SPATIO-TEMPORAL PATTERNS OF GPS TRAJECTORIES USING ASSOCIATION RULE MINING

by

VIVEK KUMAR SHARMA

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON
2016

To my father and my mother

who set the example and who made me who I am.

ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Ramez Elmasri for constantly motivating and encouraging me, and also for his invaluable advice during the course of my Master's studies. I wish to thank my academic advisors Dr. Leonidas Fegaras and Mr. David Levine for their interest in my research and for taking time to serve in my dissertation committee.

I would also like to extend my appreciation to the Computer Science and Engineering Department to support me financially in my Masters program. I am grateful to all the teachers who taught me during the years I spent in school, first in India, then in the United States.

I extend my gratitude to all my research mates including Mr. Upa Gupta, Mr. Mohammadhani Fouladgar, Mr. Neelabh Pant, Mr. Surya Swaminathan, Mr. Ahmed Ulde and everyone else whose support, encouragement and motivation helped me to complete my goals.

I am also extremely grateful to my father, mother and sisters for their sacrifice, encouragement and patience. I am extremely fortunate to be so blessed.I also thank several of my friends who have helped me throughout my career.

April 19, 2016

ABSTRACT

SPATIO-TEMPORAL PATTERNS OF GPS TRAJECTORIES USING ASSOCIATION RULE MINING

VIVEK KUMAR SHARMA, M.S.

The University of Texas at Arlington, 2016

Supervising Professor: Ramez A. Elmasri

The availability of location-tracking devices such as GPS,Cellular Networks and other devices provides the facility to log a person or device locations automatically. This creates spatio-temporal datasets of user's movement with features like latitude,longitude of a particular location on a specific day and time. With the help of these features different patterns of user movement can be collected,queues and analyzed.

In this research work, we are focused on user's movement patterns and frequent movements of users on a particular place,day or time interval. To achieve this we used Association Rule mining concept based on Apriori algorithm to find interesting movement patterns. Our dataset for this experiment is from Geolife project conducted by Microsoft Research Asia [1][2][3] which consist of 18,630 trajectories, 24 million points logged every 1-5 seconds or 5-10 meters per point.

First, we considered the spatial part of data; A two-dimensional space of (latitude,longitude) which ranges from minimum to maximum pair of latitude,longitude logged for all users. We distributed this space into equal grids along both dimen-

sions to reach a significant spatial distance range. Grids with high density pointsare sub-divided into further smaller grid cells.

For the temporal part of data; we transform the dates into days of the week to distinguish the patterns on a particular day and 12 time intervals of 2 hours each to split a day in order to distinguish peak hours of movement.

Finally we mine the data using association rules with attributes/features like user id, grid id (unique identifier for each spatial range/region of latitude and longitude), day and time. This enables us to discover patterns of user's frequent movement and similarly for a particular grid. This will give us a better recommendation based on the patterns for a set of like users, point of interests and time of day.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

The use of GPS (Global Positioning System) devices for accurate and efficient storage of locations and attributes has become a widely accepted method to collect Spatio-Temporal data. These types of datasets provide us with spatial features like latitude, longitude, altitude and temporal feature like date and time. These attributes enables us to find the different spatio-temporal patterns related to a spatial location on certain temporal factors and also to classify a set of objects who frequently travel to certain places.

In this thesis we are presenting an approach to find the intersting patterns between different attributes available to us through the spatio-temporal data collected using GPS devices [4]. The dataset we used for our experiments is part of Geolife, a project by Microsoft Research Asia [1][2][3]. This dataset covers 30 cities data where majority of data is present in Beijing, China. This dataset is spanned over 5 years for 182 users in total.

We first cleaned the data and considered only these attributes as per our experimental needs which are latitude, longitude, date and time. For storing the dataset we used SQLite database due to its fast processing time. After data cleaning is done we transformed some of the attributes so that he data mining results are more meaningful for example the date was converted into day of the week and time is transformed into time intervals of 2 hours each, which means 12 different time intervals for whole day. This allows the mining results to determine where certain person typically are during certain times of day (early morning, afternoon, night etc.), as well as discover

patterns based on weekdays and weekends. After the pre-processing of data we focus on visualizing the distribution of data among spatial and temporal features.

Initially we considered spatial aspect for which we created our problem space between minimum and maximum pair of latitude and longitude. Now we distributed this space among x and y axis to a range of cosiderable spatial distance/accuracy up to 20-25 meters. We plotted the points for each user in this grid spaces to analyze the movement.Initial distribution shows only 7.25 percent of grids with any movement of users but one of them has a dense point collection of 98.9 percent of users. So to achieve more spatially accurate results we divided this particular grid space further similarly to original problem space.

Then we analyzed the distribution for temporal features like day of the week and time of day intervals. When we plotted the data points they were quite evenly distributed among all the weekdays and a little higher on weekends especially Saturday. In case of time intervals there is a good distribution of data from morning to evening but drops later which seems intuitive as there are fewer data points as night approaches.

After the pre-processing of data we applied data-mining algorithm to find the association rules between users, spatial points, day and time. These rules can be interpreted as spatio-temporal patterns.

These patterns enable us to explore different areas like classifying location according to the frequent user movements or recommending points of interests based on the historical users movement pattern. These patterns can be used to get the traffic congesstions based on movements on certain time intervals of the day or to recommend route with less ETA (Expected Time of Arrival).

This work is organized as follows: Chapter 2 discuss about the overview of system used for the experiments and trajectory dataset used with analysis fo saptial

and temporal aspects. Chapter 3 describes the methodologies we used to process our dataset to be implemented on algorithm. Chapter 4 explains the algorithm results and our findings. In chapter 5 we listed out the conclusion and future work related to our experiments.

CHAPTER 2

OVERVIEW OF SYSTEM AND SPATIO-TEMPORAL DATA

2.1 Introduction

Spatio-temporal data has information about both space and time. Some common examples of this can be a moving object which can be at only one position at a given time but gradually changes its location over time. Basically spatio-temporal data is an extension of spatial data that includes the temporal aspect of a spatial object as it changes over time [5].

This type of data deals with the changing positon or shape of an object over time. For example points/locations recorded by GPS devices, in vehicles or personal cellphones are typical examples of spatio-temporal data. A sequence of those point over a certain period of time for a particular object is also known as *trajectory*.

A GPS trajectory can be represented in different formats but they all have a common thing which is sequence of time-stamped points each of which contains the information of latitude, longitude and in some cases altitude etc. The dataset we have used for our experiments is from Geolife project of Microsoft Research Asia [1][2][3] where they have collected GPS recorded points of different users over a period of few years. These points are collected by different GPS loggers, phones with GPS enabled feature. The sample rate for these devices are between 1-5 seconds or 5-10 meters per point.

This data is of 30 cities but the majority of data is for Bejing, China.

2.2 System Overview

For our experiments we have used Python [6] as the programming language because it provides a collection of quality libraries when dealing with large amount of data for analysis purpose. And to store the data we have used the SQLite database so that we don't have to retrieve data through raw data files everytime. The raw data in the files are preprocessed and stored in the SQLite relational database system. Most of the coding is done using Ipython notebook which is an interactive development environment for Python. It comes under the Anaconda- a package by Continnum [7] that includes most of the libraries we required for this analysis like Pandas, Numpy, Matplotlib etc.

The libraries used mostly for our experiments is Pandas which is an open-sorce data analysis library for Python and is implemented on C for fast execution purpose. It has a data structure named DataFrame similar to R language which works just like a relational data table with quick indexing and aggregation features. Next, Numpy which is very useful while working with multi-dimensional arrays as it has many mathematical operations to process the data for ease of analysis. Finally to visualize the data there are bunch of libraries avaialble, out of which we have used Matplotlib and Plotly for visulaizing the results.

All of the experiments were performed on a machine equipped with Windows 7 operating system, 12 GB of system memory, 500 GB of disk size and 2.8 GHz of clock rate.

2.3 Dataset Overview

As mentioned earlier the dataset we have used is from Geolife project of Microsoft Research Asia [1][2][3]. This data is collected from 182 users over the span

of five years i.e. April-2007 to August-2012. It contains 17,621 trajectories and 24,876,978 points including all users. This dataset has covered a wide range of users movement as a dense representation over Beijing city.

Latitude (in Decimal Degrees)	Longitude (in Decimal Degrees)	Default (All set to zero)	Altitude (in feet)	Number of days since 12/30/1899	Date (GMT)	Time (GMT)
40.008304	116.319876	0	492	39745.0902 662037	20081024	020959
40.008413	116.319962	0	491	39745.0903 240741	20081024	021004
40.007171	116.319458	0	46	39745.0903 819444	20081024	021009
40.007209	116.319484	0	48	39745.0904 398148	20081024	021014
40.007287	116.31959	0	41	39745.0904 976852	20081024	021019
40.007366	116.319727	0	40	39745.0905 555556	20081024	021024

Figure 2.1. Data Format of Geolife Dataset.

For every user there is a seprate directory/folder and within that multiple trajectory files for different days in PLT format. The format of data in each file is as follows (see Figure 2.1): line 1-line 6 can be ignored as they don't have data in them but some information related to GPS device. So from 7th line first field is Latitude then longitude both in decimal degrees format. Third is a default value which is set to zero for the whole datset. Next is altitude in feet, fifth field is called number of days and it is a number that represents the time that has passed since 12/30/1899 midnight, this number can be converted to the values in the next two fields and can be used if the tempral analysis is focused on a total order of time points. Sixth and Seventh fields are date and time respectively in GMT format.

2.4 Spatial Overview

A GPS trajectory is a collection of numerous points logged by the device during a journey/trip. Now these points are basically latitude and longitude coordi-

nates of that particular location. With this information we want to analyze the ratio/distribution of users among all data points.

So we are focused only on the spatial part of the data as of now. With the help of this spatial analysis we will be able to distinguish the users with varying data points logged by their device and how much an individual travels. We considered only latitude and longitude part of the data for each user and tried to visualize it. Below is the Figure 2.2 for number of data points for all users.

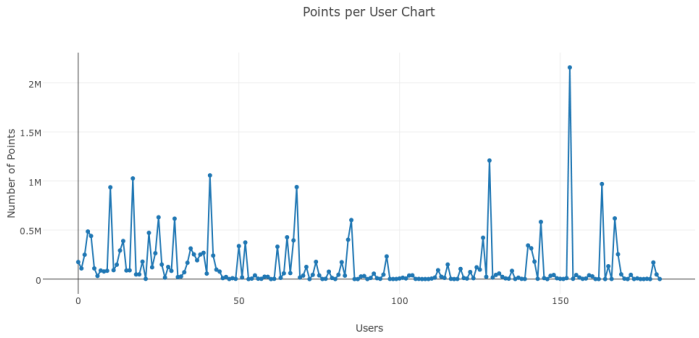

Figure 2.2. Distribution of Data points among users.

As we can see the distribution is not evenly spread, which means every individuals has their own travelling routines. From this plot, we can think of which users are travelling to the same places and have similar number of data points for a particular spatial location. This can be helpful to classify the cluster/group of users with similar travelling patterns. Another thought can be the spatial points with dense grouping to classify it as a intersting location or we can say it is a POI i.e. Point of Interest.

This spatial analysis provided us the different directions to work on with this dataset incorporating both users and spatial location's point of view. We will learn more about it as we progress towards our chapter related to the methodlogies used to get insight of spatial aspect of data.

2.5 Temporal Overview

A GPS trajectory as defined is a sequence of time-stamped points in this dataset. So as we are done with our spatial overview of data, we focus on the temporal part. In this dataset we have three fields related to temporal aspect which are number of days, date and time.

For our experiments we have considered date, time and did not use the other. Temporal analysis will help us to find those particular days and time intervals of day where a lot of activity/travelling is happening. This can be helpful to classify the locations in different categories.

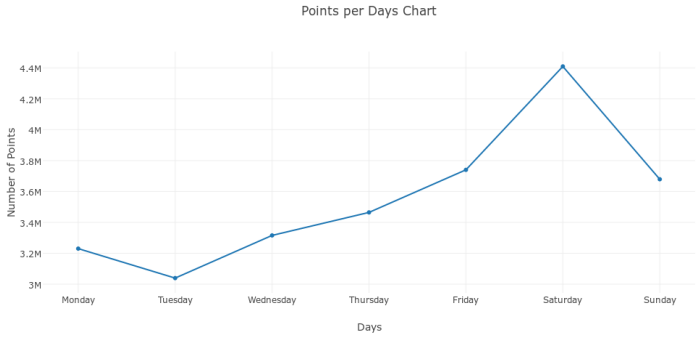

Figure 2.3. Distribution of points among different Days.

There is one more thing to notice from Figure 2.3 that on Saturdays more points had been recorded compared to other days. So being a weekend that particular spatial location where users are visiting can be classified as intersting location.

As we can see in Figure 2.4 data is quite evenly distributed for the days of week and in case of time intervals it goes to a peak around 08:00 am to 10:00 am and then dropping as the night approaches.

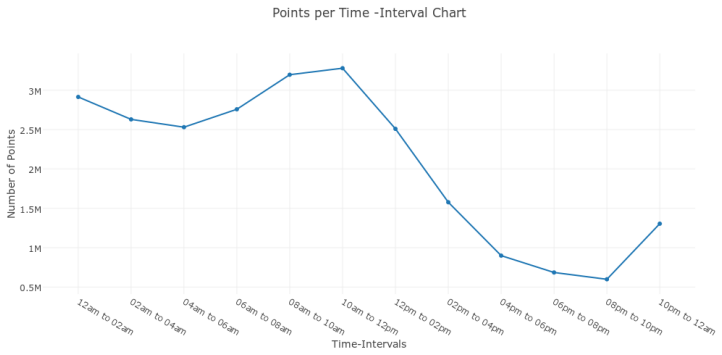

Figure 2.4. Distribution of points among different Time-Intervals.

2.6 Spatio-Temporal Overview

As we have done individual analysis of spatial and temporal aspect to get an insight of the data and to determine the probable useful information we can extract from each dimension with respect to users and spatial points. Now we have considered both spatial and temporal part together to see how it is distributed.

The plot below in Figure 2.5 is to show how data points are distributed over different users for all days of week. Similarly we can see how points are distributed for different time-intervals among users.

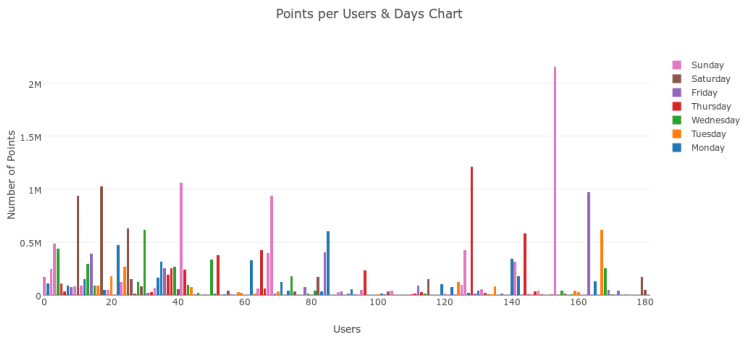

Figure 2.5. Distribution of points among different Days.

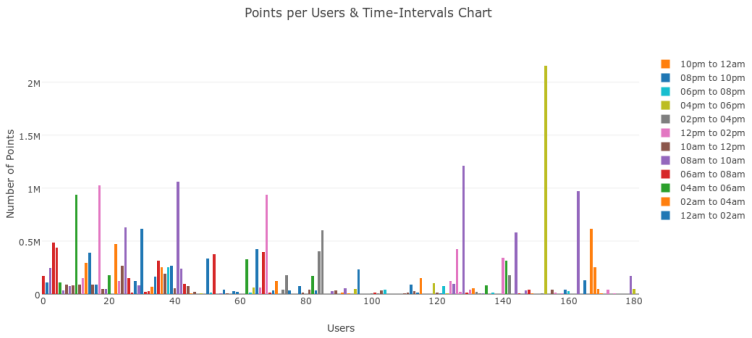

Figure 2.6. Distribution of points among different Time-Intervals.

As both spatial and temporal aspects are being visualized together, analysis can be focused on those spatial locations which have dense point repersentation with respect to the days of the week and time interval for that day. The latter is shown in Figure 2.6.

By considering both the aspects interesting patterns can be found, which we will be discussing in the later chapters.

2.7 Summary

In this chapter we gave an overview of the system we ran our experiments on and also the overview of dataset we used for our research work. As data is repersented for a dense region it is essential to analyze the distribution among various features available for this data, which might give us a direction to work on and as well eliminate the outliers from data.

CHAPTER 3

DATA PROCESSING AND GRID METHODOLOGY

3.1 Introduction

Data consist of useful information and to extract this information we have to pre-process the data in such a way so that extracted information becomes meaningful. In our research on spatio-temporal data we have features like coordinates, date and time. Now these features can be useful but to make more sense out of this information, we pre-processed it.

Pre-processing of data includes cleaning, transformation and many other steps to make data useful [8]. In our experiments we have done cleaning of data initially and then transforming date and time fields to concepts that are meaningful to our particular analysis. As this dataset includes GPS recorded logs of certain users so as to find their travelling patterns transforming date into the days of week makes much more sense to categorize patterns based on weekdays and weekends. Similarly in case of time, dividing it into 2 hours intervals enables us to get an insight of busy hours of the day.

Location accuracy is quite important when it comes to the GPS data but to predict an actual location is hard compared to predicting a region or range of locations due to limitation of GPS services. In order to implement this for our experiments we distributed the problem space equally on both x and y axis which created multiple rectangular-regions among the whole big problem space. The problem space was divided by a grid into neerly equal-sized grid cells.

A grid cell is basically a range of spatial coordinates which forms a rectanglar spatial region in two-dimensional plane of latitude, longitude. The purpose for making a grid cell is to simplify and group the travelling areas of users so that a significant spatial accuracy can be reached. We divided our problem space into grid cells to notice the user patterns and if dense grouping was found further divide that grid cell into smaller grid cells.

3.2 Data Processing

Data processing varies with the dataset being used [8]. In our experiments the dataset includes seprate folders for each user and within a user's folder there were multiple data files named on the date for which the positions were logged by GPS. In order to access data we have to follow this hierarchy of directory structure. Once we accessed the data file for a user there are few lines initially which just gives the information about the device; this is not relevant to our purpose. There is one thing to be taken care of is that these lines are not present for all users so we don't want to loose data for those users.

First we imported required software modules like Sqlite3, Datetime and OS which is being used for storing the data, track the execution time of the code and access the directory structure of current system respectively. Initially we defined our database and created a cursor which was used to acces database records and execute queries. Then we defined the path to the input file i.e. where whole dataset is located. Once the path is defined we looped through the root folder where all 182 user folders are located using OS module.

While pointer is on first user we looped again through all the files for that particular user and called our function defined to load data into the database. This function basically takes the input file's path as an argument and stores the data in

	usr_id	latitude	longitude	day	time
0	0	39.984702	116.318417	Thursday	02am to 04am
1	0	39.984683	116.318450	Thursday	02am to 04am
2	0	39.984686	116.318417	Thursday	02am to 04am
3	0	39.984688	116.318385	Thursday	02am to 04am
4	0	39.984655	116.318263	Thursday	02am to 04am

Figure 3.1. Data format after pre-processing.

'usr' table of SQLite database which has column latitude, longitude, date and time. While reading the contents of data file it checks each line if it had data in it or not to ignore unwanted content of the file.Similarly for all user's files this function was called repetitively and finally we have our 'usr' table with the whole data.

After creating our data table, we defined a dictionary, which is a data-structure in python based on key:value pairs. In our case key was the time-interval and value was the start time of that interval, for example 10pm to 12pm : 10:00:00. For converting dates into the days of week we used 'strftime' funtionality. Later we compared the time column of the table with the values of dictionary defined above and allotted the time-intervals accordingly.

After the whole process our data is stored in a table as shown in Figure 3.1 as compared to the initial dataset provided.

3.3 Grid Methodology

The problem space is divided into a grid of cells, where each grid cell is a rectangular region defined by spatial coordinates. While performing the spatial overview of data we noticed that user distribution was uneven among different grid cells, which means a broad spatial region is covered by GPS loggers/devices but much of the space is not used. As the spatial region becomes wider, our problem space also become bigger.

We defined our problem space with regards to the coordinates visited by the users. We queried our main data table to find the minimum and maximum numerical value of (latitude, longitude) pairs visited by any user. We defined our problem space based on these values as minimum pair as start and maximum pair as end. Figure 3.2 shown below is our problem space and we calculated havesine distance between these two pair of coordinates which comes out to be approximate 4350 km or 2703 miles. This distance is actually the diagonal length of the grid cell or problem space in this specific case.

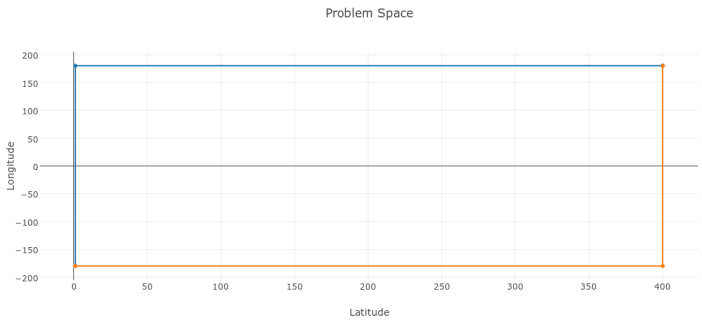

Figure 3.2. Problem Space.

This is a quite large spatial region to start with so we decided to distribute this area equally into 20 parts across both axis, which means 20 equal grid cells of latitude and longitude from min to max values as start and end points respectively. This enables us to have our problem space divided into 400 grid cells or equal regions.

Each grid cell is assigned a unique id and stored in the database as 'grd' table. After creating grid cells, haversine distance/diagonal length of each region is calculated to approximate 2793 km or 1735 miles. Figure 3.3 shows the grid distribution.

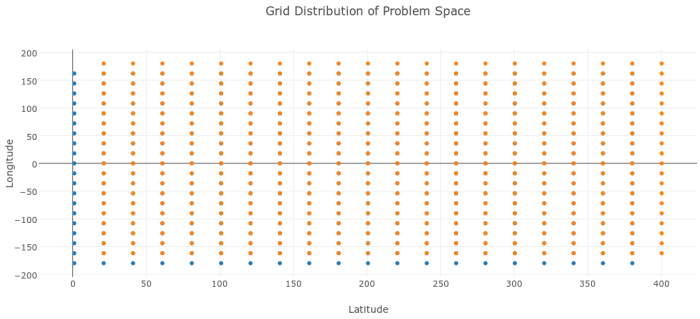

Figure 3.3. Grid distribution of problem space.

Now our area of concern was to gather those regions which have data points in them as the users hadn't visited many of the grid cells. This will help us in reducing the problem space by ignoring the grid cells where no user movement is there.

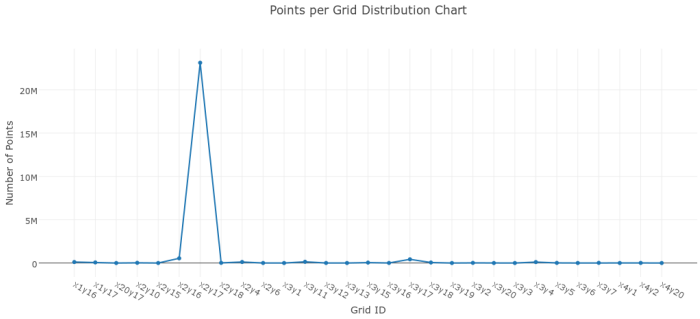

Figure 3.4. Grids with data points.

So, we plotted our data points against these regions by comparing each latitude, longitude point with the range of each region calculated earlier. Our results shows as in Figure 3.4 that only 29 grid cells had data points in them out of 400 which is roughly around 7 percent. This had reduced our problem space to only these grid cells. Figure 3.5 shows the grid points on map.

Figure 3.5. Grids with data points on map.

As we can see in Figure 3.4 there is one grid cell with almost every user visit in it. Our experiments hows that 98.9 percent of all users visited this grid space once. As mentioned in the dataset that data is being collected as dense representation and mainly over the Beijing city of China. So this grid space can be that region comprising of Bejing and its surrounding localities.

It confirmed that grid distribution had been computed correctly. For more confirmation we plotted this grid space on the map as in Figure 3.6.

Now we included the temporal part of data with the grid distribution to analyze the dataset with it. Figure 3.7 and 3.8 shows the distribution of data points among grids with respect to day and time respectively.

Now to get more spatially accurate results or in other words to define a close range/smaller region we considered this particular grid as our problem space for the next step. We repeated the same process of grid distribution similar to our initial problem space as second iteration of grid methodology.

After distributing the problem space into further grid cells of 20*20 equal distribution we plotted the users data points again. In this iteration our grid cell diagonal length is reduced to 135 km or 84 miles. In Figure 3.9 below is the distribution of data points among the grid cells. Figure 3.10 shows the grid points on a map.

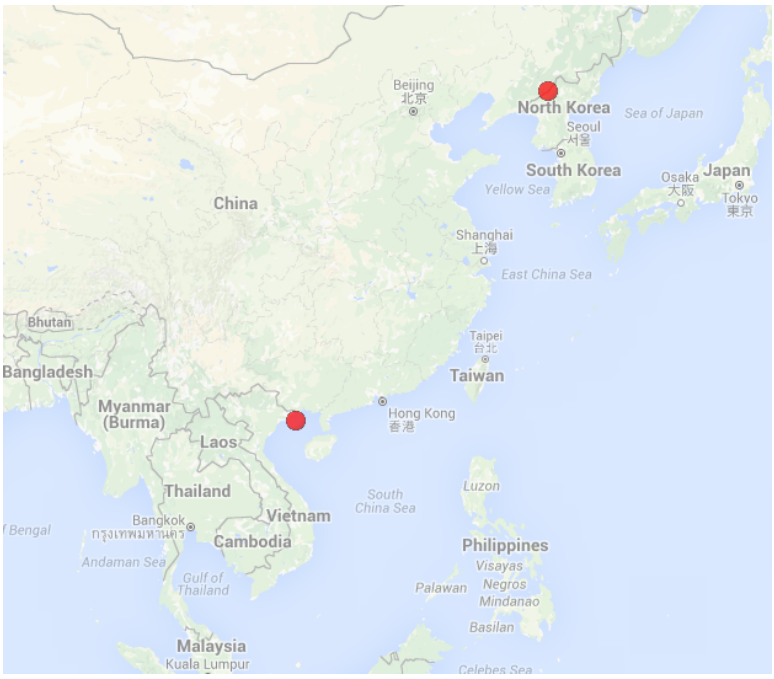

Figure 3.6. Grids with data points on map.

Temporal features for this grid distribution are shown in Figure 3.11 and 3.12 for day and time respectively.

After the second iteration we found that there were two grid cells with a majority of users movement in them. One more thing to be noticed here is that these two grids have their y coordinate common. Only difference is their x coordinates, which are also next to each other. So we decided to further divide the area between these two grids into 20*20 grids.

To confirm the grid spaces we plotted in Figure 3.13, these grid cells on the map. From this we found these grid locations covering the Beijing and surrounding provinces.

As the third iteration finishes we have more dense regions with data points in them. And the haversine distance is also computed to be approxiamtly 11 km or 6

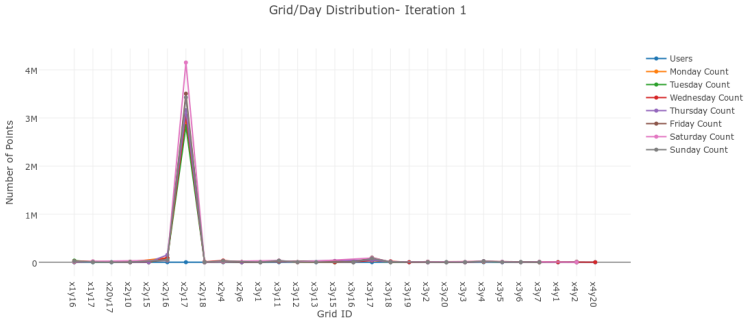

Figure 3.7. Grids distribution among Days of the week.

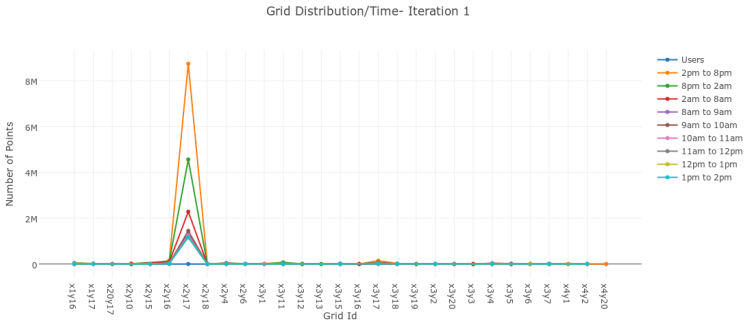

Figure 3.8. Grids distribution among Time Intervals.

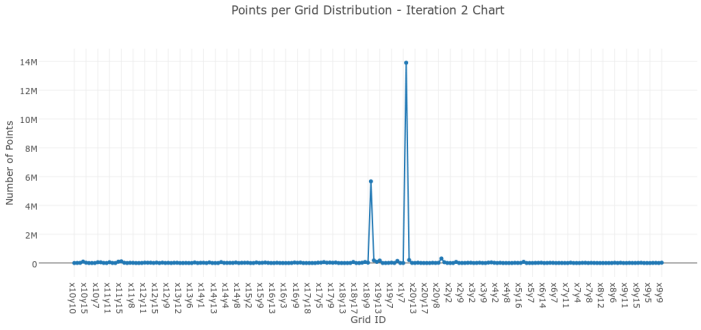

Figure 3.9. Grids with data points - Iteration 2.

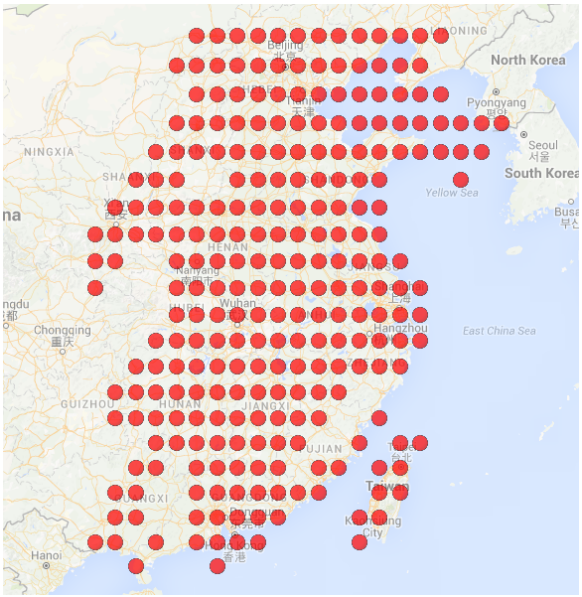

Figure 3.10. Grids with data points on map- Iteration 2.

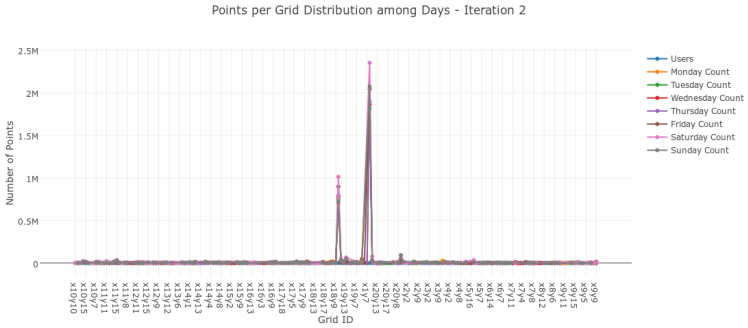

Figure 3.11. Grids distribution among Days of the week - Iteration 2.

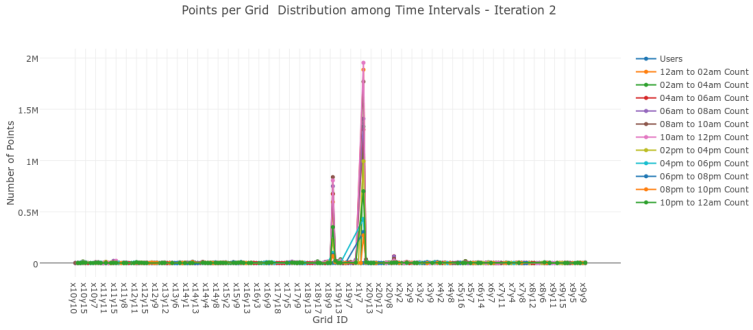

Figure 3.12. Grids distribution among Time Intervals - Iteration 2.

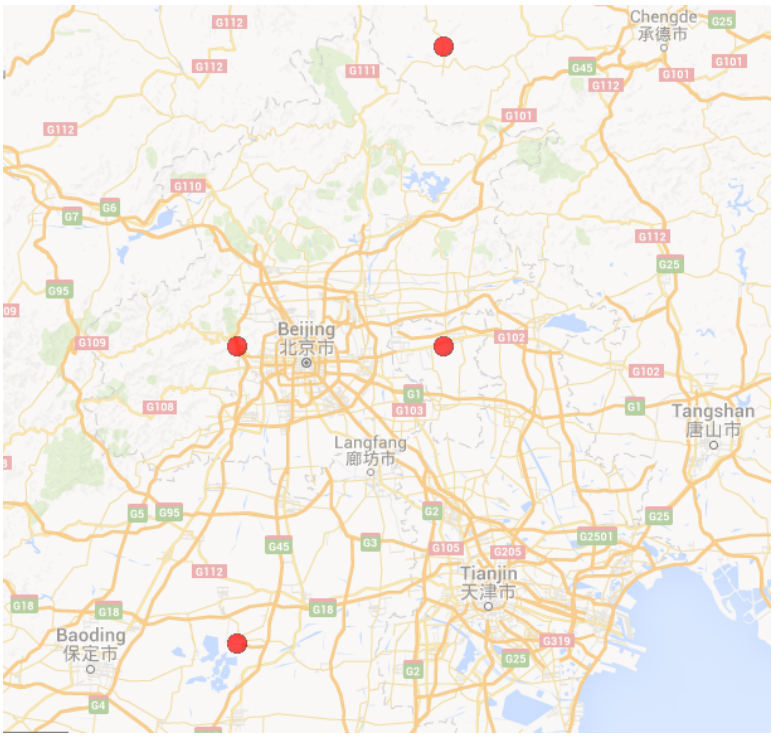

Figure 3.13. Grids with data points on map - Iteration 2.

miles. Figure 3.13 and 3.14 shows the representation of these grid spaces, on the map as well.

Temporal features for this grid distribution are shown in Figure 3.15 and 3.16 for day and time respectively.Figure 3.17 shows the grids with high density of data points on map after third iteration.

After the third iteration, the grid sapce covers around 6 miles. So we decided to further divide the grid space having dense data points within them. This iteration have reduced the grid space to 5 km or 3 miles. This much distance could be assumed as significant range of grid space keeping travelling patterns of users in consideration.

The temporal distribution among these grids after iteration-4 is shown in Figure 3.18 and 3.19 as below.

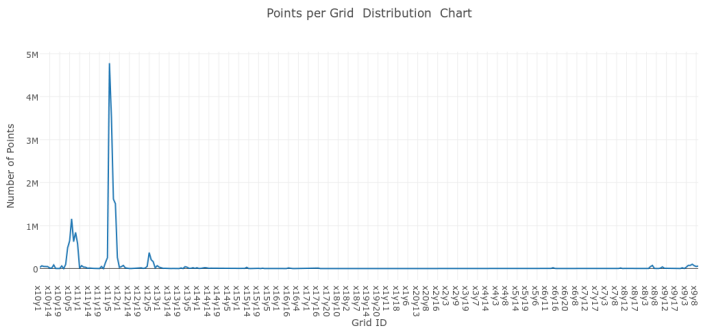

Figure 3.14. Grids with data points - Iteration 3.

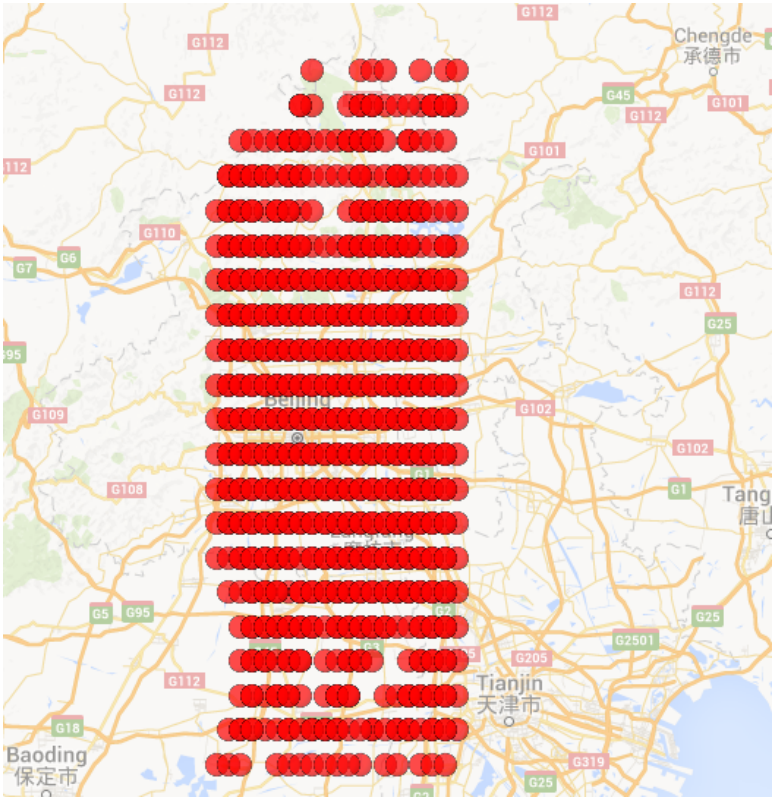

Figure 3.15. Grids with data points on map- Iteration 3.

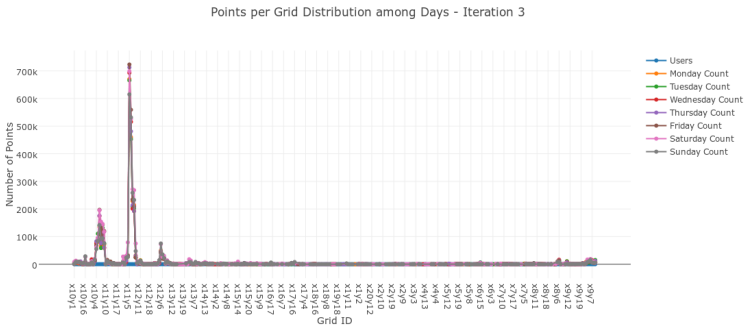

Figure 3.16. Grids distribution among Days of the week - Iteration 3.

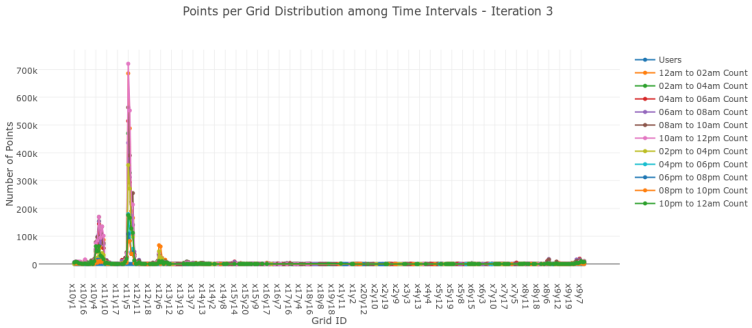

Figure 3.17. Grids distribution among Time Intervals - Iteration 3.

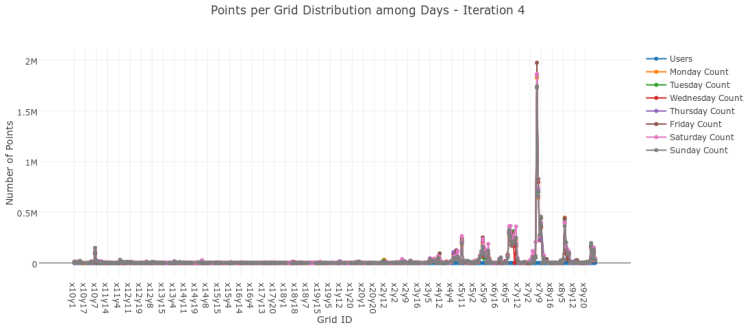

Figure 3.18. Grids distribution among Days of the week - Iteration 4.

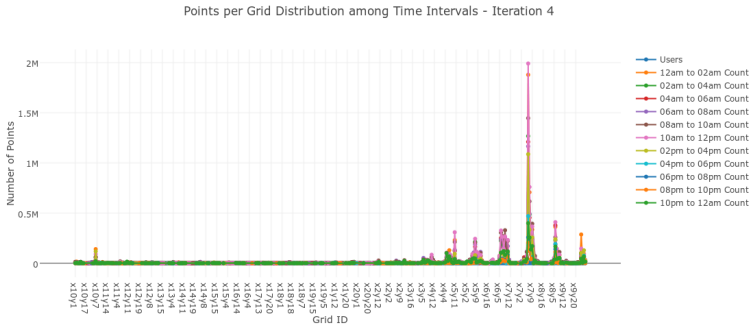

Figure 3.19. Grids distribution among Time Intervals - Iteration 4.

Once we were finsihed creating grids we assigned these grids an unique id so that it will increase the readability and also remove the redundancy of repeating grid ids. For first iteration grid ids starts from 1000 ans so on.., for second iteration from 2000 ans similarly 3000 and 4000 for third and fourth iteration respectively.

After assigning grid ids we had created a final data table with all of the iteration's data within it having changed grid ids. Now this data is ready to be run on algorithm.

3.4 Summary

In this chapter we discussed the various techniques used for processing the data for our experiments which include cleaning, transformation and feature extraction. In addition to this we also introduced the grid based structure to formulate our problem and reduce the area of concern step-by-step to acheive more spatially accurate results. By spatially accurate we means that to reduce the range of grid cells so that dense regions can be identified within a locality. Afterwards we also analyzed the temporal distribution of data points among grid cells to differentiate the patterns for both spatial and temporal dimensions.

CHAPTER 4

ASSOCIATION RULE MINING AND EXPERIMENTAL RESULTS

4.1 Introduction

Association rule learning is a data mining technique to discover interesting relationships between variables in a large database. The main purpose of this technique is to identify strong rules between variables based on some measures of interst. Based on the concept of strong rules, Rakesh Agarwal et al. introduced association rules for disovering relations between products in large-scale transaction data recorded by point-of-sale *(POS)* systems in supermarkets. That's why it is referred to as market-basket analysis as well. This analysis helped in placing items bought together near to each other and offering promotional prices for increasing sales [9].

In our experiments we have different variables like user id, grid id, day and time. Now running association rule mining on this dataset will provide us some intersting relationships between these variables which can be interpreted as spatio-temporal patterns. As our data is spatio-temporal so it will enable us to discover the rules between users travelling regions, days and time intervals. Similarly, in case of grids, which are basicaly a spatial region we can find patterns related to the busy days for this location and a particular time-interval when this place is busy.

There are many algorithms defined for finding association rules between variables in a large database. We have used a well-known algorithm known for association rule mining, Apriori algorithm. This algorithm is based on finding the frequent items in the database satisfying threshold value of *support* count and then finding associ-

ation rules between those itemsets based on another threshold known as *confidence*. The larger the value of support and confidence the stronger will be the rule.

We will be discussing the algorithm next and then the results we got from our experiments.

4.2 Apriori Algorithm

Apriori is an algorithm for frequent itemset mining and finding association rules in transactional databases. The frequent items found by this algorithm can be used to determine the association rules which highlights the trends in the database. This algorithm was purposed by Agarwal and Srikant in 1994. Each transaction is seen as a set of items, an itemset [10].

Apriori uses a bottom-up approach, where frequent item's subsets are extended one item at a time, which is also known as *candidate generation*. Then a group of candidates are tested against the data. It terminates when no successful extensions are found. Apriori uses *breadth-first search* and a *Hash-tree* structure to count candidate items efficiently.

4.2.1 Key Concepts

Support: The threshold value, which is the proportion of transactions in the database that contains the item-set.

Frequent Itemsets: The sets of item which has minimum support is known as frequent itemsets *(denoted by Li for ith-itemset).*

Apriori Property: Any subsets of frequent itemset must be frequent.

Join Operation: To find *Lk,* a set of candidate k-itemsets is generated by joining the set of *Lk-1* items with itself.

Confidence: For a rule *X–Y*, conf(X–Y) = Support(X union Y)/Support(X)

4.2.2 Apriori Algorithm in a Nutshell

- Find the frequent itemsets: the sets of items that have minimum support value.

- A subset of frequent itemset must also be a frequent itemset i.e. if AB is a frequent itemset, then both A and B should be a frequent itemset.

- Find frequent itemset with cardinality from *1 to k (k-itemset)* iteratively.

- Use the frequent itemsets to generate association rules.

4.2.3 Psuedo Code

Join Step: *Ck* is generated by joining *Lk-1* with itself.

Prune Step: Any *k-1* itemset that is not frequent can not be a subset of a frequent *k-itemset.*

Algorithm:

Ck: Set of candidate itemsets of size k

Lk: Set of frequent itemsets of size k

L1= Set of frequent itemsets of size 1;

for (k=1,Lk!=0;k++) do begin

Ck+1 = candidates generated from Lk generated by self-join on Lk;

for each transaction t in database do

increment the count of all candidates in Ck+1 that are contained in t

Lk+1 = candidates in Ck+1 with min-support

end

return Union of Lk;

For each frequent itemset "l", generate all non-empty subsets of "l"

For every nonempty subset "s" of "l", output the rule "s–(l-s)" if support-count(l)/support-count(s) greater than or equal to the min-conf where min-conf is minimum confidence threshold;

4.3 Implementation

For our experiments we implemented Apriori algorithm in Python using these libraries available, *Itertools* for combining itemsets and *Collections* for using *default-dict* dictionary to count number of occurences of an item. We also used *Frozenset* data structure massively for the implementation of this algorithm. Basically this data structure provides all the feature of a normal set data structure except that it can't be changed once created i.e. they are immutable.

First, we defined a function to read data from database and create frozenset of single items in each transaction of database. Then we call another function with argument as these frozenset of one items and calculate support of each item and return itemset who had support count above threshold value. After performing multiple experiments we found a good support threshold to be 0.0001 for our dataset i.e. one record in thousand transactions. The reason behind the low threshold value may be due to the dense regions covered by each grid cell, because of which user's movement was recorded in different grid cells.

After getting the frequent one-itemset we run the loop untill there is no frequent itemset found and in each iteration different candidate sets being generated. In our case we have four different variables/items i.e. User, Grid, Day and Time. So at most four-itemset will be generated and out of all those the ones that are above support threshold will form frequent itemsets. Once we have all the frequent itemsets, associations between these items will be calculated based on another threshold, *confidence*.

We kept confidence threshold to be 80 percent as per the definition of strong rules, which says that higher the confidence, the stronger the rule will be. Now association rules will be genrated based on this threshold between different frequent itemsets. We categorize these rules to be interpreted as spatio-temporal patterns, which will enable us to discover the patterns of users and patterns of a particular spatial region at a specific day of the week and time interval of the day.

4.4 Experimental Results

We mentioned earlier in this chapter the threshold values required by the algorithm, which are *Support* and *Confidence*. So, minimum support count for our experiment is *0.001* and confidence is considered to be *80 percent*. The reason for low support threshold is due to inconsistent data for all users as well as smaller grid cell range as iterations increases.

We have also changed the unique id's given to grid cells for more readability i.e. For initial grid cells it starts from 1000, for second iteration starts from 2000, for third iteration from 3000 and 4000 for fourth iteration. After getting the rules these grid cell ids can be interpreted on the map to visualize the results.

With these threshold values we got 697 *1-frequent itemsets*, 7744 *2-frequent itemsets*, 8961 *3-frequent itemsets* and 230 rules. These rules were interpreted on the map to verfiy our results. The most common patterns we found are listed below with the example rules generated through our experiments.

Rule Pattern 1: Grid id,Day –>User id A location on particular day will have this specific user. e.g. *[4192, Friday –>39]*, *[2041, Friday –>128]*, *[2085, Tuesday –>153]*

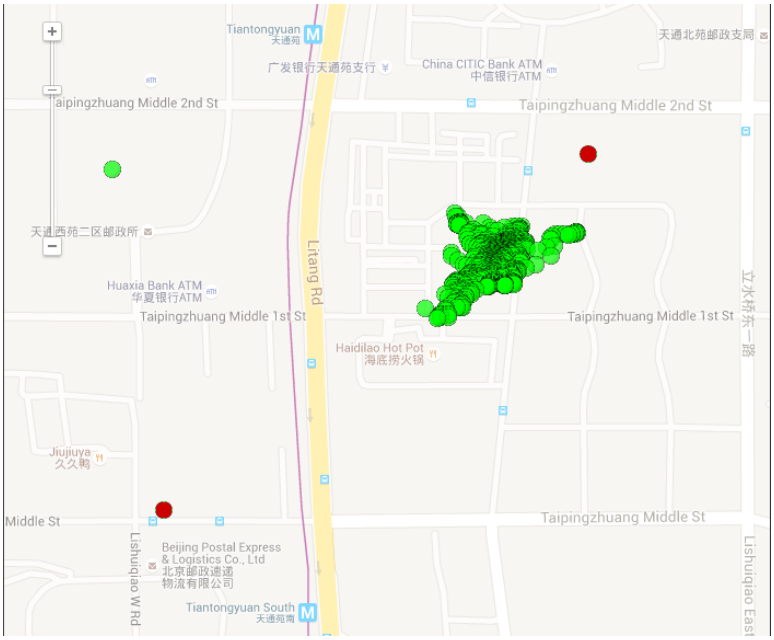

Figure 4.1. Interpretation of Rule 1.

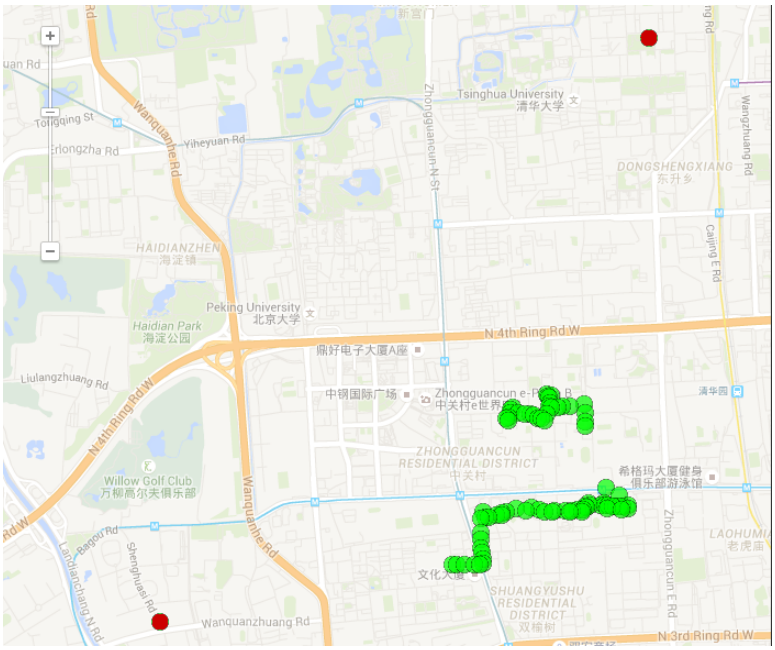

Figure 4.2. Interpretation of Rule 2.

Figure 4.3. Interpretation of Rule 3.

Rule Pattern 2: Grid id,Time-Interval –>User id A location on particular time interval will have this specific user. e.g. *[4192, 02pm to 04pm –>39]*, *[4081, 08pm to 10pm –>115]*, *[2041, 12am to 02am –>128]*

In Figure 4.1, we plotted two rule patterns stated above which says that if grid cell id is 4192, day is Friday or time-interval is 2pm to 4pm then it will be user id 39.

Now to verfiy we plotted all the points for user id 39 for Friday during time interval 2pm to 4pm and they all fall in grid cell *(marked in red)* defined by us which confirms the pattern.

Rule Pattern 3: User id,Day –>Grid id A user on particular day travel to this specific location. e.g. *[97, Monday –>4131]*, *[118, Saturday –>1020]*, *[132, Wednesday –>2076]*

Rule Pattern 4: User id,Time-Interval –>Grid id A user on particular time-interval travel to this specifc location. e.g. *[97, 08am to 10am –>4131]*, *[150, 02pm to 04pm –>4132]*, *[134, 02pm to 04pm –>4131]*

Let's take another set of rules which states that if day is Monday, user id 97 or time-interval is 8am to 10am then it will be in grid cell id 4131. In Figure 4.2 shows

the grid cell range marked in red and all the points for user id 97 for Monday and time-interval 8am to 10am.

These plots enable us to confirm the authenticity of rules generated with higher confidence threshold value. In addition to this we can also classify certain grid cell ranges based on these rules. Like in Figure 4.3 which shows the grid cell 4212 with all the points for user id 13 for weekdays during time interval 10pm to 12am and 12am to 2am.

After reviewing the points as per this rule, majority of the points are located nearby to a museum which is helpful to classify this grid cell and recommend similar locations to this user based on his patterns. Few more patterns are classified based on the rules.

Rule Pattern 5: User id,Grid id –>Day A user in a particular location will be on specific day. e.g. *[24, 3079 –>Tuesday]*, *[65, 2195 –>Saturday]*, *[5, 2005 –>Sunday]*

Rule Pattern 6: User id,Grid id –>Time-Interval A user in a particular location will be on specific time-interval. e.g. *[41, 4141 –>12am to 02am]*, *[41, 4162 –>12am to 02am]*, *[29, 4022 –>08am to 10am]*

Rule Pattern 7: User id –>Grid id A user will always visit a particular location. e.g. *[87 –>4131]*, *[97 –>4131]*, *[41 –>4308]*

Rule Pattern 8: Grid id –>User id A particular location will always be visited by this user. e.g. *[2151 –>144]*, *[3000 –>30]*, *[4323 –>128]*

Rule Pattern 9: User id, Day –>Time-Interval A user on a specifc day travels at a specific time-interval. e.g. *[124, Sunday –>10pm to 12am]*, *[149, 06pm to 08pm –>Saturday]*, *[147, 02pm to 04pm –>Sunday]*

Rule Pattern 10: User id, Time-Interval –>Day A user on particular time-interval travels on a specific day. e.g. *[48, 12pm to 02pm –>Wednesday]*, *[46, 04am to 06am –>Saturday]*, *[110, 08am to 10am –>Friday]*

Rule Pattern 11: User id –>Day A user always travels on a specifc day. e.g. *[49 –>Wednesday]*, *[151 –>Sunday]*, *[149 –>Saturday]*

Now these rules can be categorized based on the variables used to generate rules like day, time-interval, grid cells or users. In addition to this, these rules can be sorted based on confidence threshold value to get rules with varying confidence above threshold value of 80%. It will enable us to focus on strongest rules first, which has confidence value of 100% and then coming down till 80%.

Rule Pattern 12: Grid id –>Day A specific location which is visited by users on weekdays or weekends. e.g. *[4039 –>Saturday]*, *[2178 –>Sunday]*, *[2023 –>Monday]*

Rule Pattern 13: User id –>Day A user who always travels on weekends. e.g. *[177 –>Saturday]*, *[143 –>Saturday]*, *[151 –>Sunday]*

Rule Pattern 14: Time-Interval, Day –>User id A set of users who always travels between time-interval 08am to 10am. e.g. *[08am to 10am, Friday –>110]*, *[08am to 10am, 4022 –>29]*, *[08am to 10am, 4131 –>97]*

Rule Pattern 15: Grid id, Day/Time-Interval –>User id A user travels on different days visit a specifc location at a particular time-interval. e.g. *[2083, Monday –>51]*, *[2083 –>51]*, *[2083, 08pm to 10pm –>51]*, *[Tuesday, 2083 –>51]*

These are the few patterns generated from the rules.All of the 230 rules are specified in *appendix D* later.

4.5 Summary

In this chapter we described the algorithm we used to find the patterns in our data with the help of association rule learning technique. We discussed algorithm

requirements and determining the threshold values for our experiments. Moreover, indepth working/implementation of the algorithm had been also dicussed. Finally, we showed the findings for our experiments and the way to interpret those findings to build reccomendation system around it.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this thesis we ran our experiments on GPS data to find some intersting patterns based on users travelling history. We found many rules which can be interpreted to get some meaningful information out of them. Basically all the grid ids we have in our rules can be interpreted to specific location on map, which will help in classifying that location. In addition to this, rules which have temporal features as well will help in determining the busy days and time interval for the day to that specific location.

Similarly, users travel patterns can also be interpreted based on these rules. Like a set of users travels specifically between 8am to 10am on weekdays i.e. Monday to Friday. And on saturdays these set of users travel to a specific grid cell, which actually is a spatial region that can be classified.

Finally on concluding note we can say that with the help of these rules we were able to extract some meaningful information from them on spatio-temporal aspect. These rules also provides us the behavioral patterns of users which can be further explored.

5.2 Future Work

In future this work can be extended for buliding recommendation system based on different parameters.Like locations can be recommended to users based on these patterns which will be helpful to provide them a set of options based on their travelling regions. For example if users is travelling to a natural park on weekends then similar

locations can be recommended nearby to users travelling regions. Similarly for a location, once it is classified peak hours of a day can be recommended. For example if a location is classified as a restaurant then on which day or on what hours of a day it is busy can be determined and based on that it can be recommended like for lunch, dinner etc.

Distributed implementation of this research work can also be done to reduce the time complexity of the problem. These results can be considered as training set for further study of patterns using supervised learning techniques. There is a possibility of building an application where user can import their travel history to get suitable results.

In addition to this as dataset is provided with transportation modes used by users during their journey. This can be used to find traffic congestions as lot of users travel on a certain time-interval of a day. Also effective journey time can be estimated based on this analysis.

APPENDIX A

PROGRAM TO PROCESS AND STORE DATA IN DATABASE

In this appendix, we present a Python program used for our experiments to pre-process the data and storing it in SQLite datbase.

/*Importing required modules*/

import os

import sys

import sqlite3

from datetime import datetime

start-time = datetime.now()

/*Creating database*/

conn = sqlite3.connect('MyDB.db')

/*Creating cursor to access database*/

c = conn.cursor()

path="input path to the dataset's root folder"

dirlist=os.listdir(path)

/*Defining time intervals as a Dictionary*/

tm-interval='12am-to-2am':'02:00:00','2am-to-4am':'04:00:00','4am-to-6am':'06:00:00','6am-to-8am':'08:00:00','8am-to-10am':'10:00:00','10am-to-12pm':'12:00:00','12pm-to-2pm':'14:00:00','2pm-to-4pm':'16:00:00','4pm-to-6pm':'18:00:00','6pm-to-8pm':'20:00:00','8pm-to-10pm':'22:00:00','10pm-to-12am':'24:00:00'

/*Creating our main table*/

c.execute('CREATE TABLE usr (usr-id INT,latitude REAL,longitude REAL,day TEXT,time TEXT)')

/*Function to insert data into database*/

def db-access(inpt):

with open(inpt, 'r') as fileobj:

for line in fileobj:

l1=line.split(',')

/*Condition to ignore unwanted data*/

if len(l1)==7:

/*Converting date into days*/

l1[5]=datetime.strptime(str(l1[5]).replace('-',''),'%Y%m%d').strftime('%A')

/*Converting time into time-intervals*/

l1[6]=datetime.strptime(str(l1[6]).replace(':','').strip(),'%H%M%S').strftime('%H:%M:%S')

if '00:00:00'¡=l1[6]¡=tm-interval['12am-to-2am']:

l1[6]='12am to 02am'

elif tm-interval['12am-to-2am']¡=l1[6]¡=tm-interval['2am-to-4am']:

l1[6]='02am to 04am'

elif tm-interval['2am-to-4am']¡=l1[6]¡=tm-interval['4am-to-6am']:

l1[6]='04am to 06am'

elif tm-interval['4am-to-6am']¡=l1[6]¡=tm-interval['6am-to-8am']:

l1[6]='06am to 08am'

elif tm-interval['6am-to-8am']¡=l1[6]¡=tm-interval['8am-to-10am']:

l1[6]='08am to 10am'

elif tm-interval['8am-to-10am']¡=l1[6]¡=tm-interval['10am-to-12pm']:

l1[6]='10am to 12pm'

elif tm-interval['10am-to-12pm']¡=l1[6]¡=tm-interval['12pm-to-2pm']:

l1[6]='12pm to 02pm'

elif tm-interval['12pm-to-2pm']¡=l1[6]¡=tm-interval['2pm-to-4pm']:

l1[6]='02pm to 04pm'

elif tm-interval['2pm-to-4pm']¡=l1[6]¡=tm-interval['4pm-to-6pm']:

l1[6]='04pm to 06pm'

elif tm-interval['4pm-to-6pm']¡=l1[6]¡=tm-interval['6pm-to-8pm']:

l1[6]='06pm to 08pm'

elif tm-interval['6pm-to-8pm']¡=l1[6]¡=tm-interval['8pm-to-10pm']:

l1[6]='08pm to 10pm'

elif tm-interval['8pm-to-10pm']¡=l1[6]¡=tm-interval['10pm-to-12am']:

l1[6]='10pm to 12am'

/*Inserting into database*/

c.execute("INSERT INTO usr VALUES(?,?,?,?,?)",(inpt[53:56],l1[0],l1[1],l1[5],l1[6]))

/*Loop for root folder*/

for dirs in dirlist:

/*Getting path to each user's folder*/

inpt=os.path.join(path,dirs)

dirlist1=os.listdir(inpt)

/*Loop for user's folder*/

for dirs1 in dirlist1:

inpt1=inpt+'\\'+dirs1

/*Excluding any text files*/

if '.txt' in dirs1:

continue

else:

/*Getting path to trajectory files inside user's folder*/

dirlist2=os.listdir(inpt1)

/*Loop for accessing multiple trajectory files*/

for dirs2 in dirlist2:

/*Getting only trajectory files*/

if '.plt' in dirs2:

inpt2=inpt1+'\\'+dirs2

/*Calling function defined above*/

db-access(inpt2)

else:

continue

conn.commit()

print datetime.now()-start-time

APPENDIX B

PROGRAM TO CREATE GRIDS IN MULTIPLE ITERATIONS

In this appendix, we present a Python program used for our experiments to create grids and then comparing data points to those grids in multiple iterations.

import sqlite3

import numpy as np

from datetime import datetime

start-time = datetime.now()

conn = sqlite3.connect('MyDB.db')

c = conn.cursor()

/*Creating table for stroing grid cells with their id's*/

c.execute('CREATE TABLE grd (grd-id INT,min-lat REAL,min-lon REAL,max-lat REAL,max-lon REAL)')

/*Creating table for stroing the results of comparison of data points with grid cells*/

c.execute('CREATE TABLE usr-grd (usr-id INT,grd-id INT,min-lat REAL,min-lon REAL,max-lat REAL,max-lon REAL,day TEXT,time TEXT)')

/*Function for creating grid cells*/

def grids(lat-grid,lon-grid,min-lat,min-lon):

xcount=0

for i in x-grid[1:]:

xcount+=1

ycount=0

if xcount==1:

min-lat=lat1

for j in y-grid[1:]:

ycount+=1

if ycount==1:

min-lon=lon1

g-id="x"+str(xcount)+"y"+str(ycount)

c.execute("INSERT INTO grd VALUES(?,?,?,?,?)",(g-id,min-lat,min-lon,i,j))

min-lon=j

else:

g-id="x"+str(xcount)+"y"+str(ycount)

c.execute("INSERT INTO grd VALUES(?,?,?,?,?)",(g-id,min-lat,min-lon,i,j))

min-lon=j

min-lat=i

else:

for j in y-grid[1:]:

ycount+=1

if ycount==1:

min-lon=lon1

g-id="x"+str(xcount)+"y"+str(ycount)

c.execute("INSERT INTO grd VALUES(?,?,?,?,?)",(g-id,min-lat,min-lon,i,j))

min-lon=j

else:

g-id="x"+str(xcount)+"y"+str(ycount)

c.execute("INSERT INTO grd VALUES(?,?,?,?,?)",(g-id,min-lat,min-lon,i,j))

min-lon=j

min-lat=i

c.execute('SELECT min(latitude),min(longitude),max(latitude),max(longitude) FROM usr')

all-rows=c.fetchone()

/*Fetching minimum and maximum pair of latitude and longitude*/

lat1=all-rows[0]

lat2=all-rows[2]

lon1=all-rows[1]

lon2=all-rows[3]

/*20 equal distribution among min and max pair*/

x-grid=np.linspace(lat1,lat2,21)

y-grid=np.linspace(lon1,lon2,21)

grids(x-grid,y-grid,lat1,lon1)

*c.execute('SELECT * FROM usr')*

all-rows=c.fetchall()

*c.execute('SELECT * FROM grd')*

all-row=c.fetchall()

/*Comparing all data points with grid cells*/

for i in all-rows:

for j in all-row:

if j[1]¡=i[1]¡=j[3] and j[2]¡=i[2]¡=j[4]:

c.execute("INSERT or IGNORE INTO usr-grd VALUES(?,?,?,?,?,?,?,?)",(i[0],j[0],j[1],j[2],j[3],j[

else:

continue

conn.commit()

print datetime.now()-start-time

APPENDIX C

PROGRAM TO CREATE ASSOCIATION RULES

In this appendix, we present a Python program having implementation of Apriori algorithm to genrate association rules among variables present in the database.[11]

import sqlite3

import pandas as pd

from itertools import chain, combinations

from collections import defaultdict

conn = sqlite3.connect('MyDB.db')

c = conn.cursor()

/*File for storing rules generated*/

log=open('rules.txt','w')

/*Getting data from database*/

df=pd.read-sql-query('SELECT usr-id,grd-id,day,time from usr-data group by usr-id,grd-id,day,time',conn)

def subsets(arr):

/* Returns non empty subsets of arr*/

return chain([combinations(arr, i + 1) for i, a in enumerate(arr)])*

/*Function for getting frequent itemsets*/

def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):

/*calculates the support for items in the itemSet and returns a subset of the itemSet each of whose elements satisfies the minimum support*/

-itemSet = set()

localSet = defaultdict(int)

for item in itemSet:

for transaction in transactionList:

if item.issubset(transaction):

freqSet[item] += 1

localSet[item] += 1

for item, count in localSet.items():

support = float(count)/len(transactionList)

if support ¿= minSupport:

-itemSet.add(item)

return -itemSet

def joinSet(itemSet, length):

/*Join a set with itself and returns the n-element itemsets/*

return set([i.union(j) for i in itemSet for j in itemSet if len(i.union(j)) == length])

def getItemSetTransactionList(data-iterator):

transactionList = list()

itemSet = set()

for record in data-iterator:

transaction = frozenset(record)

transactionList.append(transaction)

for item in transaction:

/*Generate 1-itemSets*/

itemSet.add(frozenset([item]))

return itemSet, transactionList

def runApriori(data-iter, minSupport, minConfidence):

/*run the apriori algorithm. data-iter is a record iterator Return both: - items (tuple, support)- rules ((pretuple, posttuple), confidence)*/

itemSet, transactionList = getItemSetTransactionList(data-iter)

freqSet = defaultdict(int)

largeSet = dict()

/*Global dictionary which stores (key=n-itemSets,value=support) which satisfy minSupport*/

assocRules = dict()

/*Dictionary which stores Association Rules*/

oneCSet = returnItemsWithMinSupport(itemSet,transactionList,minSupport,freqSet)

currentLSet = oneCSet

k = 2

while(currentLSet != set([])):

largeSet[k-1] = currentLSet

currentLSet = joinSet(currentLSet, k)

currentCSet = returnItemsWithMinSupport(currentLSet,transactionList,minSupport,freqSet)

currentLSet = currentCSet

k = k + 1

def getSupport(item):

/*local function which Returns the support of an item*/

return float(freqSet[item])/len(transactionList)

toRetItems = []

for key, value in largeSet.items():

toRetItems.extend([(tuple(item), getSupport(item)) for item in value])

toRetRules = []

for key, value in largeSet.items()[1:]:

for item in value:

-subsets = map(frozenset, [x for x in subsets(item)])

for element in -subsets:

remain = item.difference(element)

if len(remain) ¿ 0:

confidence = getSupport(item)/getSupport(element)

if confidence ¿= minConfidence:

toRetRules.append(((tuple(element), tuple(remain)),confidence))

return toRetItems, toRetRules

def printResults(items, rules):

/*prints the generated itemsets sorted by support and the confidence rules sorted by confidence*/

for item, support in sorted(items, key=lambda (item, support): support):

print¿¿log, "item: %s , %.3f" % (str(item), support)

print¿¿log, "\n———————————— RULES:"

for rule, confidence in sorted(rules, key=lambda (rule, confidence): confidence):

pre, post = rule

print¿¿log, "Rule: %s == %s , %.3f" % (str(pre), str(post), confidence)

def dataFromFile():

/*Function which reads from the file and yields a generator/*

for i,j in df.iterrows():

record = frozenset([str(j['usr-id']).replace('L',''),str(j['grd-id']).replace('L',''),str(j['day']).replace(

yield record

if –name– == "–main–":

inFile=dataFromFile()

minSupport = 0.0001

minConfidence = 0.8

items, rules = runApriori(inFile, minSupport, minConfidence)

printResults(items, rules)

log.close()

APPENDIX D

ASSOCIATION RULES

In this appendix, we listed out all the 230 rules we got from our experiments using threshold value of support as 0.0001 and confidence as 80%. Also as there are grid id's used for increasing readability, co-ordinates of particular grid cell in a rule are also mentioned explicitly.

These rules are sorted based on confidence value in decreasing order and 1000-4000 depicts grid cell id's, 0-182 for user id's.

Rule: ('87') == ('4131') , **1.000** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('118') == ('1020') , **1.000** *40.956288266666704, 108.00363462000004, 60.91242040000005, 126.00196136500003*

Rule: ('160') == ('1000') , **1.000** *1.044024, 90.00530787500006, 21.000156133333352, 108.00363462000004*

Rule: ('2002') == ('23') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('143') == ('Saturday') , **1.000**

Rule: ('1013') == ('163') , **1.000** *40.956288266666704, -89.977959575, 60.91242040000005, -71.97963282999999*

Rule: ('2004') == ('128') , **1.000** *21.99796274000002, 112.50321630625004, 22.995769346666687, 113.40313264350004*

Rule: ('1003') == ('163') , **1.000** *21.000156133333352, -89.977959575, 40.956288266666704, -71.97963282999999*

Rule: ('151') == ('Sunday') , **1.000**

Rule: ('2041') == ('128') , **1.000** *25.98918916666669, 116.10288165525003, 26.986995773333355, 117.00279799250004*

Rule: ('2028') == ('125') , **1.000** *24.991382560000023, 112.50321630625004, 25.98918916666669, 113.40313264350004*

Rule: ('177') == ('Saturday') , **1.000**

Rule: ('1005') == ('10') , **1.000** *21.000156133333352, 72.00698113000004, 40.956288266666704, 90.00530787500006*

Rule: ('2151') == ('144') , **1.000** *35.96725523333336, 119.70254700425004, 36.96506184000003, 120.60246334150003*

Rule: ('149') == ('Saturday') , **1.000**

Rule: ('49') == ('Wednesday') , **1.000**

Rule: ('2023') == ('Monday') , **1.000** *23.993575953333355, 117.00279799250004, 24.991382560000023, 117.90271432975004*

Rule: ('2178') == ('Sunday') , **1.000** *38.96067505333337, 112.50321630625004, 39.95848166000003, 113.40313264350004*

Rule: ('4308') == ('41') , **1.000** *40.31769203840003, 116.48534609858129, 40.35261526963337, 116.5235925429144*

Rule: ('4081', '08pm to 10pm') == ('115') , **1.000** *39.89861326360004, 116.60008543158065, 39.93353649483337, 116.63833187591378*

Rule: ('3078', '24') == ('Tuesday') , **1.000** *39.459578356666704, 116.68782727446253, 39.55935901733337, 116.73282309132503*

Rule: ('Friday', '2041') == ('128') , **1.000** *25.98918916666669, 116.10288165525003, 26.986995773333355, 117.00279799250004*

Rule: ('67', '4133') == ('Saturday') , **1.000** *39.968459726066705, 116.37060676558191, 40.00338295730004, 116.40885320991504*

Rule: ('50', '4081') == ('Friday') , **1.000** *39.8986132636004, 116.60008543158065, 39.93353649483337, 116.63833187591378*

Rule: ('2002', 'Thursday') == ('23') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('Tuesday', '2085') == ('153') , **1.000** *29.980415593333362, 119.70254700425004, 30.978222200000026, 120.60246334150003*

Rule: ('4170', '78') == ('Sunday') , **1.000** *40.03830618853337, 116.29411387691566, 40.073229419766704, 116.33236032124879*

Rule: ('08am to 10am', '149') == ('Saturday') , **1.000**

Rule: ('160', 'Monday') == ('1000') , **1.000** *1.044024, 90.00530787500006, 21.000156133333352, 108.00363462000004*

Rule: ('124', 'Thursday') == ('1006') , **1.000** *21.000156133333352, 90.00530787500006, 40.956288266666704, 108.00363462000004*

Rule: ('2041', '12am to 02am') == ('128') , **1.000** *25.98918916666669, 116.10288165525003, 26.986995773333355, 117.00279799250004*

Rule: ('Tuesday', '2041') == ('128') , **1.000** *25.98918916666669, 116.10288165525003, 26.986995773333355, 117.00279799250004*

Rule: ('147', '02pm to 04pm') == ('Sunday') , **1.000**

Rule: ('06pm to 08pm', '149') == ('Saturday') , **1.000**

Rule: ('2151', 'Saturday') == ('144') , **1.000** *35.96725523333336, 119.70254700425004, 36.96506184000003, 120.60246334150003*

Rule: ('Sunday', '124') == ('10pm to 12am') , **1.000**

Rule: ('Tuesday', '2083') == ('51') , **1.000** *29.980415593333362, 117.00279799250004, 30.978222200000026, 117.90271432975004*

Rule: ('08am to 10am', '2041') == ('128') , **1.000** *25.98918916666669, 116.10288165525003, 26.986995773333355, 117.00279799250004*

Rule: ('114', '10am to 12pm') == ('Saturday') , **1.000**

Rule: ('1021', 'Monday') == ('128') , **1.000** *40.956288266666704, 126.00196136500003, 60.91242040000005, 144.00028811000007*

Rule: ('Tuesday', '2030') == ('128') , **1.000** *24.991382560000023, 114.30304898075003, 25.98918916666669, 115.20296531800004*

Rule: ('156', '4061') == ('Saturday') , **1.000** *39.863690032366705, 116.48534609858129, 39.89861326360004, 116.5235925429144*

Rule: ('Friday', '27') == ('2116') , **1.000** *32.97383541333336, 116.10288165525003, 33.97164202000003, 117.00279799250004*

Rule: ('Sunday', '1013') == ('163') , **1.000** *40.956288266666704, -89.977959575, 60.91242040000005, -71.97963282999999*

Rule: ('137', '10pm to 12am') == ('Thursday') , **1.000**

Rule: ('1021', 'Saturday') == ('128') , **1.000** *40.956288266666704, 126.00196136500003, 60.91242040000005, 144.00028811000007*

Rule: ('Sunday', '2030') == ('128') , **1.000** *24.991382560000023, 114.30304898075003, 25.98918916666669, 115.20296531800004*

Rule: ('150', '02pm to 04pm') == ('4132') , **1.000** *39.968459726066705, 116.33236032124879, 40.00338295730004, 116.37060676558191*

Rule: ('4212', '12am to 02am') == ('13') , **1.000** *40.14307588223337, 116.10288165525003, 40.177999113466704, 116.14112809958316*

Rule: ('2002', 'Wednesday') == ('23') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('Wednesday', '23') == ('2002') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('112', '4061') == ('Friday') , **1.000** *39.863690032366705, 116.48534609858129, 39.89861326360004, 116.5235925429144*

Rule: ('95', '2191') == ('Thursday') , **1.000** *39.95848166000003, 115.20296531800004, 40.956288266666704, 116.10288165525003*

Rule: ('4232', 'Friday') == ('24') , **1.000** *40.177999113466704, 116.10288165525003, 40.21292234470003, 116.14112809958316*

Rule: ('Tuesday', '2075') == ('144') , **1.000** *28.98260898666669, 120.60246334150003, 29.980415593333362, 121.50237967875003*

Rule: ('1001', 'Monday') == ('25') , **1.000** *1.044024, 108.00363462000004, 21.000156133333352, 126.00196136500003*

Rule: ('2151', 'Friday') == ('144') , **1.000** *35.96725523333336, 119.70254700425004, 36.96506184000003, 120.60246334150003*

Rule: ('Sunday', '4122') == ('33') , **1.000** *39.968459726066705, 116.6765783202469, 40.00338295730004, 116.71482476458003*

Rule: ('06am to 08am', '45') == ('Saturday') , **1.000**

Rule: ('10am to 12pm', '149') == ('Saturday') , **1.000**

Rule: ('4077', '2') == ('Saturday') , **1.000** *39.89861326360004, 116.44709965424816, 39.93353649483337, 116.48534609858129*

Rule: ('1018', 'Monday') == ('10') , **1.000** *40.956288266666704, 72.00698113000004, 60.91242040000005, 90.00530787500006*

Rule: ('10am to 12pm', '1001') == ('25') , **1.000** *1.044024, 108.00363462000004, 21.000156133333352, 126.00196136500003*

Rule: ('45', '06pm to 08pm') == ('Saturday') , **1.000**

Rule: ('2154', '04am to 06am') == ('2') , **1.000** *36.96506184000003, 110.70338363175004, 37.962868446666704, 111.60329996900003*

Rule: ('10am to 12pm', '86') == ('Sunday') , **1.000**

Rule: ('06am to 08am', '58') == ('Saturday') , **1.000**

Rule: ('180', '2069') == ('Friday') , **1.000** *28.98260898666669, 115.20296531800004, 29.980415593333362, 116.10288165525003*

Rule: ('134', '02pm to 04pm') == ('4131') , **1.000** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('Friday', '2108') == ('101') , **1.000** *31.976028806666697, 119.70254700425004, 32.97383541333336, 120.60246334150003*

Rule: ('2041', 'Wednesday') == ('128') , **1.000** *25.98918916666669, 116.10288165525003, 26.986995773333355, 117.00279799250004*

Rule: ('7', '04pm to 06pm') == ('4132') , **1.000** *39.968459726066705, 116.33236032124879, 40.00338295730004, 116.37060676558191*

Rule: ('Tuesday', '1001') == ('25') , **1.000** *1.044024, 108.00363462000004, 21.000156133333352, 126.00196136500003*

Rule: ('2002', 'Tuesday') == ('23') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('2085', 'Wednesday') == ('153') , **1.000** *29.980415593333362, 119.70254700425004, 30.978222200000026, 120.60246334150003*

Rule: ('2002', '04pm to 06pm') == ('23') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('4212', '10pm to 12am') == ('13') , **1.000** *40.14307588223337, 116.10288165525003, 40.177999113466704, 116.14112809958316*

Rule: ('02am to 04am', '148') == ('Sunday') , **1.000**

Rule: ('2002', 'Friday') == ('23') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('2168', '142') == ('Friday') , **1.000** *37.96286844666704, 114.30304898075003, 38.96067505333337, 115.20296531800004*

Rule: ('2151', 'Thursday') == ('144') , **1.000** *35.96725523333336, 119.70254700425004, 36.96506184000003, 120.60246334150003*

Rule: ('2005', '5') == ('Sunday') , **1.000** *21.99796274000002, 113.40313264350004, 22.995769346666687, 114.30304898075003*

Rule: ('108', '02am to 04am') == ('Thursday') , **1.000**

Rule: ('161', 'Wednesday') == ('02am to 04am') , **1.000**

Rule: ('2195', '65') == ('Saturday') , **1.000** *39.95848166000003, 119.70254700425004, 40.956288266666704, 120.60246334150003*

Rule: ('63', 'Wednesday') == ('2093') , **1.000** *30.978222200000026, 117.00279799250004, 31.976028806666697, 117.90271432975004*

Rule: ('94', '02pm to 04pm') == ('4132') , **1.000** *39.968459726066705, 116.33236032124879, 40.00338295730004, 116.37060676558191*

Rule: ('125', '08pm to 10pm') == ('Wednesday') , **1.000**

Rule: ('24', '3079') == ('Tuesday') , **1.000** *39.459578356666704, 116.73282309132503, 39.55935901733337, 116.77781890818753*

Rule: ('08am to 10am', '59') == ('Saturday') , **1.000**

Rule: ('2109', 'Wednesday') == ('3') , **1.000** *31.976028806666697, 120.60246334150003, 32.97383541333336, 121.50237967875003*

Rule: ('2002', 'Sunday') == ('23') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('1018', 'Thursday') == ('10') , **1.000** *40.956288266666704, 72.00698113000004, 60.91242040000005, 90.00530787500006*

Rule: ('2109', 'Sunday') == ('3') , **1.000** *31.976028806666697, 120.60246334150003, 32.97383541333336, 121.50237967875003*

Rule: ('12am to 02am', '97') == ('4131') , **1.000** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('10pm to 12am', '34') == ('Sunday') , **1.000**

Rule: ('2154', '08am to 10am') == ('2') , **1.000** *36.96506184000003, 110.70338363175004, 37.962868446666704, 111.60329996900003*

Rule: ('02am to 04am', '2041') == ('128') , **1.000** *25.98918916666669, 116.10288165525003, 26.986995773333355, 117.00279799250004*

Rule: ('2109', 'Friday') == ('3') , **1.000** *31.976028806666697, 120.60246334150003, 32.97383541333336, 121.50237967875003*

Rule: ('02pm to 04pm', '49') == ('Wednesday') , **1.000**

Rule: ('31', '10pm to 12am') == ('Friday') , **1.000**

Rule: ('137', '12am to 02am') == ('Friday') , **1.000**

Rule: ('57', '02pm to 04pm') == ('4132') , **1.000** *39.968459726066705, 116.33236032124879, 40.00338295730004, 116.37060676558191*

Rule: ('2154', 'Tuesday') == ('2') , **1.000** *36.96506184000003, 110.70338363175004, 37.962868446666704, 111.60329996900003*

Rule: ('31', 'Saturday') == ('12am to 02am') , **1.000**

Rule: ('31', '12am to 02am') == ('Saturday') , **1.000**

Rule: ('118', 'Saturday') == ('1020') , **1.000** *40.956288266666704, 108.00363462000004, 60.91242040000005, 126.00196136500003*

Rule: ('119', '2182') == ('Friday') , **1.000** *38.96067505333337, 117.00279799250004, 39.95848166000003, 117.90271432975004*

Rule: ('2080', 'Tuesday') == ('25') , **1.000** *29.980415593333362, 114.30304898075003, 30.978222200000026, 115.20296531800004*

Rule: ('1001', '12pm to 02pm') == ('25') , **1.000** *1.044024, 108.00363462000004, 21.000156133333352, 126.00196136500003*

Rule: ('2083', '08pm to 10pm') == ('51') , **1.000** *29.980415593333362, 117.00279799250004, 30.978222200000026, 117.90271432975004*

Rule: ('31', '06am to 08am') == ('Friday') , **1.000**

Rule: ('2002', 'Monday') == ('23') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('132', 'Wednesday') == ('2076') , **1.000** *28.98260898666669, 121.50237967875003, 29.980415593333362, 122.40229601600004*

Rule: ('2076', 'Wednesday') == ('132') , **1.000** *28.98260898666669, 121.50237967875003, 29.980415593333362, 122.40229601600004*

Rule: ('4130', '112') == ('Saturday') , **1.000** *39.968459726066705, 116.25586743258253, 40.00338295730004, 116.29411387691566*

Rule: ('56', '04pm to 06pm') == ('Wednesday') , **1.000**

Rule: ('124', 'Wednesday') == ('1006') , **1.000** *21.000156133333352, 90.00530787500006, 40.956288266666704, 108.00363462000004*

Rule: ('2109', 'Tuesday') == ('3') , **1.000** *31.976028806666697, 120.60246334150003, 32.97383541333336, 121.50237967875003*

Rule: ('4061', '163') == ('Friday') , **1.000** *39.863690032366705, 116.48534609858129, 39.89861326360004, 116.5235925429144*

Rule: ('Friday', '4192') == ('39') , **1.000** *40.073229419766704, 116.40885320991504, 40.10815265100004, 116.44709965424816*

Rule: ('27', 'Thursday') == ('2116') , **1.000** *32.97383541333336, 116.10288165525003, 33.97164202000003, 117.00279799250004*

Rule: ('1025', 'Friday') == ('172') , **1.000** *60.91242040000005, -161.971266555, 80.8685525333334, -143.97293981*

Rule: ('12pm to 02pm', '49') == ('Wednesday') , **1.000**

Rule: ('4192', '02pm to 04pm') == ('39') , **1.000** *40.073229419766704, 116.40885320991504, 40.10815265100004, 116.44709965424816*

Rule: ('2002', '12pm to 02pm') == ('23') , **1.000** *21.99796274000002, 108.00363462000004, 22.995769346666687, 108.90355095725005*

Rule: ('61', 'Saturday') == ('2184') , **1.000** *38.96067505333337, 118.80263066700003, 39.95848166000003, 119.70254700425004*

Rule: ('2109', '06am to 08am') == ('3') , **1.000** *31.976028806666697, 120.60246334150003, 32.97383541333336, 121.50237967875003*

Rule: ('1003', 'Wednesday') == ('163') , **1.000** *21.000156133333352, -89.977959575, 40.956288266666704, -71.97963282999999*

Rule: ('Tuesday', '160') == ('1000') , **1.000** *1.044024, 90.00530787500006, 21.000156133333352, 108.00363462000004*

Rule: ('2109') == ('3') , **0.974** *31.976028806666697, 120.60246334150003, 32.97383541333336, 121.50237967875003*

Rule: ('2154') == ('2') , **0.972** *36.96506184000003, 110.70338363175004, 37.962868446666704, 111.60329996900003*

Rule: ('120', '06am to 08am') == ('Saturday') , **0.952**

Rule: ('3000') == ('30') , **0.950** *38.96067505333337, 116.10288165525003, 39.060455714000035, 116.14787747211253*

Rule: ('48', '02pm to 04pm') == ('Wednesday') , **0.933**

Rule: ('2030') == ('128') , **0.931** *24.991382560000023, 114.30304898075003, 25.98918916666669, 115.20296531800004*

Rule: ('06pm to 08pm', '134') == ('Thursday') , **0.929**

Rule: ('12am to 02am', '6') == ('Saturday') , **0.929**

Rule: ('02am to 04am', '156') == ('Thursday') , **0.929**

Rule: ('120') == ('Saturday') , **0.927**

Rule: ('55', 'Wednesday') == ('4131') , **0.923** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('120', '08am to 10am') == ('Saturday') , **0.917**

Rule: ('124', '06pm to 08pm') == ('Friday') , **0.917**

Rule: ('62', '04pm to 06pm') == ('Wednesday') , **0.917**

Rule: ('06am to 08am', '70') == ('Sunday') , **0.913**

Rule: ('2040') == ('128') , **0.909** *25.98918916666669, 115.20296531800004, 26.986995773333355, 116.10288165525003*

Rule: ('48', '04pm to 06pm') == ('Friday') , **0.909**

Rule: ('10pm to 12am', '21') == ('Sunday') , **0.909**

Rule: ('75', 'Wednesday') == ('4131') , **0.909** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('1025') == ('172') , **0.900** *60.91242040000005, -161.971266555, 80.8685525333334, -143.97293981*

Rule: ('Monday', '7') == ('4132') , **0.900** *39.968459726066705, 116.33236032124879, 40.00338295730004, 116.37060676558191*

Rule: ('137', 'Thursday') == ('10pm to 12am') , **0.900**

Rule: ('08am to 10am', '58') == ('Saturday') , **0.900**

Rule: ('2083') == ('51') , **0.897** *29.980415593333362, 117.00279799250004, 30.978222200000026, 117.90271432975004*

Rule: ('55', 'Saturday') == ('4131') , **0.889** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('4129', '04pm to 06pm') == ('Wednesday') , **0.889**

Rule: ('08am to 10am', '135') == ('Saturday') , **0.889**

Rule: ('Wednesday', '78') == ('4131') , **0.889** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('Thursday', '78') == ('4131') , **0.889** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('Tuesday', '78') == ('4131') , **0.889** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('02am to 04am', '136') == ('Sunday') , **0.875**

Rule: ('4212', 'Wednesday') == ('13') , **0.875** *40.14307588223337, 116.10288165525003, 40.177999113466704, 116.14112809958316*

Rule: ('2154', 'Wednesday') == ('2') , **0.875** *36.96506184000003, 110.70338363175004, 37.962868446666704, 111.60329996900003*

Rule: ('31', '08am to 10am') == ('Friday') , **0.875**

Rule: ('148', '12pm to 02pm') == ('Saturday') , **0.875**

Rule: ('114', '06am to 08am') == ('Saturday') , **0.875**

Rule: ('2076') == ('132') , **0.867** *28.98260898666669, 121.50237967875003, 29.980415593333362, 122.40229601600004*

Rule: ('2031') == ('128') , **0.867** *24.991382560000023, 115.20296531800004, 25.98918916666669, 116.10288165525003*

Rule: ('46', '08am to 10am') == ('Saturday') , **0.864**

Rule: ('2188') == ('Saturday') , **0.857** *39.95848166000003, 112.50321630625004, 40.956288266666704, 113.40313264350004*

Rule: ('4039') == ('Saturday') , **0.857** *39.82876680113337, 116.10288165525003, 39.863690032366705, 116.14112809958316*

Rule: ('72') == ('Friday') , **0.857**

Rule: ('Tuesday', '4212') == ('13') , **0.857** *40.14307588223337, 116.10288165525003, 40.177999113466704, 116.14112809958316*

Rule: ('126', '4061') == ('Friday') , **0.857** *39.863690032366705, 116.48534609858129, 39.89861326360004, 116.5235925429144*

Rule: ('167', '4061') == ('Friday') , **0.857** *39.863690032366705, 116.48534609858129, 39.89861326360004, 116.5235925429144*

Rule: ('4144', '12am to 02am') == ('41') , **0.857** *40.00338295730004, 116.75307120891316, 40.03830618853337, 116.79131765324628*

Rule: ('Monday', '97') == ('4131') , **0.857** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('4143', '12am to 02am') == ('41') , **0.857** *40.00338295730004, 116.71482476458003, 40.03830618853337, 116.75307120891316*

Rule: ('02am to 04am', '152') == ('Friday') , **0.857**

Rule: ('153', '2188') == ('Saturday') , **0.857** *39.95848166000003, 112.50321630625004, 40.956288266666704, 113.40313264350004*

Rule: ('102', 'Thursday') == ('4131') , **0.857** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('4163', '12am to 02am') == ('41') , **0.857** *40.03830618853337, 116.75307120891316, 40.073229419766704, 116.79131765324628*

Rule: ('4093', '131') == ('Sunday') , **0.857** *39.89861326360004, 116.37060676558191, 39.93353649483337, 116.40885320991504*

Rule: ('08am to 10am', '97') == ('4131') , **0.857** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('10am to 12pm', '172') == ('Friday') , **0.857**

Rule: ('4212', 'Friday') == ('13') , **0.857** *40.14307588223337, 116.10288165525003, 40.177999113466704, 116.14112809958316*

Rule: ('2018', '163') == ('Sunday') , **0.857** *23.993575953333355, 109.80346729450004, 24.991382560000023, 110.70338363175004*

Rule: ('10am to 12pm', '148') 2154== ('Saturday') , **0.857**

Rule: ('4189', '84') == ('Saturday') , **0.857** *40.073229419766704, 116.29411387691566, 40.10815265100004, 116.33236032124879*

Rule: ('36', '04pm to 06pm') == ('4132') , **0.857** *39.968459726066705, 116.33236032124879, 40.00338295730004, 116.37060676558191*

Rule: ('Saturday', '4289') == ('128') , **0.857** *40.24784557593337, 116.33236032124879, 40.282768807166704, 116.37060676558191*

Rule: ('Saturday', '2105') == ('174') , **0.857** *31.976028806666697, 117.00279799250004, 32.97383541333336, 117.90271432975004*

Rule: ('4130', '65') == ('Saturday') , **0.857** *39.968459726066705, 116.25586743258253, 40.00338295730004, 116.29411387691566*

Rule: ('180', '08pm to 10pm') == ('Thursday') , **0.857**

Rule: ('4171', '67') == ('Saturday') , **0.857** *40.03830618853337, 116.33236032124879, 40.073229419766704, 116.37060676558191*

Rule: ('180', 'Saturday') == ('04pm to 06pm') , **0.857**

Rule: ('180', '04pm to 06pm') == ('Saturday') , **0.857**

Rule: ('45', '10am to 12pm') == ('Saturday') , **0.857**

Rule: ('4212', 'Monday') == ('13') , **0.857** *40.14307588223337, 116.10288165525003, 40.177999113466704, 116.14112809958316*

Rule: ('176', '04am to 06am') == ('Friday') , **0.857**

Rule: ('83', '02pm to 04pm') == ('Saturday') , **0.857**

Rule: ('2176', 'Thursday') == ('4') , **0.857** *37.962868446666704, 121.50237967875003, 38.96067505333337, 122.40229601600004*

Rule: ('4022', '29') == ('08am to 10am') , **0.857** *39.79384356990004, 116.5235925429144, 39.82876680113337, 116.56183898724753*

Rule: ('2018', '153') == ('Sunday') , **0.857** *23.993575953333355, 109.80346729450004, 24.991382560000023, 110.70338363175004*

Rule: ('163', '2188') == ('Saturday') , **0.857** *39.95848166000003, 112.50321630625004, 40.956288266666704, 113.40313264350004*

Rule: ('77', '12pm to 02pm') == ('Sunday') , **0.857**

Rule: ('75', 'Saturday') == ('4131') , **0.857** *39.968459726066705, 116.29411387691566, 40.00338295730004, 116.33236032124879*

Rule: ('4141', '12am to 02am') == ('41') , **0.857** *40.00338295730004, 116.63833187591378, 40.03830618853337, 116.6765783202469*

Rule: ('4162', '12am to 02am') == ('41') , **0.857** *40.03830618853337, 116.6765783202469, 40.073229419766704, 116.71482476458003*

Rule: ('48', '12pm to 02pm') == ('Wednesday') , **0.857**

Rule: ('1018') == ('10') , **0.853** *40.956288266666704, 72.00698113000004, 60.91242040000005, 90.00530787500006*

Rule: ('2039') == ('128') , **0.846** *25.98918916666669, 114.30304898075003, 26.986995773333355, 115.20296531800004*

Rule: ('123') == ('2122') , **0.833** *33.97164202000003, 108.90355095725005, 34.9694486266667, 109.80346729450004*

Rule: ('2060') == ('Tuesday') , **0.833** *27.984802380000026, 116.10288165525003, 28.98260898666669, 117.00279799250004*

Rule: ('Friday', '48') == ('04pm to 06pm') , **0.833**

Rule: ('108', '12am to 02am') == ('Thursday') , **0.833**

Rule: ('46', '04am to 06am') == ('Saturday') , **0.828**

Rule: ('4323') == ('128') , **0.818** *40.31769203840003, 116.37060676558191, 40.35261526963337, 116.40885320991504*

Rule: ('2085', 'Monday') == ('153') , **0.818** *29.980415593333362, 119.70254700425004, 30.978222200000026, 120.60246334150003*

Rule: ('Wednesday', '71') == ('4131') , **0.818** *39.968459726066705, 116.33236032124879, 40.00338295730004, 116.37060676558191*

Rule: ('139', '02am to 04am') == ('Thursday') , **0.818**

Rule: ('Wednesday', '2108') == ('101') , **0.818** *31.976028806666697, 119.70254700425004, 32.97383541333336, 120.60246334150003*

Rule: ('08am to 10am', '110') == ('Friday') , **0.818**

Rule: ('31') == ('Friday') , **0.817**

Rule: ('2075') == ('144') , **0.812** *28.98260898666669, 120.60246334150003, 29.980415593333362, 121.50237967875003*

Rule: ('32', '04am to 06am') == ('Thursday') , **0.812**

Rule: ('97') == ('4131') , **0.805** *39.968459726066705, 116.33236032124879, 40.00338295730004, 116.37060676558191*

Rule: ('2083', 'Monday') == ('51') , **0.800** *29.980415593333362, 117.00279799250004, 30.978222200000026, 117.90271432975004*

Rule: ('Wednesday', '127') == ('1021') , **0.800** *40.956288266666704, 126.00196136500003, 60.91242040000005, 144.00028811000007*

Rule: ('80', '10am to 12pm') == ('Friday') , **0.800**

REFERENCES

[1] X. X. Yu Zheng, Lizhu Zhang and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *Proc. ACM International conference on World Wild Web WWW'09*, Madrid, Spain, 2009, pp. 791–800.

[2] Y. C. X. X. Yu Zheng, Quannan Li and W.-Y. Ma, "Understanding mobility based on gps data," in *Proc. ACM ACM conference on Ubiquitous Computing UbiComp'08*, Seoul, Korea, 2008, pp. 312–321.

[3] X. X. Yu Zheng and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data Engineering Bulletin*, vol. 33, pp. 32–40, 2010.

[4] F. Verhein and S. Chawla, "Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases," vol. 3882, pp. 187–201, 2006.

[5] L. Manikonda. Introduction to spatio-temporal databases. [Online]. Available: http://www-users.cs.umn.edu/ lmani/spatial/HW5-B4.pdf

[6] [Online]. Available: https://www.python.org/

[7] [Online]. Available: https://www.continuum.io/downloads

[8] N. H. Son. (2006) Data cleaning and data preprocessing. [Online]. Available: http://www.mimuw.edu.pl/ son/datamining/DM/4-preprocess.pdf

[9] [Online]. Available: https://en.wikipedia.org/wiki/Association_rule_learning

[10] A. Wasilewska. (2016) Apriori algorithm. [Online]. Available: http://www3.cs.stonybrook.edu/ cse634/lecture_notes/07apriori.pdf

[11] asaini. (2015) Apriori algorithm. [Online]. Available: https://github.com/asaini/Apriori

BIOGRAPHICAL STATEMENT

Vivek Kumar Sharma was born in Hamirpur, Himachal Pradesh India, in 1990. He received his B.Tech. degree from Guru Nanak Dev University, Amritsar, Punjab India in 2011 and his M.S. degree from The University of Texas at Arlington in 2016, majoring in Computer Science. From 2011 to 2013, he was with the Capgemini India Pvt. Ltd. as a Quality Analyst in the Data Warehousing domain. In 2015, he interned at Grifols Theraputics as an Application Developer working on automating the data analysis process of their product line. His current research interest is in the area of Data Mining and Big Data problems.