeling system that labels the location most visited on Sunday as home. The final image is achieved by the labeling the location most visited from 8 pm to 8 am as home.

These are not excellent accuracy rates for as large of a margin of error that I have. Latitudes and longitudes with differences of 0.05 are as far as 4.81 kilometers from each other. I would much prefer a margin of error as small as l kilometer with these accuracy rates. In fact, some home and work locations are not even 4 kilometers apart. The difference in the home and work locations of the example I have used throughout this paper is much less than 4 kilometers.

For future work, I would like to improve the accuracy rate and narrow the margin of error. To do this, I would reevaluate my clustering and labeling methods. Switching to a different clustering method would be more complex than using a different labeling method. Hopefully only a different labeling method is needed to improve the accuracy as opposed to something being fundamentally wrong with the clusters.

I have consistently used 2 clusters throughout this project for simplicity's sake, but it is possible better results may be achieved from more clusters. Two clusters is not as precise as it could be and often the clusters contain points far from the home and work locations. Although k-means has proved rather useful in this experiment, I would like to try clustering methods that are less susceptible to outliers and do not require the program to specify the number of clusters needed. DBSCAN, for example, is a density-based clustering algorithm. It groups together the data points that are closely packed together and marks points that occur alone in low-density regions as outliers. Additionally, DBSCAN does not require a predetermined number of clusters.

Although I have attempted many different labeling methods, there are many more left to try. I have yet to deviate from labeling the centroids of the clusters as the home and work locations, but perhaps the centroid is not representative of these locations consistently. Since the clusters can contain points that are rather far from the most dense regions of the cluster, the centroid will be shifted from the heavily packed regions. Selecting the most dense areas of the clusters may be a better point to be labeled as home or work. Also, instead of just relying on the timestamp, perhaps I could investigate the pattern of travel throughout the day. People typically start their day at their home and then travel to work and then back home to end their day. I could label the

cluster that contains the data points where the user first opens the app in the day as home and the cluster to which they most often travel as work.

Another important note to make is that I have only used data from April for the results in this report. Including more location data for these users and incorporating other users could change the results without changing the algorithm. Eventually, I would like to see this algorithm applied to the whole dataset that is available. This way home and work locations of users that have not already saved this information can be inferred. This would be a more marketable process of the algorithm as Transit would be more interested in finding information from their data that is not already provided to them. In addition to this algorithm, I would really like to see this project expanded so that more complex machine learning is used on this dataset, using the saved home and work locations as training data.

Acknowledgments

This project has been done during the tenure of the RECSEM program of 2018. The program took place at The University of Tennessee. The program is funded by JICS, NSF, and UTK. Nothing could have been accomplished without the guidance of my mentors, Dr. Brakewood, Dr. Wong, and Dr. Liu.

References

[1] "Regions." *Transit*, https://transitapp.com/region. Accessed 31 July 2018.
[2] Transit. "Transit 4.0 Is Now Live." *Medium*, 20 Sept. 2016, https://medium.com/transit-app/transit-4-0-is-now-live-2329f60fb3bb. Accessed 31 July 2018.
[3] Brakewood, Candace. "Transit App Research Project." 4 June 2018. Presentation Accessed 31 July 2018.
[4] Wong, Kwai, et al., "Distributive Interoperable Exectuive Library (DIEL) for Systems of MultiPhysics Simulation." Accessed 31 July 2018.