

Institut für Parallele und Verteilte Systeme

Abteilung Verteilte Systeme

Universität Stuttgart

Universitätsstraße 38

D-70569 Stuttgart

Master Thesis

**Design and Development of**

**Software Agents for**

**Location Privacy-risk estimation**

Shashank Sharma

**Studiengang:** INFOTECH

**Prüfer:**  Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel

**Betreuer:**  M. Sc. Zohaib Riaz

**Begonnen am:**  5th May 2018

**Beendet am:**  5th Dec 2018

# Abstract

The rise of information technology in the last two decades has been phenomenal. The usage of mobile devices has become ubiquitous in today’s world. More people are adopting technology to be able to connect with family and friends, easily perform day-to-day tasks, discover new facets, etc. Most of the modern mobile devices are GPS capable. The number of options to receive location information has risen tremendously. Mobile phones, personal computers and car navigation systems, are all location aware devices. Thus, considering the popularity of GPS enabled devices, the privacy of the users is arguable.

The human geographical location is shared with different applications on a mobile device. The sharing of location is active not only when the application is used, but also share the user whereabouts with third-party applications in the background. This location is used by different applications for advertising, recommendations, finding new friends, suggesting new point of interests, etc., based on user trends. The location data is shared while we are navigating using Google Maps, tagging a lunch party on Instagram or simply connecting to a public Wi-Fi. The location data can also be used by different third-party applications to predict user’s future locations. This in turn can model the entire user mobility patterns. The characteristics revealed just by location details are narrowly perceived by most users.

In this thesis we present an algorithm which predicts a user’s future movements with confidence percentages. This algorithm is first processing the raw GPS coordinates and extracts the meaningful locations. Based on the meaningful locations, a prediction model is formed. This model is used to predict the future visits based on user’s current location. The algorithm is evaluated using real-life data from Microsoft Geolife dataset [1] [2] [3]. This data contains 182 user trajectories data for a period of 5 years. To inform user about this privacy threat, a path prediction model is suggested. Finally, a prototype on an android device is presented to be able to inform user about their location predictability.

# Acknowledgement

I would like to thank my thesis supervisor, M. Sc. Zohaib Riaz, for his valuable assistance, useful critiques, time and suggestions during the thesis. His willingness to give me constructive advices has been very much appreciated. I’m also grateful to Prof. Dr. Kurt Rothermel for allowing me to do my thesis in the Department of Distributed Systems.

I would like to extend my thanks to my family members, for their enthusiastic encouragement, and to all my friends for their motivation.

Contents

[Abstract 3](#_Toc530415001)

[Acknowledgement 4](#_Toc530415002)

[List of Figures 7](#_Toc530415003)

[List of Tables 9](#_Toc530415004)

[**1.** **Introduction** 10](#_Toc530415005)

[1.1 Motivation 11](#_Toc530415006)

[1.2 Contribution and Thesis Outline 11](#_Toc530415007)

[**2.** **Related Work** 13](#_Toc530415008)

[2.1 Location Privacy 13](#_Toc530415009)

[2.2 Extracting Interesting Locations 13](#_Toc530415010)

[2.3 Next Location Prediction 15](#_Toc530415011)

[2.4 Algorithms Comparison 18](#_Toc530415012)

[**3.** **System Model and Problem Statement** 20](#_Toc530415013)

[3.1 System Model 20](#_Toc530415014)

[3.1.1 Model Overview 20](#_Toc530415015)

[3.1.2 Components 22](#_Toc530415016)

[3.2.3 Hypothesis 23](#_Toc530415017)

[3.2 Problem Statement 24](#_Toc530415018)

[**4.** **Concept and Implementation** 25](#_Toc530415019)

[4.1 Overview 25](#_Toc530415020)

[4.2 Transition Probabilities 27](#_Toc530415021)

[4.3 State Formation Steps 29](#_Toc530415022)

[4.4 Implementation 30](#_Toc530415023)

[4.4.1 Variables Used 30](#_Toc530415024)

[4.4.2 Stay-points 31](#_Toc530415025)

[4.4.2.1 Sporadic GPS Trajectories 31](#_Toc530415026)

[4.4.2.2 Estimating Arrival and Departure times of Stay-points 32](#_Toc530415027)

[4.4.2.3 Algorithm 33](#_Toc530415028)

[4.4.3 State Formation 36](#_Toc530415029)

[4.4.3.1 Drifting Problem 36](#_Toc530415030)

[4.4.3.2 Algorithm 37](#_Toc530415031)

[4.4.4 State Weights 38](#_Toc530415032)

[4.4.5 Forming Markov chain 39](#_Toc530415033)

[4.4.5.1 Algorithm 40](#_Toc530415034)

[4.4.5.2 Approach with Additional Steps 40](#_Toc530415035)

[4.4.6 Path Prediction 42](#_Toc530415036)

[4.4.6.1 Approach 42](#_Toc530415037)

[4.4.6.2 Proposition 44](#_Toc530415038)

[**5.** **Android Implementation** 46](#_Toc530415039)

[5.1 Objective 46](#_Toc530415040)

[5.2 User Interface Design 46](#_Toc530415041)

[5.3 Future Work 48](#_Toc530415042)

[**6.** **Evaluation** 49](#_Toc530415043)

[6.1 User data analysis 49](#_Toc530415044)

[6.2 Evaluation and Results 51](#_Toc530415045)

[6.2.1 Stay-points Evaluation 51](#_Toc530415046)

[6.2.2 Time-slotted States 52](#_Toc530415047)

[6.2.3 Path Prediction Evaluation 55](#_Toc530415048)

[6.2.3.1 Survey for Memory-loss Factor 56](#_Toc530415049)

[6.2.4 Prediction Performance 59](#_Toc530415050)

[6.2.4.1 Metric 59](#_Toc530415051)

[6.2.4.2 Results 60](#_Toc530415052)

[**7.** **Conclusion and Future Work** 62](#_Toc530415053)

[7.1 Summary 62](#_Toc530415054)

[7.2 Discussion 63](#_Toc530415055)

[7.3 Future Work 63](#_Toc530415056)

[Declaration 67](#_Toc530415057)

# List of Figures

[Figure 1 Time-based clustering [6] 15](#_Toc530415058)

[Figure 2 Design Flow-chart 21](#_Toc530415059)

[Figure 3 GPS coordinates to Markov Chain Model 22](#_Toc530415060)

[Figure 4 Markov chain on states 23](#_Toc530415061)

[Figure 5 Markov chain example on a map 25](#_Toc530415062)

[Figure 6 Path modeled using markov chain 26](#_Toc530415063)

[Figure 7 Markov chain derived from state 27](#_Toc530415064)

[Figure 8 State transitioning 27](#_Toc530415065)

[Figure 9 State weights in each time-slot 28](#_Toc530415066)

[Figure 10 State weights normalized 28](#_Toc530415067)

[Figure 11 State transition matrix 29](#_Toc530415068)

[Figure 12 Markov model for two states 29](#_Toc530415069)

[Figure 13 Example 1 of missing data 33](#_Toc530415070)

[Figure 14 Example 2 of missing data 33](#_Toc530415071)

[Figure 15 Extracting stay-points from GPS Trajectories 34](#_Toc530415072)

[Figure 16 Snapping stay-points to states 36](#_Toc530415073)

[Figure 17 Drifting problem while snapping stay-points to the states 37](#_Toc530415074)

[Figure 18 Normalizing state weights 39](#_Toc530415075)

[Figure 19 Forming time-slotted data from states 41](#_Toc530415076)

[Figure 20 Transition matrix for state 1 and state 2 for time-slot 5 (left without and right with addition of the dummy values) 41](#_Toc530415077)

[Figure 21 Steps followed to smooth the transition matrix 42](#_Toc530415078)

[Figure 22 Path Prediction Model using Markov Model 43](#_Toc530415079)

[Figure 23 Steps to apply memory-loss factor to the markov chain 44](#_Toc530415080)

[Figure 24 Example of application of memory-loss factor 45](#_Toc530415081)

[Figure 25 Android application screen 1 47](#_Toc530415082)

[Figure 26 Android application screen 2(left) and 3(right) 48](#_Toc530415083)

[Figure 27 User transportation mode 49](#_Toc530415084)

[Figure 28 Geolife dataset trajectory average speed 50](#_Toc530415085)

[Figure 29 User 1 start and end points from each trajectory 50](#_Toc530415086)

[Figure 30 Stay-points count for different time and distance thresholds 52](#_Toc530415087)

[Figure 31 User 1 raw trajectory data vs stay-points extracted 53](#_Toc530415088)

[Figure 32 User 1 eight days’ time-slotted; 1. Top: Raw Trajectory Data, 2. Middle: Stay-points within Trajectories only, 3. Bottom: Stay-points with our algorithm 54](#_Toc530415089)

[Figure 33 User 1 time-slotted data for the November 2008 55](#_Toc530415090)

[Figure 34 Path prediction for user 1 56](#_Toc530415091)

[Figure 35 User 1 November 2011 hourly distributed data 57](#_Toc530415092)

[Figure 36 Path prediction user 1 with minimum confidence of 50% 57](#_Toc530415093)

[Figure 37 Path prediction result for user 1 without changes in the algorithm 58](#_Toc530415094)

[Figure 38 Path prediction result for user 1 with memory-loss factor of 0.22 58](#_Toc530415095)

[Figure 39 Memory-loss factor result from survey 59](#_Toc530415096)

[Figure 40 Correct(left), Incorrect(middle) or none(right) prediction scenario 60](#_Toc530415097)

[Figure 41 Evaluation for Geolife dataset 61](#_Toc530415098)

# List of Tables

[Table 1 Variables used in algorithms 31](#_Toc530415099)

# **Introduction**

Personal computers and mobile devices have become part of everyday life. It is ubiquitous to interact with computers on an everyday basis. Some of these computers are capable of location awareness, for instance, a mobile device. There are many applications in our mobiles and computers which uses location data for personalized recommendations and advertisements. These applications are called location-based services. The applications uses user location data while the application is active and inactive. The information is often stored on the servers. Most applications have in their terms and condition mentioned the usage of the location data which are easily ignored or misunderstood. Consider an example of a weather application forecasting the weather for the next few days on your mobile device. The weather application is using the current location every time the user is moving to a new location for fitting weather forecasts. Another example is of a social networking website like Facebook. The user makes check-ins and shares location uninterruptedly to find friends and recommendations. It is important to understand that this location data can be used to model human mobility. It should not be forgotten that it enhances quality-of-life but at the same time can reveal mobility patterns based on the data received in the past.

The geographic location can be shared from many sources like Global System for Mobile Communication GSM, Global Positioning System GPS, Wi-Fi network location and so on. The most predominant use of location data is the modern navigation system like Google Maps and social networking websites like Facebook and Instagram. The user shares outdoor movements with the application to receive path recommendations or share some events with friends and family. The path reveals two important facts, the source location and the destination location, and the path taken. The source could be “home”, “hotel”, “shopping mall” or a “restaurant” location and so could be the destination. The social events shared can reveal a user’s favorite restaurant types or a club/gym he/she has been visiting in the past. When this information is collected for several days, it could reveal the user’s “home”, “work” and other important locations. Also, it reveals, the time spent at these locations, arrival and leaving times and the transitions from one place to another. This could help, for instance, a restaurant application to suggest a new restaurant, keeping in mind the user’s home location, type of food he/she prefers/like and his/her time preference of visiting a restaurant. If this data is continuously stored, a mobility model can be created. This, in turn, can make predictions based on a user’s current location. For instance, if the user is at “home” at 9 am, the mobility pattern can reveal his next locations for the rest of the day, using the model.

An overwhelming number of personal characteristics can be predicted online based on location data. This can reveal user’s private information like the illnesses based on the specialist he/she has been visiting, vacation locations he/she has been to and many others. This is a privacy attack. This information can also be leaked on third-party applications and can have unauthorized access. It is easy to oversee the privacy threats that location-based services can impose. The goal of this thesis is to acquaint users about the consequences and the predictability of the future visits. This is achieved by making location predictions based on location-based data received for a few weeks. The future location predictions are shown to the users with confidence percentages.

## 1.1 Motivation

The location-based data can tell a lot about the user. If the location-based data is received for a few weeks, it is easy to answer questions like, where does user live/work? Where is the user’s club/gym? What restaurant does he/she likes? Where is he/she on weekends? Which hospital has he/she been visiting? All the answers to these questions can give an insight about the user’s private life and whereabouts. After a few weeks of data, the algorithm can precisely define the user’s movement trends, his/her favorite places, his/her lifestyle and so on. With continuous learning, this data can be updated and have the user’s “home” location and “work” location updated with time. This is a privacy attack which makes use of the user’s past location data. Several cases have been emerged in the past where the location data is used unlawfully. For example, Michael Cunningham and his family [4] was tracked for 24 hours for a period of two months using a GPS device. This was initiated as investigation on whether he filled out his time sheets at work accurately. Cunningham was terminated at work based on the GPS data reports which tracked his whereabouts outside the working hours as well, in which he lied about being at work.

This private data is often distributed to third parties. This could be an attack on user’s private life, compromising user privacy and sharing his data with other applications, friends and family. With the advancement of Artificial Intelligence and Machine Learning, it has become even easier to exploit this data, understand bulk geographic data and infer meanings from several locations.

## 1.2 Contribution and Thesis Outline

The thesis contributions are:

* Design and development of location prediction algorithm.
* Process the raw location coordinates for stay-point detection and state formation.
* Evaluate the algorithm on real-life location data from Microsoft Geolife dataset.
* Implement the markov chain model algorithm on android application.

This thesis is organized as follows:

**Chapter 2 (Related Work)** briefs the related work researched in location prediction and presents the basic understanding of the topics.

**Chapter 3 (System Model and Problem Statement)** describes the building blocks of the model which includes components used in the algorithm and the hypothesis in general. The components explain the [intermediary](https://www.google.de/search?q=intermediary&spell=1&sa=X&ved=0ahUKEwjnh8r0xKPeAhUGWSwKHWMECDAQBQgrKAA) steps followed to build location prediction model from raw GPS coordinate points. This chapter also includes the problem statement in general.

**Chapter 4 (Concept and Implementation)** explains the concepts build for location prediction model. In this chapter, we first provide an overview of the concept. After that, each element implementation is detailed along with the algorithms. This chapter explains the aspects for building a location prediction model and path prediction based on the location data received.

**Chapter 5 (Android Implementation)** contains the user interface of android implementation, details about the application and potential improvements of the application in future.

**Chapter 6 (Evaluation)** contains the evaluation of each individual approach and results of the location prediction model. This outcome of the algorithm on the dataset is also discussed here.

**Chapter 7 (Conclusion and Future Work)** concludes this thesis by summarizing the work done, results obtained, open topics and future work.

# **Related Work**

In this section, we introduce the related work in the field, the references and motivational work and few parts in details.

## 2.1 Location Privacy

The modern computing devices are location-aware. This could pose several privacy threats. The authors [5] surveys how the location-aware computers can be a threat to our private information. The attacker can gain access to this data and reveal many private elements of user behavior.

The authors [5] describes that the information can be gained by first-hand communication and second-hand communication, observation, or inference. The first-hand communication takes place when the user provides the information to the attackers first-hand. An example of first-hand commination is WLAN proving the MAC address. Another example is the location-based services like Google Maps. The second-hand communication takes place when the attacker gains the information from third party. In ubiquitous computing, this is often the case where the information shared with one website is often spread with the others to gain knowledge on user preferences. The attacker may also gain information by observing the user environment. An example of observation attack is the cameras installed in public. The last approach is the inference where with the enough data about the user, inferences can be drawn. For instance, if a user is often visiting a Cardiac Care, he/she has some heart related issues. With enough data, the modern technologies like Machine Learning algorithms can easily draw inferences based on the data collected. The authors [5] also proposes solutions like policies, limiting first-hand communication or reducing amount of information disclosed to third parties. Since the solutions are not the area of interest of this thesis, we do not discuss them in detail here.

## 2.2 Extracting Interesting Locations

The first step for location prediction is to extract meaningful information from location data. This can take place with first-hand communication. An example of this is Google Maps asking the user to tag locations like “home” or “work”. This reveals directly the most important locations for the user. Another approach of finding the significant places from raw GPS trajectories is inferences. The inferences can be drawn based on the location data for few weeks, which in turn, reveals the locations like “home”, “work”, “favorite restaurant”. This can be done using several clustering algorithms. It is important to understand that the fundamental clustering algorithms like k-mean clustering is insufficient to extract meaningful locations. The typical clustering algorithm do not consider factors like travelling GPS coordinates or short duration stays. We discuss some related work in this subsection for extracting the interesting locations.

Researchers [6] investigate how to extract significant places for the users from raw coordinates data. The author suggests that users are more interested in “places” rather than location. By “places” they mean where the user work/live/play and so on.

Since the Wi-Fi shares the MAC address periodically, hence, the location information is received continuously if the Wi-Fi is connected. The researchers used Place Lab to collect user location data from Wi-Fi enabled devices which works best also for indoors locations. This MAC addresses are then converted into latitude and longitude information with an estimate of 50-100m. The estimation works best in urban areas where the density of access points is high.

The authors [6] compared the typical clustering algorithms, k-mean and Gaussian mixture model, on the location data received from Place Lab. Two major drawbacks reported are, the input on the number of clusters in advance, and clusters becoming large comprising of unimportant locations. Knowing the count of clusters or significant locations in advance is difficult as it can vary with users. There are several known algorithms to compute the number of ideal clusters on its own, but it parallelly increases the complexity of the algorithm. Another issue with these clustering was the increased size of clusters. These large clusters contain unimportant locations because of several transitions between the locations.

The authors [6] introduces a time-based clustering algorithm to determine user’s interesting locations. The algorithm is depicted in the Figure 1. This algorithm waits for the next location to determine if it belong to the significant place cluster or not. The cluster within a distance threshold which is stayed for at least a given time threshold is considered as a significant place. If the next point is moving geographically away from the cluster mean location, then the new point is not added to this cluster. At this point, the previous cluster total duration stay is checked. If the stay duration is greater than certain time threshold, the cluster is regarded as a significant location, otherwise the cluster is deleted. Hence this clustering excludes all irrelevant or shortly stayed locations. In the Figure 1, the clusters *a* and *b* are considered as a significant place and the intermediator clusters like i1, i2, i3 are rejected as the duration of the stay was less than the threshold.

The location data is collected from Place Lab and tracked every second. Their [6] results show that the algorithm could extract significant places and works better than k-mean or Gaussian mixture model clustering on location data. The researchers also suggest that the locations must be labelled to extract semantic meaning behind the location coordinates like work/restaurants and so on. The time-based clustering, with some extensions, is also used in this thesis for stay-points extraction.

A research done by [1] also investigate in the direction of mining the interesting locations from location data. The authors aim to extract the interesting locations and classical travel sequences based on GPS trajectory data. Based on multiple GPS trajectories, a Tree-based Hierarchical Graph (TBHG) is created. After this, an approach called Hypertext Induced Topic Search (HIT) is applied.

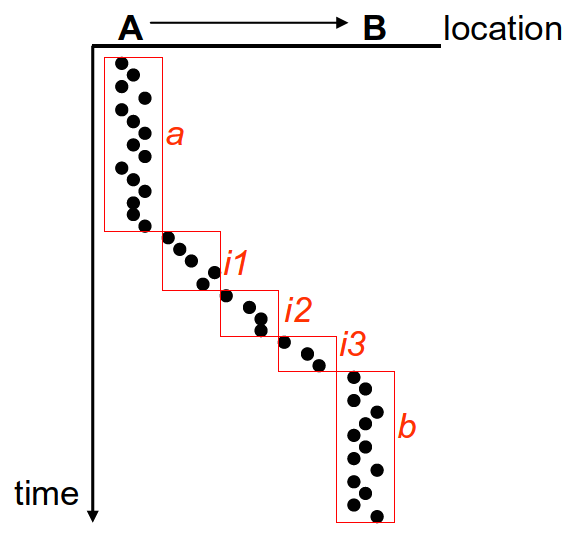


Figure 1 Time-based clustering [6]

In this approach [1], a user will be linked to many locations and a location will be linked to many users. The links between users and locations are weighted based on user GPS trajectory data. A hub score is given to a geographical region for a user based on the travel experiences of the user. It is suggested that a user with a high hub score in a region will visit many places in that region and has a rich travel experience in that region. Considering the hub scores from several users, many interesting places can be mined in the regions. This provides each location within the hub regions an authority score. A user with high travel experience in one region will contribute more to estimate an interesting location in that region.

The researchers [1] system is used by 107 users and the work is also part of the Geolife location dataset from Microsoft [1] [2] [3]. A comparison with rank-by-count and rank-by-frequency is done. Rank-by-count ranks a location interesting if more users have visited that place and rank-by-frequency ranks a location interesting if the location has been visited by the users more frequently. Their approach out performed these two algorithms. Since this approach requires travel experiences from multiple user location trajectories to extract the significant location for one user, we keep this methodology for potential future work.

## 2.3 Next Location Prediction

After extracting the interesting locations, the next step is to use them for location predictions. The next location prediction can inspect user’s day-to-day locations. This can also suggest user’s future placements. Research was done by [7] in the field of next place location prediction which suggests the next predicted location based on user behavior. The main idea is to use user check-ins on Foursquare to predict user movements. The data of 35 million check-ins from across the globe over the period of 5 years is used. The idea is explained how user check-ins not only allows us to see the locations user visited in the past but also help us understand the mobility patterns of the users. They have used prediction features like user preferences, the popularity of the places and geographic distance between places. On these features, they have applied supervised the learning linear model and M5 model trees.

One of the first tasks addressed [7] is the next check-in prediction. The next check-in is predicted based on the current check-in data and several other factors. First, all the possible next location check-in based on current check-in are ranked, a suggestion that 99% of the next check-ins are within 10 kilometers radius from the current check-in. The check-ins are also mostly in urban areas. The ranking is performed based on historical visits by the user to a place, categorical preferences based on what category of places have user checked-in in the past and social filtering based on where user’s friends are checked-in. The next task is a global mobility feature to determine check-in patterns irrespective of user preferences. This uses the popularity of the geographic location, geographic and relative distance of all the other locations from user’s current location and activity transition where few locations are visited after specific locations, for instance going to a hotel after an airport or railway station visit. Next, they assign the temporal features to each place. Based on the hour category, what type of place has been checked-in in a particular hour of the day or week?

After the assigning these features, the ranks(k), percentile rank (PR) and average percentile rank (APR) are defined for each venue. The highest rank is given to the location where the next check-in could take place. The analyses from the researchers suggest that APR is scored higher for categorial preferences with 0.84 when compared to historical visits with APR 0.68 and social filtering with APR 0.61. In global mobility section, place popularity has better APR which is 0.86. Activity transition features also achieve only 0.60. The study [7] also suggested that people tend to stick to their set of location check-ins during the day time but visited new locations during evenings. All these features suggest that there could be many factors which can affect user movement patterns. They finally used all these features and combined them into a supervised learning framework. With M5 tree, they have received an APR of 0.94 and linear regression model only resulted with an APR of 0.81 which is less than many individual feature APR. The authors [7] explained how the prediction model can have better performances with several features combined. The evaluation is done on a social networking check-in dataset and hence this could have influenced the evaluation results. In the scope of this thesis, the GPS trajectory data is to be received as an input from mobile devices and not intermittent location sharing like only check-ins, which could sometime have gaps of days between them. Hence, we do not explore in this direction further in this thesis.

Authors [8] also discusses mobile based next location prediction based on current location. They suggest using contextual data along with spatial and temporal data associated with location. The mobile call/SMS logs, accelerometer and Bluetooth can have additional information which has not been investigated before for location predictions. The researchers explain how location prediction is very user specific, the data is evolving with changing city/work location, etc. and it is possible to have missing location data.

The model [8] pre-process the raw data which keeps the short-term data with its contextual information, the model should accept and integrate new location data and an updated check of actual next location vs predicted next location to keep an updated accuracy rate. This model is implemented in an online manner on a mobile device. The data used as an input is from Nokia Mobile Data Challenge (MDC) from 200 participants over one year. First, the raw data is processed to extract temporal features, phone status, phone usage and other features. These features include the hour and time duration of a visit, the ring-tone used, last call/SMS log and others. Then a classification technique is used with a software named WEKA.

The results [8] suggested that the regular users were easy to predict with an accuracy percentage of 80% whereas the users with irregular movement patterns are difficult to predict. In feature selection phase, it has been found that keeping all the features gives the best results with accuracy of 92%. The contextual information addition is added as a potential future work of this thesis.

The paper [8] also suggests an alternative advertisement approach. For instance, a user will be more interested in dinner promotion/discount before he/she goes outside on dinner time. The user can share the location-based data with the telecom provider, and the telecom service provider can act as an intermediary between the user and the restaurant advertisement company. The third party like, in this case, the restaurant company, can push the relevant advertainments on the user’s phone based on the predictions result shared by the telecom provider, without disclosing user’s information or location data to the third parties. In this scenario, the user can receive more relevant advertisements and still have not shared his private information with advertisement companies. This can help in preventing personal data to be shared with companies and third parties and hence preserving user privacy.

The authors [9] design hierarchical hidden semi-markov-model concerning spatio-temporal of location data to predict human mobility patterns. The hidden semi-markov model does not consider the distance between the two observations, whereas the proposed hierarchical model considers the distance between the observations. In this model, each state denotes either a stay-point or a transition from one place to another. The states which are more visited will be super states consisting of other states, and the states geographically closer or spatio-temporally closer are more likeable to be in one state. Hence, there are super states which contain other states. The next step is to map each location coordinate in a grid with cell id. The algorithm becomes expensive with increasing states. The states can be reduced by using states at a higher level in the hierarchy. This, in turn, reduces the total complexity of the algorithm. The researchers have used real-life location data Geolife dataset and Capricorn dataset. The approach has better results in the presence of noise and missing data.

## 2.4 Algorithms Comparison

The author [10] compared 18 different location prediction algorithms. The focus here is on the accuracy of prediction along with other parameters to compare these algorithms with each other. Based on their analysis and knowledge gained during the algorithm comparison, they also present a new next-place prediction algorithm called MAJOR. The dataset used by the researchers is Nokia Mobile Data Challenge (MDC). It contains 37 user mobile phone data over 1.5 years.

They [10] have considered several spatial and temporal features with different combinations and named each algorithm based on the features used. For instance, the features used are, current location of the user P1, current and previous location of the user P2, time of the day H, Day of the week D, weekday or weekend W. Here P1 and P2 are spatial features and H, D and W are temporal features. Using the combination of these spatial and temporal features, they have formed several algorithms e.g., DP1, WHP2 and so on. On these algorithms, several performance metrics are calculated, which contains factors like accuracy percentage A1 which is the ratio of correct predictions to the total predictions. Other performance factors which included the true positive, false positive, true negative and false negative with respect to transitions. For example, a true positive transition is the transition which is correctly predicted from one place to another. True positive transition rate TTPR is the ratio of true positive transitions over the total transitions. Similarly, other rates like false positive transition rate, are calculated. Some other interesting performance parameters included transition precision ratio which is calculated as ratio of number of correctly predicted transitions and the total predicted transitions. The researchers also considered the arrival and departure events prediction from a particular place as a performance metric.

Using the combination of different spatial and temporal features, they [10] have compared 18 different prediction algorithms for their predefined performance metrics. The first comparison is highlighted for algorithms considering only spatial features, only temporal features or both together. The most algorithms which can achieve a good prediction accuracy fail to predict a transition and vice versa. This led to the conclusion that there exists a trade-off between prediction accuracy and transition prediction. This is overcome by the novel approach introduced in the paper [10] called MAJOR. This new approach runs all 18 algorithms (spatial and temporal combinations) together and selects the one with the highest votes. This means that, if the highest vote among the 18 algorithms suggests a transition, then the transition is predicted, otherwise not. The same voting approach is applied for the prediction of next place. This gave MAJOR an accuracy of 82% but only 21% detection of true transition. To improve the transition detection ability, they have introduced a voting threshold. The analyses suggested that a median of 8 approaches predict a true transition and a median of 3 approaches predict a transition when no transition occurs. This will help to decide the voting threshold offline. If the minimum number of approaches voting for the transition is greater than or equal to the threshold, then it is considered a transition, otherwise not. The researchers [10] help us to understand the different metric which is important in location prediction algorithms. The paper also suggests that a high accuracy algorithm will have a trade-off for detecting the true transitions.

# **System Model and Problem Statement**

## 3.1 System Model

In this chapter, we will discuss the system model that is proposed for location prediction, an overview of the components that are involved and the hypothesis we draw out of our analysis.

### 3.1.1 Model Overview

The human mobility pattern can be dependent on several features like user’s occupation. For instance, if the user is a, salesperson or has an occupation which requires daily travel, it is very unlikely that the user has a regular “home-work-home” pattern. These users are difficult to be predicted. Oppositely, users who have very regular movement patterns are easily predicted. The idea is to have a prediction model which can work for everyone.

The user tends to have a pattern where the next location is dependent on their current location. Consider an example where “work” is often visited directly after “home”, but “home” is not always visited directly after “work.” This is a common trend where the user visits “restaurants”, “gym” or some other location after “work” before he/she comes back to “home.” It is very often that after “supermarket” visit, the user tends to go back to “home.” These trends could be often predictable but also sometimes not obvious. For instance, a “restaurant” visit could occur after “home” or “work” visit or even after a “shopping mall” visit. Hence, we can say that many transitions from one place to another, are dependent on the current location and many can vary based on situations. Hence, the thesis suggests predicting the next location based on markov chain, by noting the states representing user significant places based on his/her past visits.

The system model contains several steps. The process takes GPS trajectory points as input and processes them to create markov chain model on states. The intermediate steps are stay-point detection, state formation, time-slotted states creation and state transitions. Each of these intermediator steps are introduced here and are explained in detail in further sections.

The flow chart depicted in Figure 2 explains how the model works. The input GPS coordinates data with the date and time information is fed to the algorithm. The algorithm keeps collecting the GPS points unit the end of a time-slot which is depicted using the “hour change” decision block. After the end of the time-slot, several steps are performed on these points from the previous hour. The several steps are:

* Detect stay-points (also detect the start or end of the trajectory.)
* Adjust leaving time from one location and arriving time to another location based on speed and distance between them.
* Group the stay-points to form states.
* Create time-slotted data from the states.
* Apply Markov chain for the states available.

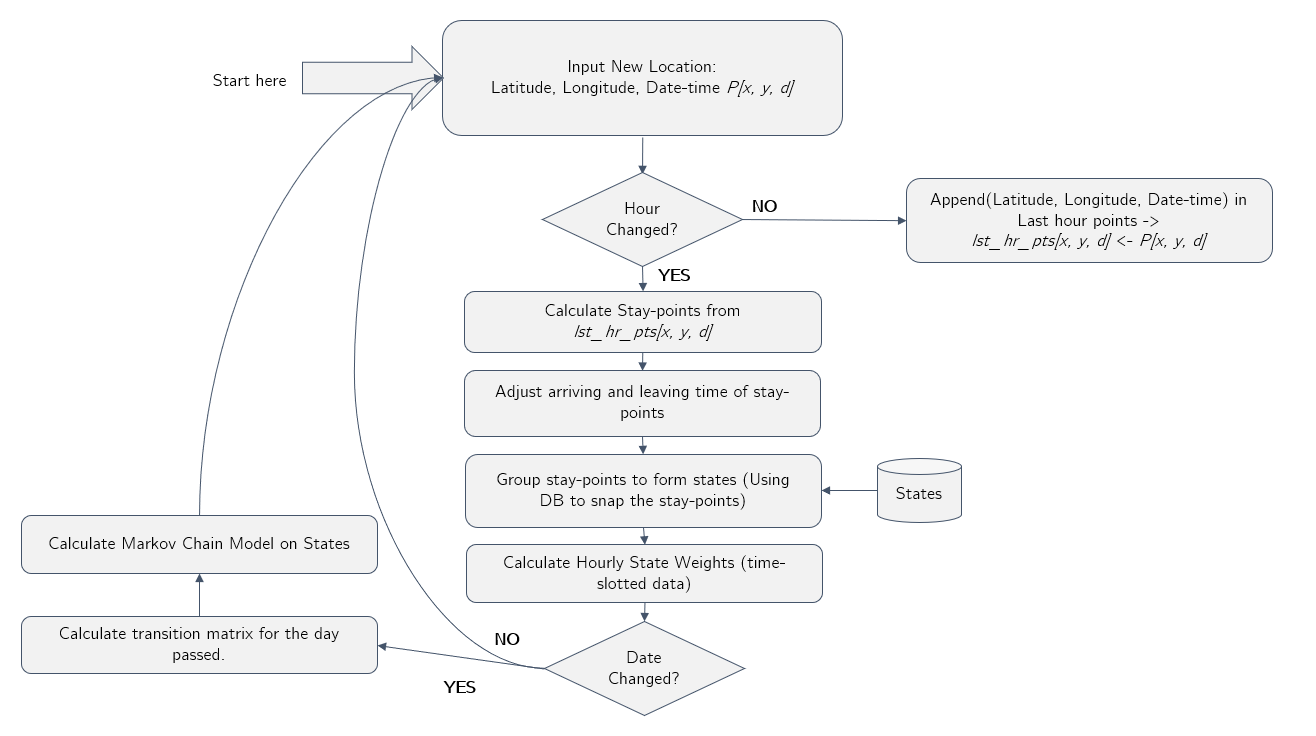


Figure 2 Design Flow-chart

The location coordinates *P[x, y, d]* are received and after each time-slot, significant locations called stay-points are extracted. These stay-points represents “home”, “supermarket”, etc. Once the stay-points are extracted, each stay-point arrival time and leaving time is adjusted based on the distance and time between two consecutive stay-points. After this smoothing, the stay-points are clustered together to form states. These states used to form time-slotted data and finally forms the markov chain model. This is just a quick introduction of the model. Each of these elements are explained in detail in the further chapters.

In this thesis, we present a location prediction model. The location-based data acts as input for the model and forms a prediction model as an output. This model is then used to make future location predictions with confidence percentages, based on a known location as input. The location data used is from Microsoft Geolife data [1] [2] [3] of 182 users for 5 years.

This algorithm is designed to predict human locations in a real-world scenario. We use this location prediction model to predict the future locations with confidence percentages.

### 3.1.2 Components

In the model, the location data is made input as an online GPS location data on a mobile system. This is to simulate how user shares the location details with other Location Based applications like Google+, Facebook, etc. These locations are sampled and the noise (travelling locations or short stay locations like post-office visit) are removed. This makes sure that the markov model is built on stable and longer stayed locations which are later formed as states.

These location coordinates are read from Geolife dataset user files in an online manner. This location data has latitude, longitude, date and time information along with other information. This location coordinates are read in an online manner to extract the locations where the user has spent more time. These locations are called stay-points sp. The extraction of stay-points is stored as {sp1, ap2, … spn}. These stay-points are the significant places for this user which has semantic meaning behind the location coordinates specific to the user.

Once the stay-point is extracted, we snap these stay-points to states which forms st = {st1, st2, …stn}. The states st are simply combining the similar stay-points based on their geographical distance from each other. If there is already a state existing, which is close-by to the new stay-point location, it is snapped to the existing state. If a new location stay-point is found, a new state is formed. Once we have the states, we create time-slotted states for each hour as shown in Figure 3. The top layer depicts the incoming location points. The longer stayed locations are tracked down as stay-points as shown in the second layer of the Figure 3. The stay-points are combined, and the states are formed as the next step. These states are then arranged with in their corresponding time-slot which forms the time-slotted data.

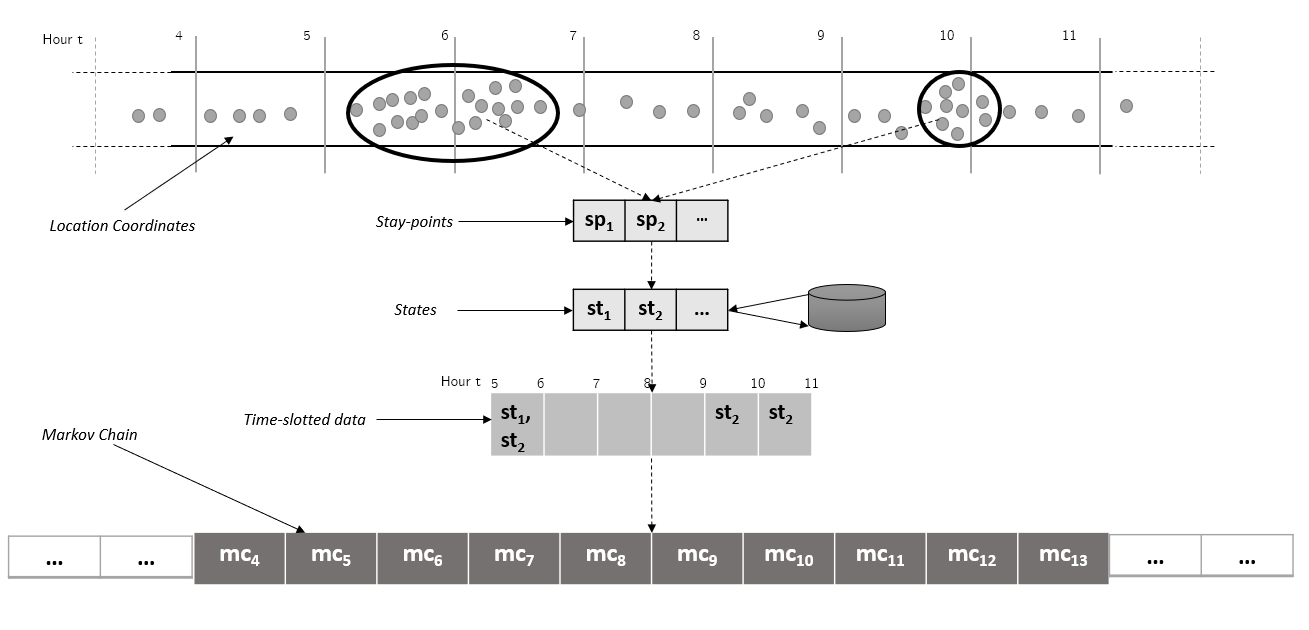


Figure 3 GPS coordinates to Markov Chain Model

Once the states are formed, the time-slotted data is formed and then finally markov chain on these states are formed. The markov chain holds the probability of transitioning from one state to all the other state (including self) from one time-slot to the next. The Figure 4, on the left, depicts two states st1 and st2 markov model. This model involves 4 probabilities *p1* to *p4*. Each of these probabilities represents the transition probabilities from one state to another. For instance, *p1* is the transition probability from st1 and st2. Figure 4, on the right, depicts a third state added to the model st3, which in turn, increase the probability count from 4 to 9. This explains, how the complexity and the computation of the model increase as the number of states increases in the model.

This markov chain model is then used to predict the movements from st1 to all the other states at any given time-slot, based on the probabilities calculated. The similar probabilities are calculated every time a new state is added as depicted in Figure 4 for a new state st3.

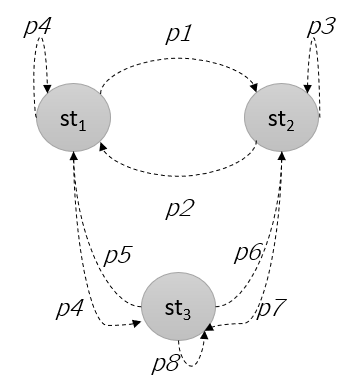
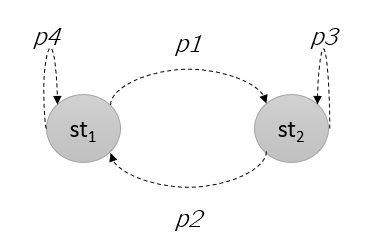


Figure 4 Markov chain on states

### 3.2.3 Hypothesis

The location-based services rely on the assumption that the location is shared for longer periods. The model behaves poorly if the location data is shared very rarely by the user. The occurrences of a few popular locations like “home” and “work” for a user will be more, compared to other location. It is very often that the user stays at “home” location during the night hours and spends more time at “work” location during the day. Of course, there could be night-shifts, but then the duration of stay at “work” location, which usually ranges from 8-9 hours, can help to make the right indications. The “home” location is also often the one which has occurrences during weekends or public holidays. These indicators help us in marking the “home” and “work” location while analyzing the data.

However, the location data is sometimes not available to be shared or just turned-off. For instance, there is no internet in a skyscraper “work” location or it is cloudy day or the location is completely turned-off as soon as the user has entered the “work” location. The next location input, after few hours, is again the “work” location when the user comes in a network coverage area. The first and the last known location help us to fill the missing information during these hours, based on the distance and time difference between the two locations. In this case, we can assume that the user stayed at the “work” location for the missing hours.

It is important to understand that the model proposed in this thesis considers only the significant locations and the transitions between them. Hence the travelling GPS coordinates will be ignored as noise and will not be part of the markov chain model.

## 3.2 Problem Statement

Use of mobile devices is ubiquitous. Users are often using applications on the mobile system which uses their location coordinates. Although the location usage is consensual, the duration of sharing is often not well understood by the users. The applications use the location data in the background when the application is not even in use. According to a Pew survey [11], 91% of Americans agree that they have lost control on their private data and 64% reported that government must regulate advertisers. Another survey reported that only 9% users are confident that social media companies will protect their data.

The threat of location prediction and exploitation of user privacy is to be shown to the users so that the user can make a wise decision before sharing the location with third-party applications on mobile devices.

The goal of this thesis is to simplify the location prediction algorithm on python and test the prediction accuracy. The same algorithm is implemented on an android device. The model of location prediction is built which takes in the location coordinates as input in an online manner and forms a markov chain model. This markov chain model is then used to make predictions based on current location and hour of the day. The location prediction should continue predicting the future locations until the confidence falls below a certain threshold value. The predicted paths are shown to the users to inform about the predictability of his/her whereabouts.

# **Concept and Implementation**

In this thesis, a location prediction model is proposed. The location prediction model predicts the future visits based on the current location. The idea is to raise awareness to users about the privacy risk while sharing the location with applications on mobiles and computers. This chapter briefs markov chain model for location prediction and path prediction overview.

## 4.1 Overview

The markov model for location prediction is formed based on states. The assumption in this chapter is that the states are already created. The state formation from raw location coordinates is explained in further sub-chapters.

Once the time-slotted data is created and normalized, the markov chain is build. The model of markov chain is created to calculate the probability of going from one location to another. The Figure 5 shows on a geographical map how the markov chain looks like. There are four important locations marked on the map. These marked locations are called states. The markov chain model will contain the transition probabilities from one of these important places to all the other places (including self) as depicted in the Figure 5. The arrow depicts the transitions from one location to another (including self). Hence, the model is built using only the important locations. The extraction of these important locations from the raw GPS points is detailed in the further sub-chapters.

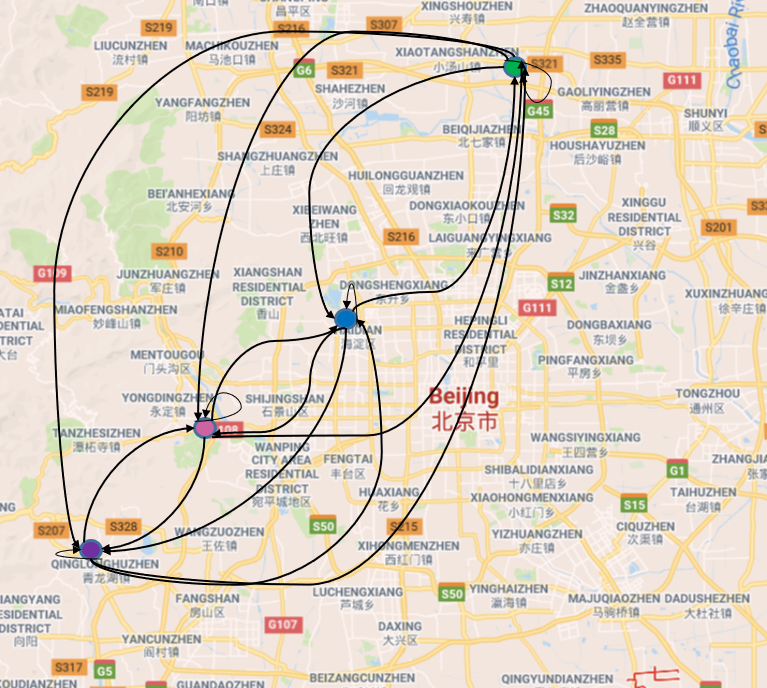


Figure 5 Markov chain example on a map

The transitions are recorded from one time-slot to the next. The time-slots are simplified as hours of the day, hence, there are 0 to 23 time-slots. Hence the transition from hour t to t+1 is recorded. These transitions are recorded over the period (for few days) to create a model.

Once the model is created, the transition probabilities for each time-slot are recorded. Based on the transition probabilities for each time-slot, a path can be predicted. A path is the series of locations visit in the coming time-slots. The path prediction uses the current location, current time-slot and the markov model. The example is extended on a geographical map as shown in the Figure 6. For instance, the model suggests if the user is found to be at state 1 at 7am, the next visits will be at location 2, 3, 4, 3, 1 at corresponding time-slots. The green arrow depicts the transition from state 1 to 4 and the red arrows depicts the transition from state 4 back to 1. Each next visit is predicted with a corresponding confidence percentage which is derived from the markov model. The path prediction is done to indicate the predictability of a user knowing the current location, hour information and past location trends. The path prediction is detailed in further sub-chapters.

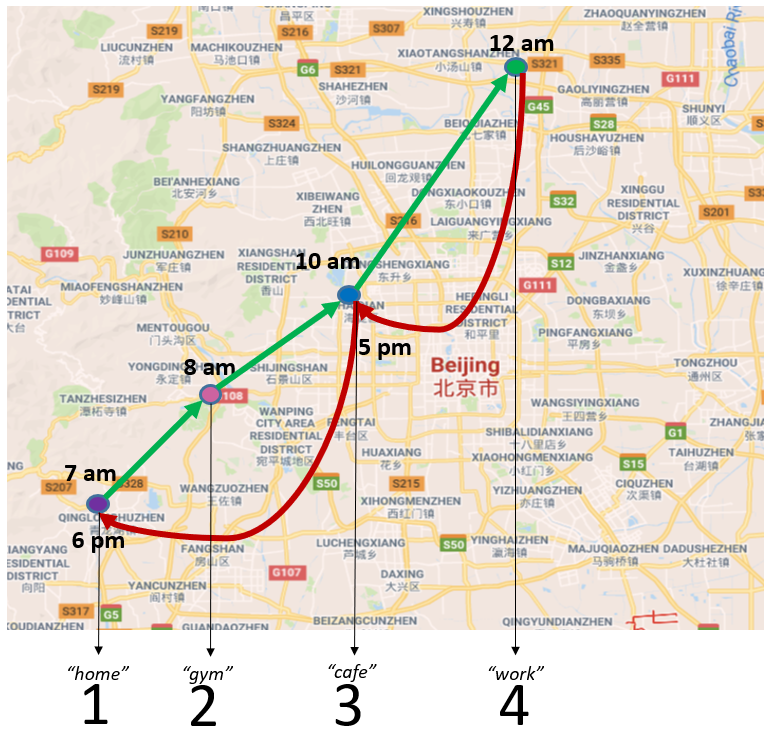
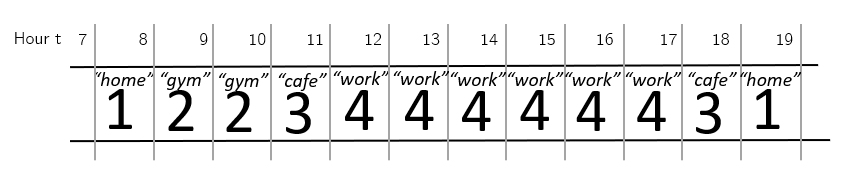


Figure 6 Path modeled using markov chain

## 4.2 Transition Probabilities

The states are the significant locations for a user which are extracted from raw location coordinates. The semantic meaning behind these states are, for instance, “home”, “work” or “gym”. The markov chain holds the probability of transitioning from one state to another, in each time-slot. The time-slots are represented by hour of the day, as shown in Figure 7. Hence, the markov chain records all the transitions from hour t to hour t+1.

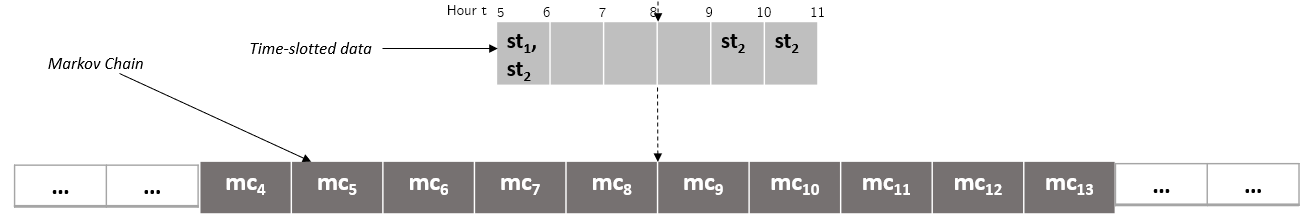


Figure 7 Markov chain derived from state

To understand this in detail, consider an example as shown below in the Figure 8. There are three states st1, st2 and st3 which exists between hour 4-6, 5-9 and 9-11 respectively. We consider the time lot between hour 4-5 as 4 for simplicity. Hence, we have a total of 24 time-slots ranging from 0 to 23 in a day, representing each hour of the day. The vertical drawn lines represent the hour of the day and the width of a state rectangle represents the duration of the stay at that state. For instance, st1 stay duration is at time-slot 4 and 5. The states can exist in more than one time-slot, like st1. There can be many states in one time-slot, like st1 and st2 in time-slot 5.

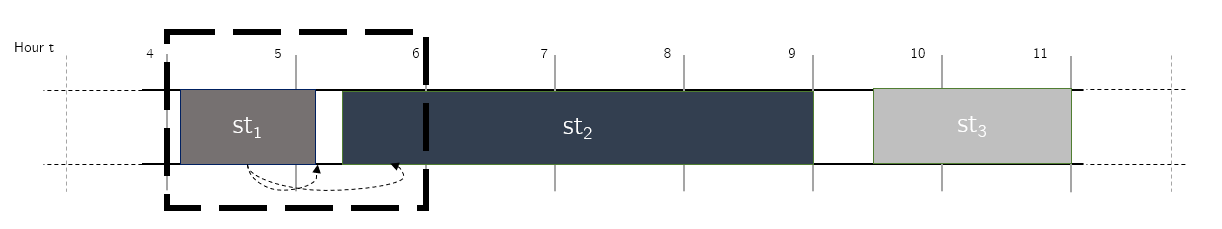


Figure 8 State transitioning

Let us consider the example of transitioning from hour 4 to hour 5 as depicted in Figure 9. In this example, the state transition from hour 4 to hour 5 is from st1 to st1 and st1 to st2, represented by dotted arrows. The state weights represent the duration of the state in that time-slot. The weight of state st1 is *60/60* as the duration of the stay at st1 in time-slot 4 is for 60 minutes. Similarly, the weights of st1 and st2 are calculated in time-slot 5, which is also indicated in the Figure 9.

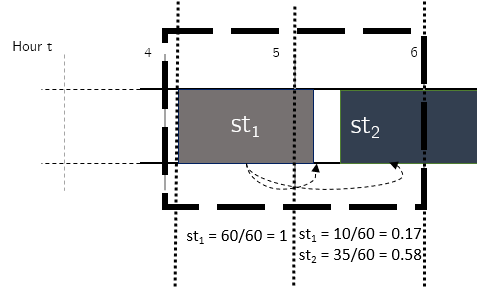


Figure 9 State weights in each time-slot

It is important to mention that the hourly weights are normalized before the markov chain is calculated. The normalization is done by dividing the weights of all the states in a time-slot with the sum of all the weights in that time-slot. Let us consider the time-slot 5 in our example Figure 9. The state weights for st1 0.17 and st2 0.58 in time-slot 5 are divided with the sum both the weights 0.75. After normalization the states are shown in the Figure 10. This is done to smoothen the data in each time-slot. The sum of the weights in time-slot 5 is 1.

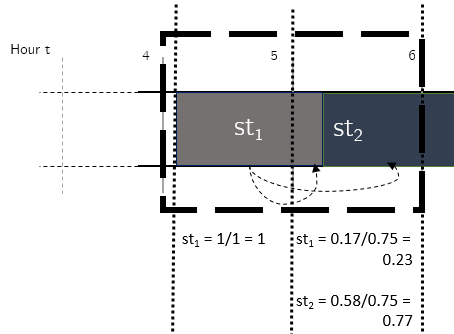


Figure 10 State weights normalized

The probability of transitioning from st1 to st1 from hour 4 to 5 is calculated based on the hourly weight of st1 in hour 4 and hourly weight of st1 in hour 5. Similarly, the probability of transitioning from st1 to st2 from hour 4 to 5 is calculated based on the hourly weight of st1 in hour 4 and hourly weight of st2 in hour 5. The weight vector wt = {w1, w2, … wk} represents the weight of states from state 1 to k for time-slot t. The weights vector w4 = {w1, w2} for hour 4 can be defined as w4 = {1, 0}, where the 1 represents the weight of state st1 and 0 represents the weight of state st2 in time-slot 4. A similar weight vector w for the next time-slot 5 can be represented as w5 = {0.23, 0.77}. The multiplication w4(transpose)\* w5 results into a matrix as represented in the Figure 11. The table represents the transitions probabilities among states st1 and st2 from time-slot 4 to time-slot 5. The first row represents the transition probabilities from state st1 all the other states i.e. st1 and st2. Similarly, the second row represents the transition probabilities from state st2 to all the other states.This transition probability matrix is used to calculate the markov chain model for the given time-slot and including all the states found. The markov chain for transition from hour 4 to hour 5 is called mc5 which is depicted in the Figure 12. The arrows indicate the transition probabilities from one state to another.

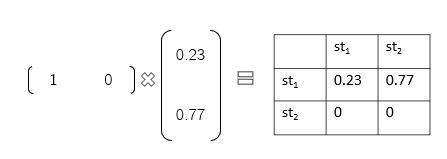


Figure 11 State transition matrix

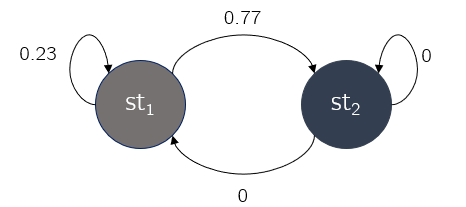


Figure 12 Markov model for two states

## 4.3 State Formation Steps

Using the markov chain model, the future location predictions are done. The predictions are based on user’s current location. Continuing the previous example, if the user is at location st1 at hour 4, there are 77% chances that the user will move to state st2 at hour 5 and 23% chances of staying at state st1 at hour 5. The probability of going to all the other places from his/her current location, at this time-slot, is present in markov chain model.

The states are formed using the raw GPS coordinates. The GPS trajectory points are received as input. For each new GPS Point *P[x, y, d]*, which contains *x* as latitude, *y* as longitude and *d* as datetime, the process is run. The transitions are recorded from one time-slot to the next. The first step is to detect the time-slot change. Until the time-slot is changed, the GPS coordinates are collected and kept in last hour points *lst\_hr\_pts[x, y, d]*. Once, the time-slot has changed or the next hour is detected, *lst\_hr\_pts[x, y, d]* are processed to extract states. There are several steps performed on *lst\_hr\_pts[x, y, d]*  as listed below:

1. **Stay-point extraction:** For every hour or time-slot, the extraction of stay-points extractStaypoints() is run creating the stay-points sp = {sp1, sp2,… spn}. The stay-points represents the location points which are stayed for longer durations.
2. **States:** From the stay-points sp, the states st = {st1, st2, … stn} are formed using formStates(). This process snaps the geographically close-by stay-points to one state. Similarly, several states are formed from stay-points. These states are used later for markov chain model. The states’ st represent “home”, “work” and other important visited places.
3. **State weights:** The next step after state formation is to calculate state weights w = {w1, w2, … wk} in each time-slot. A state weight w1 (minutes/60) represent the hourly duration of the state st1 in this time-slot. These states are later normalized in each time-slot.

This process (A, B, C) is repeated for the entire time-slot. The same steps are repeated for each time-slot and the data is accumulated. This means, the same locations visited are snapped to the same state ids. For example, user has been at location “home” during the early hours of the day. If location “home” is visited again during the day, it will be extracted as a new stay-point in step A. Since “home” location was visited already known, it will be snapped to an existing state in step B.

In the algorithm, the states are represented with a numeric id number. Each of these states can have a semantic meaning like “home”. The last step C is used to calculate the weight, or the time spent at “home” at this new time-slot. This process ensures that we keep snapping the known locations with the same ids. Once the day is changed, the markov model *mc* is created using the state weights w for 24 time-slots. The individual algorithms of stay-point detection, state formation, state weight calculation and markov chain model creation is explained in further sub-chapters.

## 4.4 Implementation

In this subsection we introduce the implementation of each component of the model. The subsections include the explanation of each component in detail and the corresponding algorithm.

### 4.4.1 Variables Used

Table 1 is to provide an overview of the variables used in the further sub-sections. The variables are used in algorithms and in explanation. The list of variables covers majority of variables used in further sections, but the list is not exhaustive. Few new variables are introduced and explained in the coming sections for clear understanding of concepts.

|  |  |
| --- | --- |
| Variable | Description |
| *P[x, y, d]* | GPS raw coordinate point:  (Latitude, Longitude, Datetime) |
| *lst\_hr\_pts[x, y, d]* | Last hour GPS coordinate points  (Latitude, Longitude, Datetime) |
| *sp* | Stay-point |
| *st* | State |
| *w* | State hour weights |
| *mc* | Markov Chain |
| *th\_tck* | Threshold time for tracking GPS location data |
| *th\_d* | Threshold distance for staypoints |
| *th\_t* | Threshold time for staypoints |

Table 1 Variables used in algorithms

### 4.4.2 Stay-points

Stay-points are those important places where user spends ample amount of time. The places like “café”, “gym”, “restaurant”, etc., are identified using distance and time-based clustering. Distance and time-based clustering work best in case of location data as explained by the authors [6]. This clustering approach is not so complex and can be run on a mobile device as a background process. The clustering has two thresholds, one for distance *th\_d* and one for time *th\_t.* These threshold help determine the stay-points in an online fashion. The location points within the radius of distance threshold *th\_d* and time spent at this location are greater than or equal to the time threshold *th\_t*, is regarded as a stay-point. For example, a set of points within 50m of radius and total duration of stay greater than 10 minutes, can be regarded as a stay-point. In the example above, the café location will be a stay-point only if the stay is greater than or equal to the time threshold *th\_t*. This help to remove noise, like travelling GPS coordinates or short stay locations. Hence, only significant locations from the trajectory are extracted and noise is removed.

#### 4.4.2.1 Sporadic GPS Trajectories

A GPS trajectory is the path taken by the user where the GPS points are continuously received i.e. every 5-10 seconds a coordinate is received. A trajectory can end for several reasons, for example, if the user turns off the phone or turns off location sharing, or the user enters a low network area. The clustering approach works adequately if the GPS points are received continuously. But if the location is shared only at intermittent intervals, the extraction of all the important locations is incomplete. This type of GPS trajectories contributes to the sporadic GPS trajectories.

Stay-points are any points which are stayed by the user during the user GPS trajectories or it is the start or the end of the GPS trajectory. For example, if the user starts his/her trajectory at “home” location, the “home” itself is a stay-point. Now he moves towards “work”, but he visits a “café” in between for breakfast. The “café” is also, a stay-point and then he finishes his trajectory at “work”, where “work” is again a stay-point.

Trajectories are continuously receiving GPS points. The gap of time greater than tracking time threshold *th\_tck* between two GPS points breaks the old trajectory and starts a new one. This means, if the location coordinates are received continuously for few hours, then the location coordinates are stopped, therefore ending this trajectory. As soon as the new location coordinates starts, a new trajectory will start. Note that the stay-points, within the trajectory, are found using the time and distance clustering algorithm [6]. The second type of stay-points also exists. These stay-points are the locations where user has ended or started his/her trajectory. For instance, the user has entered his “work” location and now he/she do not share his/her location. If the next shared location is after a threshold tracking time *th\_tck*, the last trajectory is broken and a new has been started. This means, if there have been no new location coordinates received for a given time, the last point in the previous trajectory is added as a stay-point and so is the next point received consequently in the next trajectory. This ensures that the important locations are not missed even if the location data is not present. Since, the location data is often turned off at stable locations like “home” and “work”, this algorithm makes sure that these locations are not missed for future user movement analysis.

#### 4.4.2.2 Estimating Arrival and Departure times of Stay-points

After the collection of stay-points, the stay-points entering and leaving time is recalculated. This is done to estimate the time of leaving a stay-point and the time of entering the next stay-point.

*Case 1:*

Let us understand this using an example as depicted in Figure 13. In this example, we consider the distance threshold *th\_d* to be 200 meters or 0.2 kms. For instance, user is reported to be at “home” location at 7am and then the next stay-point is found to be “work” location at 8 am. The departure time from “home” and arrival time at “work” can be estimated.

Furthermore, in Figure 13, the distance between these two locations is *x* kms, which is easy to calculate as the “home” location coordinates and “work” location coordinates are known. Let’s consider the distance *d* to be 6 kms between “home” and “work” location. The time difference *t* between the two points “home” and “work” is also known, which is 1 hour in this example. This information helps us to estimate the actual leaving time from “home” location and actual arriving time at “work” location. The speed of user *spd* can be calculated as (*x* kms / *t* minutes) i.e. (6/60) kms/mins or 0.1 kms/mins. Now, the delta time is calculated as minimum(*th\_d*, *d*)/*spd* i.e. minimum(0.2, 6)/0.6 or 0.33 minutes. This delta time is added in the departure time of “home” location and subtracted from the arrival time of “work” location. So, the estimated departure time at “home” location is 7am + delta time i.e. 7am + 0.33 minutes and the estimated arrival time at “work” locations is 8am – delta time i.e. 8am - 0.33 minutes.

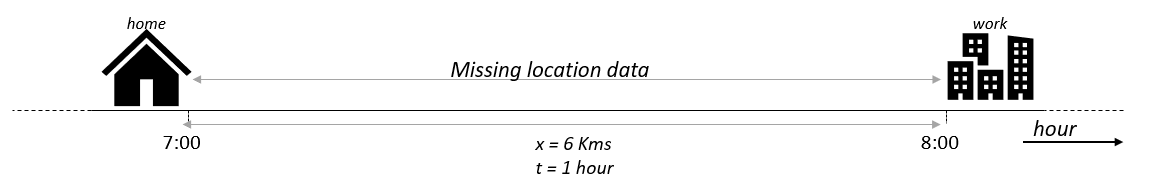


Figure 13 Example 1 of missing data

*Case 2:*

Although, there could be the case where the user is not travelling or moving from one location to another, but rather the user stays at a location after some missing data. Consider the example shown in Figure 14 where user has shared his/her location at “home” at 18:00 and now the location data is not shared for some reason for the next few hours. The next location shared is again “home” location at 06:00 the next day. The missing location data is most likely the “home” location for the entire time-slots between 18:00 on this day till 6:00 on the next day, Since the distance between these two points is 0, the speed *spd* of travel will also results in 0 km/hour. Now, the estimated time of being at “home” location is recalculated. The time difference between the two known points is 12 hours. The time of leaving “home” location is recalculated as (18:00 + 12/2) i.e. at 00:00 on this day and the time of arriving at “home” location for the next day is (06:00 – 12/2) i.e. at 00:00 on the next day. In other words, it is estimated that the user stayed at “home” location for this period.

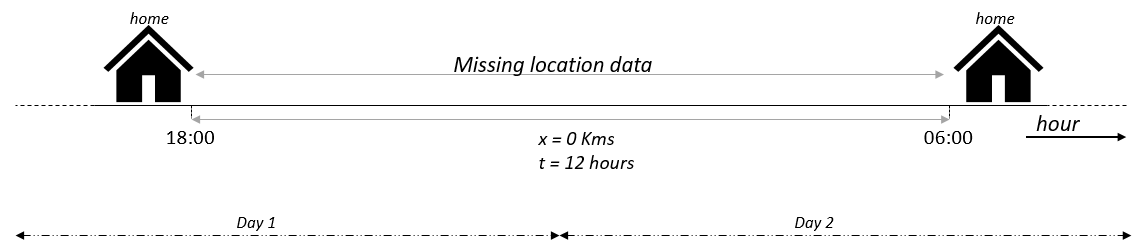


Figure 14 Example 2 of missing data

#### 4.4.2.3 Algorithm

The stay-points are extracted from raw points to remove the noisy points. The noisy points could be travelling with the bus or train or a short stop at the letter box. The stay-point extraction is the process of extracting longer stayed locations from raw GPS trajectories. The Figure 15 shows the transition from “home” to “work”. In this case, both “home” and “work” are extracted as stay-points. The two bigger circles denote the stay-points and the smaller dots within represents the GPS points. The radius of the stay-point circles is the distance threshold distance *th\_d*.

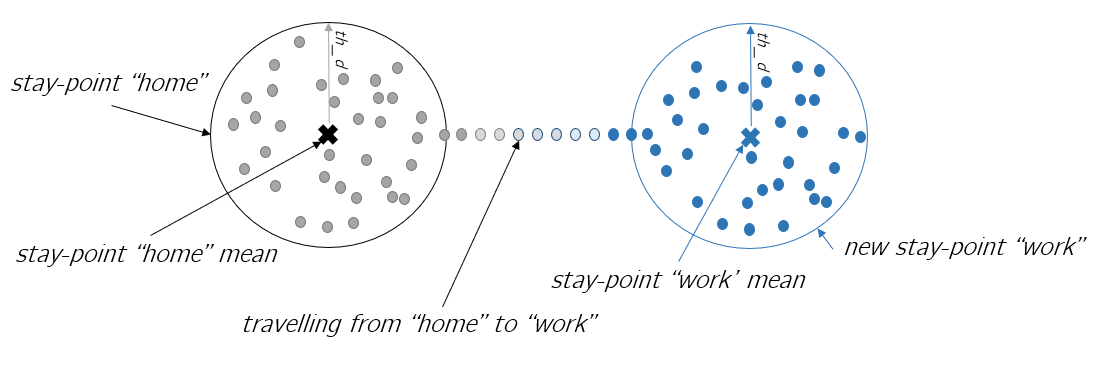
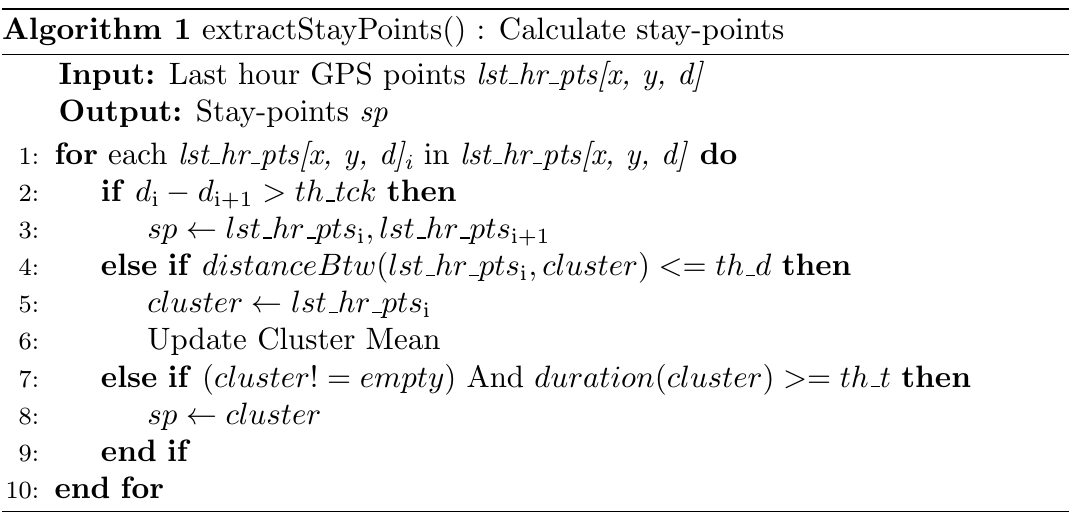


Figure 15 Extracting stay-points from GPS Trajectories

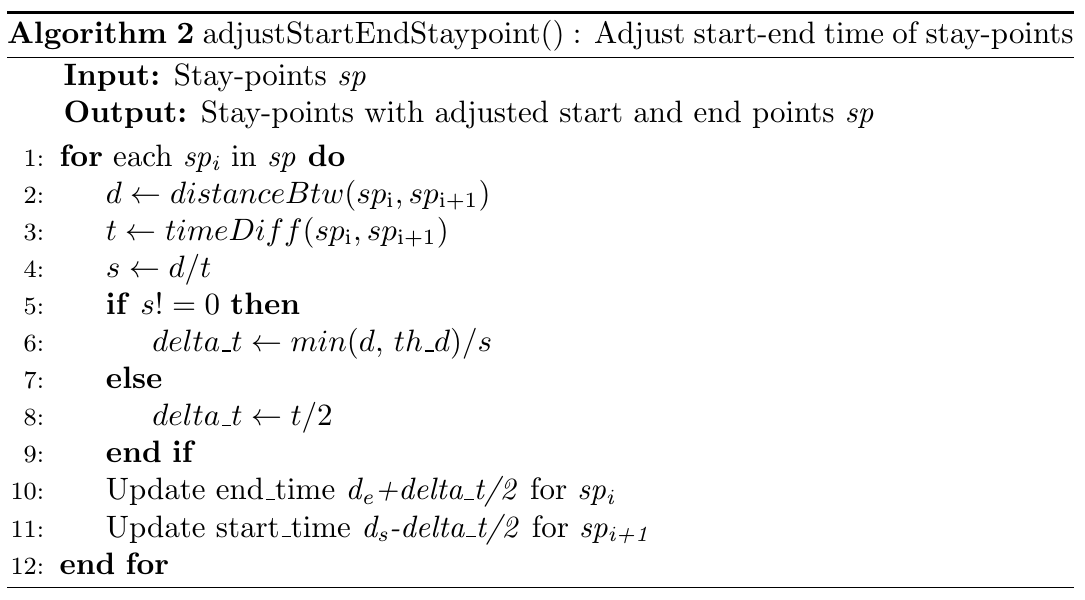
The extraction of stay-points takes *lst\_hr\_pts* as input and generates stay-points sp = {sp1, sp2,… spn} as output. The algorithm cluster the points within the radius of stay-point distance threshold *th\_d* for a minimum duration of time *th\_t*. The selection of distance and time threshold is very important. If the distance threshold value is too large, the mean of the stay-point locations will be a confusing location on the map and many unimportant locations will be marked as stay-point. If the time threshold value is very small, a lot of insignificant locations with short stay durations will be added as stay-points. A balanced selection of distance and time threshold can range from 100-300 meters and 10-30 minutes respectively.

Algorithm 1 explains the approach used for stay-point extraction. The GPS location points *P[x, y, d]* are received and stored in last hour points *lst\_hr\_pts* until the next hour. The points from *lst\_hr\_pts* is read and the points which are close-by are added to the same cluster. A new location from *lst\_hr\_pts* is added to the cluster if the distance between the new point and the cluster mean is less than or equal to the distance threshold *th\_d*. Every time a new point is added to the cluster, a new mean of the cluster is calculated and the process repeats. If the new point from *lst\_hr\_pts* is moving away from the cluster mean, then the point is not added to the same cluster. This means, that if the distance between the mean of the cluster and the new point from *lst\_hr\_pts* is greater than threshold *th\_d*, then the new point is not added to the same cluster. At this point, this cluster duration is checked. The cluster duration is nothing but the largest datetime – smallest datetime from the cluster elements. If the cluster duration is greater than or equal to *th\_t,* then the cluster is added as the stay-point sp with latitude and longitude as cluster mean, otherwise the cluster is not added as a stay-point.

The new point from *lst\_hr\_pts* is also added as a stay-point if the difference of time between the new point and the previous point is greater than time tracking threshold *th\_tck*. This is to ensure that if the GPS points are not received for a long time, we add the last and the new point as a stay-point assuming the end of the previous trajectory and the start of the new trajectory.



The stay-points are often not continuously distributed over time. Consider a scenario where a user is at “work” location till 9 am. After entering “work”, there are no points received for next few hours. The next stay-point detected is “work” location at 1 pm when user came out for lunch from the building. The time of stay at “work” in this case till 9 am is misleading as user stayed at this location till 1 pm. Hence, once the stay-points are collected, we adjust the starting time and the leaving time of each stay-point.



The algorithm 2 details how to achieve this. A comparison for each stay-point in sp = {sp1, sp2,… spn} to it’s very next stay-point is done. Now, the distance d and time difference t between the two stay-points spi and spi+1 is calculated. Using distance d and time t, the average speed s of travel can be easily calculated, which is d/t. If the speed s is not 0, the delta time delta\_t is calculated as division of minimum of distance between spi and spi+1 d and threshold distance th\_d, to the average speed s. If the speed s is 0, this means that the user has not displaced from the previous location and hence the time difference t is filled with the same location. Now we add the delta time delta\_t to spi end time, to change the leaving time at the spi location and subtract delta time delta\_t for spi+1 start time to change the entering time at location spi+1.

### 4.4.3 State Formation

A state is formed using a group of stay-points represented as s = {st1, st2, …stn}. Each state has a semantic meaning like “gym”, “restaurant”, “home” or so on. The states are formed to join the stay-points which are geographically close-by as one state. The states are found using a distance threshold. All the stay-points within this threshold distance radius is grouped together as a single state. This is called snapping stay-points to the states. This is depicted in the Figure 16. The two stay-points are snapped to states to form st1. The first stay-point between hour 5 and 7 is assigned to a state st1. When the second stay-point between hour 9 and 11 is extracted, the distance between the stay-point and the state st1 is checked. If the distance between them is less than the threshold distance *th\_d* , the stay-point is also snapped to the same state st1.

The mean of latitude and longitude of all stay-points, forming the state st1, is stored. This makes sure that if a known location is visited after few days, then it should get the same id as given before. Imagine a case where the user visited a restaurant last Monday and visits the same restaurant again the next Monday. In this case, the restaurant visits during next Monday should get snapped to the already existing state from last Monday. Finally, markov Chain model is applied to these states.

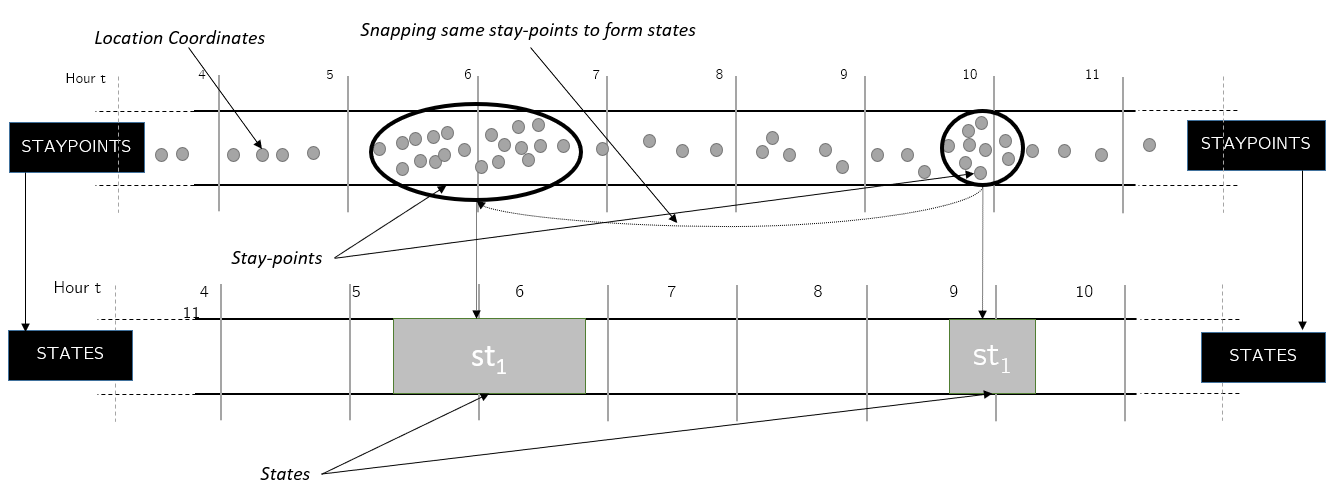


Figure 16 Snapping stay-points to states

#### 4.4.3.1 Drifting Problem

The stay-points are snapped to an existing state with an exception. The Figure 17 depicts the drifting problem. The current mean of state sti is marked with **×**. A new stay-point marked with a red dot on the right has arrived. The addition of the new stay-point spj to this state will make the mean of the state shifted, denoted by **×**. The new mean of the state sti will throw some of the existing points from left out of the state radius. Therefore, the new stay-point spj is added to a new state in this case. Hence, the idea is, while adding the new stay-point to an existing state, a check is done. If all the existing stay-points stays within the radius of the state mean, then the stay-point is snapped to the state, otherwise a new state is formed. In other words, if any of the existing stay-points contributing to the state formation is moving out of the state radius, then the new stay-point is not added to this state to avoid the drifting problem.

#### 4.4.3.2 Algorithm

Algorithm 3 explains the details of state formation from stay-points. Each new stay-point goes through this step. When a new stay-point is extracted and added to the stay-point sp = {sp1, sp2,… spn, spn+1}, the stay-point spn+1 distance is compared with all the existing states in st = {st1, st2, …stn}. If the distance between a state and the new stay-point spn+1 is less than the distance threshold *th\_d*, then the stay-point spn+1 added to the that state. Otherwise, a new state is formed.

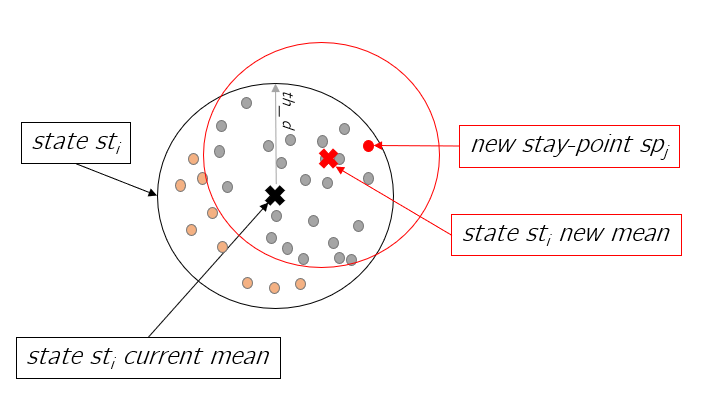
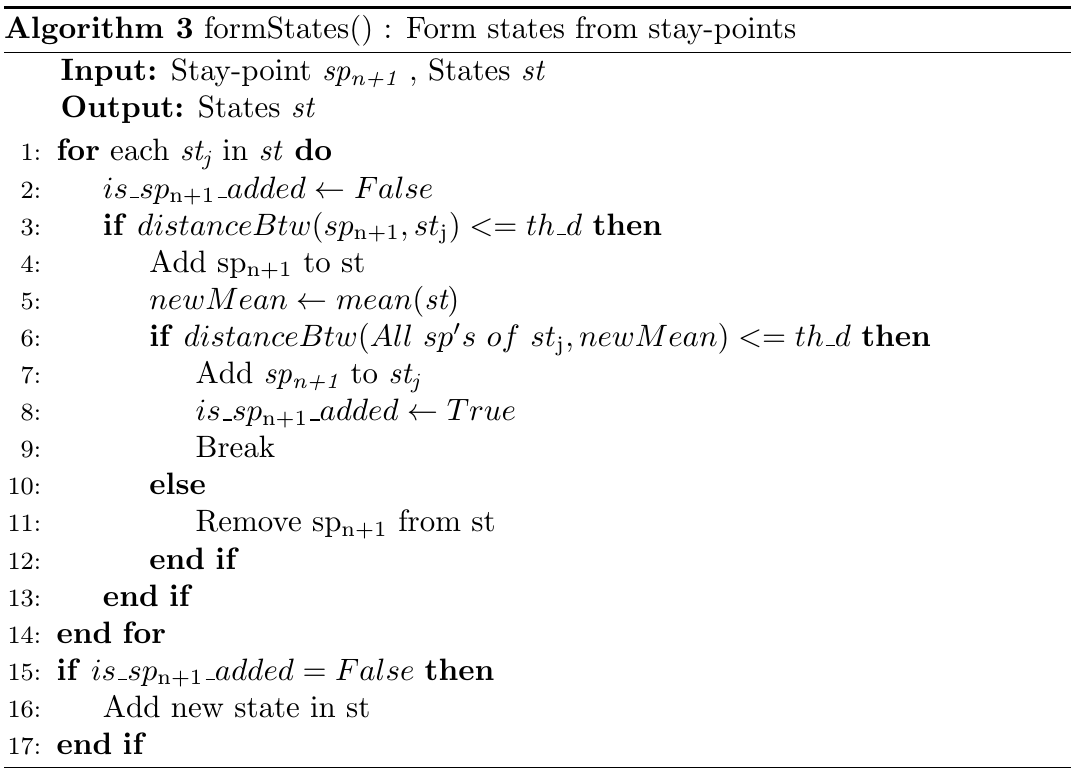


Figure 17 Drifting problem while snapping stay-points to the states

#### 



### 4.4.4 State Weights

Once the states are formed from stay-points, the next step is to create time-slotted data. The time-slotted data is nothing but the state weights in each time-slots. In our case, time-slots are divided into hours of the day, hence the hourly weights form the time-slotted state data.

The states st = {st1, st2, …stn} are used for calculating the weight w = {w1, w2, … wk} of each state in each time-slot. It is important to know that there can be many states in one time-slot. The weight wi represents the stay at state sti in a time-slot. Since states itself carry some semantic meaning like “home”, “work” or others, the weights w represents how long did the user stayed at “home” in this time-slot.

The same state can appear in different time-slots. The set of states st = {st1, st2, …stn} are assigned with weights at each time-slot to form wt = {w1, w2, …wn} for timeslot t, where w1 to wn represents the weights of states st1 and stn respectively. The hourly weights are the ratio of minutes spent at a state sti to the total minutes in one hour i.e. 60. The weights are then normalized to smoothen the data in each hour slot. As shown in the Figure 18, the st1 hourly weight is calculated for 5-6, 6-7, 8-9 and 9-10. The state st1 location is recorded for 40 minutes between hour 5 and 6. Hence the weight for st1 is calculated as 40/60 = 0.67. Similarly, the weights are calculated for each time-slot, for all the states. In this example, we have only state st1. After the hourly weights are calculated for the state, the weights are normalized in each hour before they are used for markov chain calculation. This is also shown in Figure 18, where the state weight between hour 5-6 changes to 1. This is done by simply dividing the weights in weight w = {w1, w2, … wk} with the sum of all the weights in w.

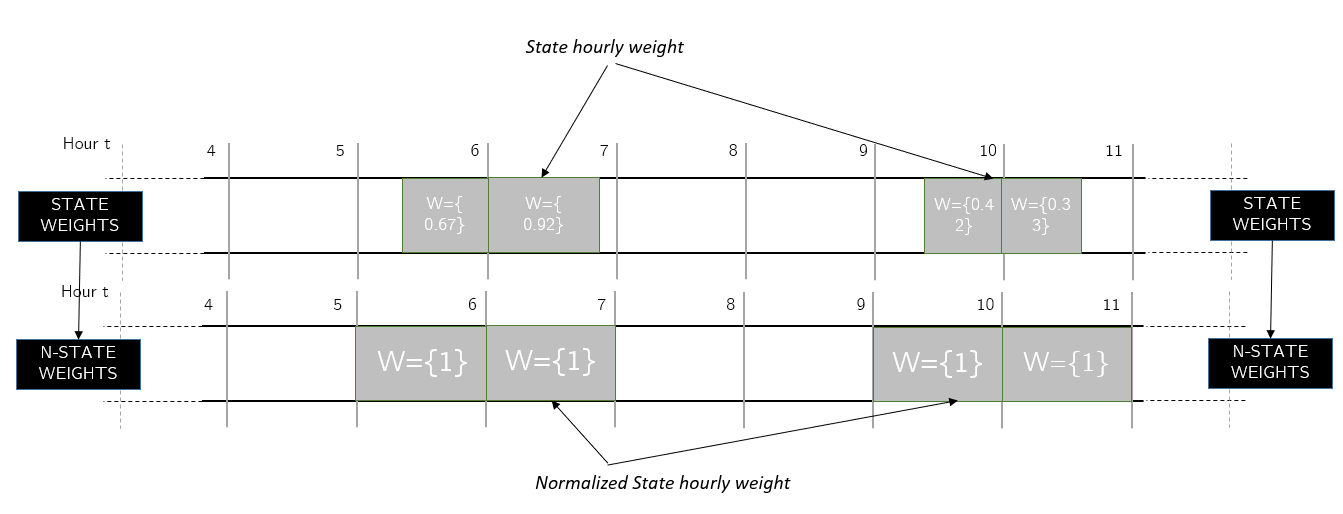


Figure 18 Normalizing state weights

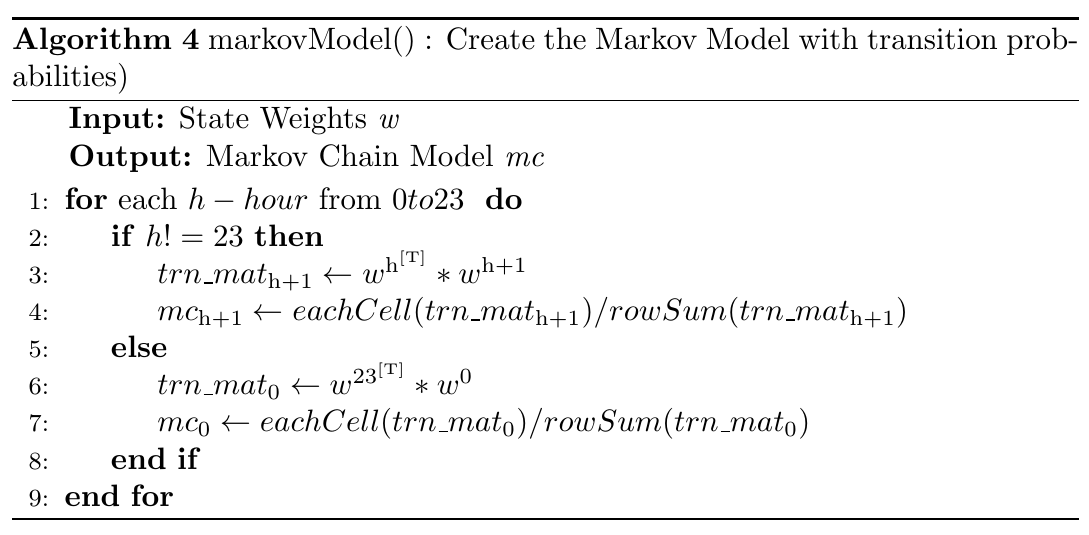
### 4.4.5 Forming Markov chain

Creating the markov model involves calculation of the transition probabilities. The transition probabilities indicate the chances of transitioning from one state to another. A set of n states’ st = {st1, st2, …stn} where each state represents a significant place or place of interest, with weights wt = {w1, w2, …stn} at hour t and with weights wt+1 = {w1, w2, …stn} at hour t+1 is used to calculate the transition probabilities. The weight vector Wtand Wt+1 are multiplied to calculate the transition probabilities from each state in set st to every other state in st from time-slot t to t+1. A transition can be a self-transition, for instance, st1 to st1 or a transition to another state, for instance, st1 to st2. These transitions represent human mobility from one important place like “work” to another like “gym”. The transitions are recorded for each time-slot.

The markov chain built is standard and the probability only depends on the current location and not the locations before. The model is also built on states which are only the important places. Hence the travelling GPS coordinates are not part of the markov model. The model is updated after each day to keep a track of all the new locations user has visited in the previous day and update the mobility pattern. The continuous update of the markov chain model helps to track the changes in user behavior. For instance, user may change the work location, working hours, join a new gym or move to a new city.

#### 4.4.5.1 Algorithm

Algorithm 4 describes the creation of markov chain model. After state weights w = {w1, w2, … wk} are calculated, the next step is to build the markov chain model mc. The markov model mc contains the probability of transitioning from one state sti to another state stj for time-slot t to t+1. This information is stored for each state in st = {st1, st2, …stn} transitioning to every state in st for all time-slots. The time-slots in this case represents the hour of the day. For each hour of the day, the transposed weights wiis multiplied with the weight wi+1 for hour i to i+1. This gives the transition probabilities matrix trn\_mati+1. Now the transition matrix trn\_mati+1 is used to calculate the markov chain by dividing each cell element of the matrix with the sum of the row. This converts the cell elements into probabilities. The time-slot t and t+1 weights are used to calculate the transition probabilities and kept in time slot t+1. Hence, for the last time-slot of the day, i.e. 23, the weight for time-slot 23 and the weight of time-slot 0 for the next day is used.



#### 4.4.5.2 Approach with Additional Steps

Consider the Figure 19 as an example. The weights of state st1 and st2 for hour 4 and 5 are depicted in the figure. There is a transition from hour 4 to 5 from state st1 to st1 and another transition from st1 to st2. The corresponding state weights represents the normalized duration of stay in that time-slot. Hence, the w1 is 1 at time-slot 4, because the entire stay is at this time-slot is at state st1. w1 is 0.33 (20/60) between time-slot 5 and 6 as the stay at state st1 between 5 and 6 hours is 20 minutes. Similarly, the weights of the other states are calculated.

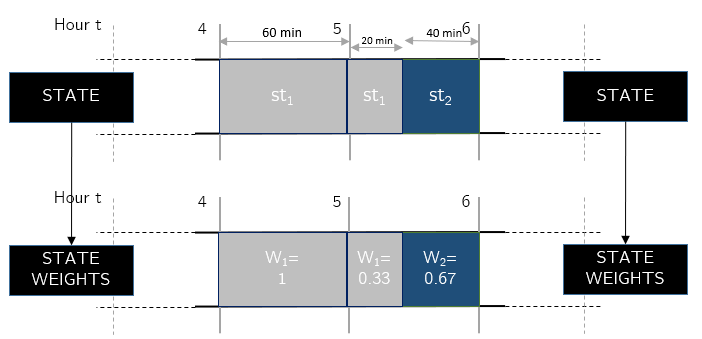


Figure 19 Forming time-slotted data from states

Once the state weights are calculated and normalized, the same weights are used to build the transition matrices for each time-slot. The weights are multiplied from one time-slot to another to create the transition matrices. On the left of the Figure 20, shows the transition matrix for time-slot 5, build for the previous example. The rows and the columns are representing each state and each cell represent the transition from one state to another.

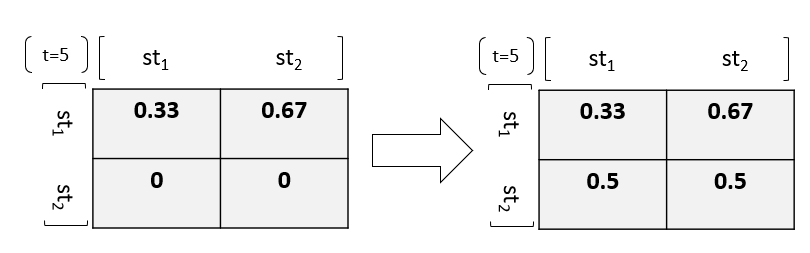


Figure 20 Transition matrix for state 1 and state 2 for time-slot 5 (left without and right with addition of the dummy values)

Once the transition matrices for each time-slot are calculated, the markov chain model is build. In markov chain model, transition probabilities from one state to the another is calculated, which is done using the transition matrix. Each row in the transition matrix represent the transition from one state to all the other state. In the Figure 20, the first row represents the transition from st1 to all the other states i.e. st1 and st2. To calculate the probability of transitioning from st1 to all the other states can simply be calculated by dividing each cell elements with the sum of the row. If the sum of the row is zero, this indicates that we have no information about this transition. In this case, equal probabilities are assigned to all the states. This is shown on the right of the Figure 20, where the transitions from state st2 are assigned with equal probabilities.

It is important to understand that all the 0 transitions probabilities are replaced with a small probability. This is done to ensure that the chances of transitioning from a known state to another known state, is never 0. The Figure 21 explains the process of replacing the zero probabilities. The first step shows the transition matrix received, as explained above. In step 2, 0 is replaced by a 0.00001, a small weight. In step 3, each cell value is divided by the sum of the row, making the transition probability from state st1 to st3 non-zero.

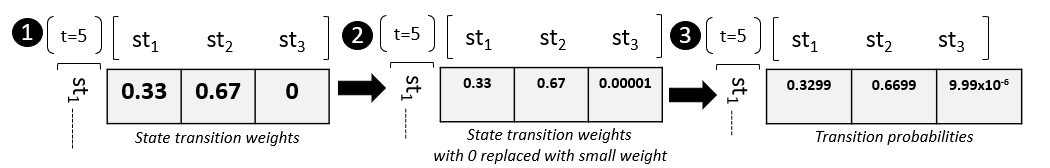


Figure 21 Steps followed to smooth the transition matrix

### 4.4.6 Path Prediction

The path prediction is to inform the user about his/her predictability. The idea is to predict the several paths that user may take, from a known location, in future time-slots. For example, user’s current location is known to be at “work” at 8 am. Knowing his/her location data for few weeks, the markov model is built. The path prediction can be done using markov chain model. The model is memoryless and hence depends only on current location. This means, the prediction is done based on the user’s current location and is independent of the locations before. Now this markov model is used to predict all the locations in consecutive time-slots. One of the paths predicted could suggest that he will stay at work till 5pm with very high confidence. Another path could suggest that he will go back to his “home” at 3 pm with medium to low confidence. Each next location is predicted with a confidence percentage. There could be several such paths that could be predicted.

The paths are predicted only based on the markov chain model. If the model is not recent or is built with very limited data, the predicted paths may not make much sense. For example, the markov model is build based on the 2 days data where user transitioned between “home”, “shopping malls” and “restaurants” only. The two days do not include “work” location. It could have been a weekend, public holidays or user has taken vacation. Now, on third day, if the paths are predicted, only these locations are taken into consideration and the paths could be misleading if user started working from day 3. Hence the data for markov chain should be enough and recent to make improved predictions.

#### 4.4.6.1 Approach

The markov model contains the transition probabilities from one state to all the other states. The states st = {st1, st2, …stn} has corresponding probabilities p = {p1, p2, … pn} in a time-slot. The state probabilities are used for path prediction. For a path prediction, a known start state and a time-slot is required as input. In other words, the algorithm needs to know a location and hour, to be able to predict future visits. The prediction stops one step after the confidence drops less than the confidence threshold.

Consider the Figure 22, as an example figure for markov model. The start state is i with a probability of pi at time-slot t. The predictions are made for the future time slots i.e. t+1 and further. At each time-slot, there exists probabilities for all the other states, denoted by p = {p1, p2, … pn}. The confidence percentage at any state, at any time-slot, is calculated by multiplying it with the probability of the state at this time-slot with the state probability at the previous time-slot. Hence at time slot t+1, the confidence to be at state st1 is calculated as pi\*p1. The confidence calculated should be more than certain threshold to continue the prediction. The algorithm selects the next visit in each path after the confidence drops below the threshold value. This is done to indicate the confidence drop right after the last selected state.

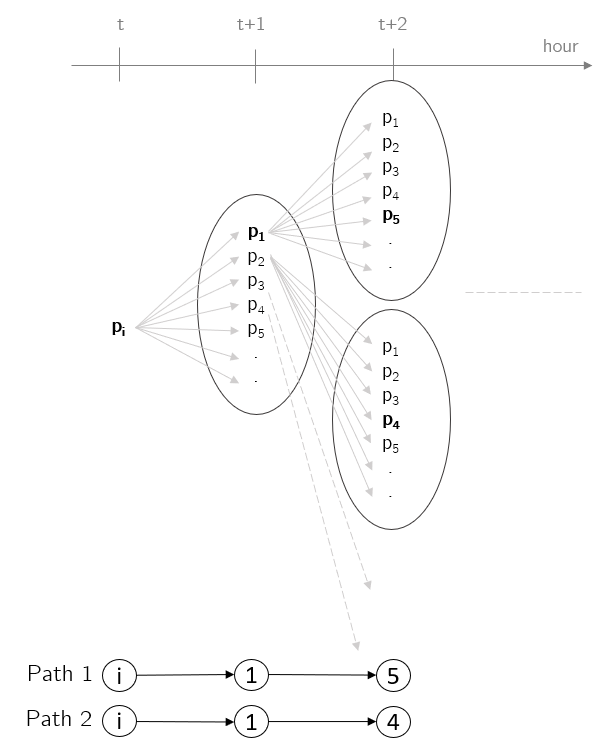


Figure 22 Path Prediction Model using Markov Model

It each step, the probabilities from the next time-slot are put into a priority queue and the with one with the highest confidence is the one preceded with. The selected paths are all the states selected with confidence greater than the threshold. The confidence reduces as we go further from the start time-slot. In the Figure 22, the selected paths are i-1-5 and i-1-4. This means the final confidence calculated for path i-1-5, which is pi\*p1\*p5 is greater than the confidence threshold and for path i-1-4, which is pi\*p1\*p4 is also greater than the confidence threshold. The process continues till the confidence drops below the threshold value, with an additional state for each path indicating the drop in the confidence.

#### 4.4.6.2 Proposition

The path prediction algorithm is to suggest a user for his/her predictability. The algorithm confidence reduces as we go further away from the starting time-slot. This is because of the way markov chain model works. The model keeps a record of each transition.

The human may have different expectations for their predictability. Most of us ignore the infrequent and minor transitions. For instance, the exact place and time for the lunch 1 week ago is easily forgettable. But markov model record each of these transitions. This affects the predictions made during these hours where several infrequent transitions take place. Hence, markov model also need to forget these minor transitions to have the same results as expected by the users of the application. This is called as applying memory-loss factor to the markov model.

The memory-loss factor *mlf* can be calculated and applied to the markov model. This can be implemented if the probabilities are reduced by a factor and then normalized again. This will strengthen the higher probabilities and diminish the lower probabilities completely. The process is explained with the help of a the Figure 23 . It is important to remember that the markov chain model contains probabilities of transitioning from one state to another. In step 1, the relevant probability vector is extracted. In step 2, the memory-loss factor *mlf* is subtracted from each of these probabilities and if the subtraction results into a negative number then it is zeroed. In step 3, the vector is renormalized by dividing each element with the sum of the vector.

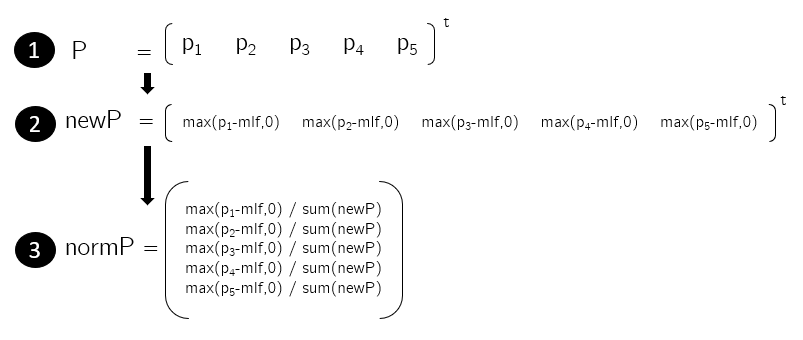


Figure 23 Steps to apply memory-loss factor to the markov chain

Figure 24 depicts an example with memory-loss factor *mlf* as 0.05. When this factor is applied to the probability vector P, the stronger probabilities like 0.79 is strengthened to 0.91 and all the other probabilities are reduced. The very small transitioning probabilities like 0.03 are zeroed.

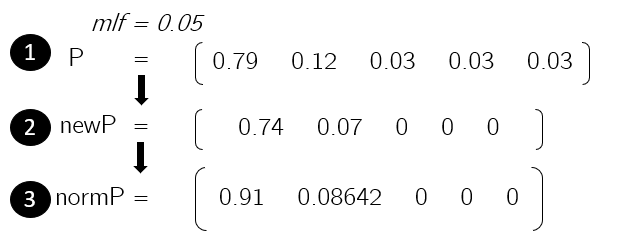


Figure 24 Example of application of memory-loss factor

The application of memory-loss factor will make the stronger probabilities stronger and the weaker probabilities weaker. This is how a user of the application will also expect. The most significant places like “home”, “work” or “favorite restaurant” are remembered by the user more often than the fast food restaurant or the petrol pump stop visited last week. The path prediction can use the markov model with the memory-loss factor to make meaningful path predictions.

# **Android Implementation**

The implementation is done on an android phone as a prototype. This is to showcase how the mobile devices receiving location data can exploit the data and make predictions. The markov chain algorithm is the same as explained earlier. The data is also from Geolife dataset for prototyping purpose. In the actual scenario, this data will be fed from GPS to the application.

The android implementation has challenges like computational cost. The implementation is similar as the python implementation. The user can see the prediction results as path with corresponding confidence levels.

## 5.1 Objective

The intention is to have an operational android application which can make path predictions and showcase the privacy threats of location-based services. The locations predicted in each step should be with certain confidence percentage. The visualization of the predicted paths should be understandable to the user.

## 5.2 User Interface Design

The user interface comprises of 3 sections. Each section consists of an independent screen for user interface.

**Screen 1:** The start screen of the android application, as shown in the Figure 25, has two modes “GPS” or “Geolife User Data”. The mode “Geolife User Data” is to use the Geolife user dataset. Once the user has selected “Geolife User Data”, the option of choosing user and month is made visible. The user and month must be selected from this input screen to be able to go forward. The “GPS” mode will be used in the actual application usage scenario where the user location details will be read for few weeks.

**Screen 2:** Once the user has made the selection and hit “Continue” button on screen 1, the markov chain is built in the background from the user data file. The states are extracted which represents the significant places for the user. These states are displayed as shown on left of the Figure 26. The states may not be the exact positions user visited, but rather in the vicinity, as these states represent the mean of several coordinates within a range. The list is displayed to show the user all the visited placed that has been tracked down as important places.

**Screen 3:** Once the user proceeds by pressing the button “Find Predictions”, user is taken to the final prediction screen. Here user can select from the list of states and hour from the screen. The selection is to prototype if the user’s location was found to be at this selected state location at this selected hour, which paths could be predicted? The confidence threshold input can be changed using the slider. This threshold controls till when the paths are predicted. The confidence threshold ranges from 10% to 100%. A confidence threshold as low as 10% can generate many longer paths and a higher confidence threshold as 90% can produce fewer shorter paths. Once the selection is done, the button “Run” can be hit to produce the predicted paths. The paths are displayed below as shown in the Figure 26 on the right. The paths are displayed with the help of states. The states confidence is represented by the help of the font color. The darker the state font color depicts higher the confidence and vice-versa. In the example Figure 26 on the right, two paths are predicted. The starting hour and the ending hour are displayed at the beginning and the end of the path respectively. In below example, the known state or location is 3 at hour 0. The predicted paths start from hour 1 till hour 15. Path 1 suggests that user will transition from state 3 to state 6 and will stay at state 6 till hour 15. Whereas path 2 suggests that user will stay at state 3 for longer duration. It is also easy to spot that the prediction confidence of path 2 drops earlier (as the color starts to fade away quite earlier) than that of path 1. Hence the prediction output path 1 is more confident than prediction path 2.

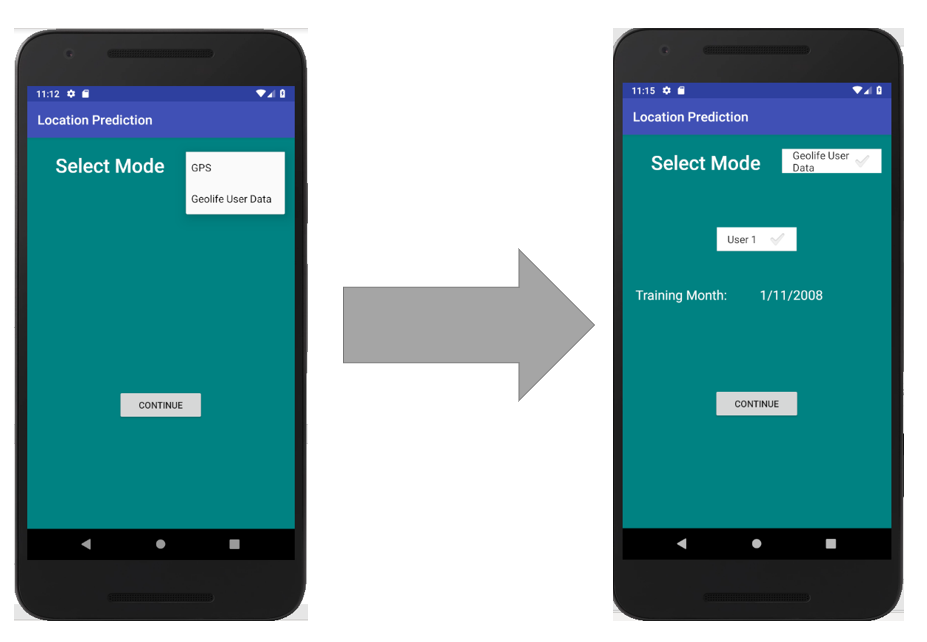


Figure 25 Android application screen 1

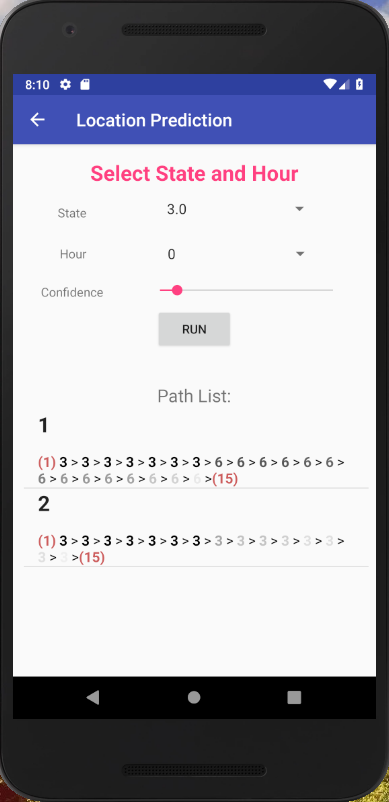


Figure 26 Android application screen 2(left) and 3(right)

## 5.3 Future Work

The android application is implemented as a prototype. The user interface can have several potential improvements. The Geolife dataset for users is smoothened using python and is fed as input to the application. The data fed as input is the time-slotted data. For further analysis, the application must be modified to accept the raw GPS coordinates and create time-slotted data. This process should be implemented as a service and should run on the background when the application is not being used. The markov chain and path prediction algorithms are implemented on android application independently.

# **Evaluation**

In this chapter, we evaluate the algorithms explained in the above sections on Geolife dataset. We also evaluate user datasets to extract the meaningful data and then the prediction results by the application of the algorithm.

## 6.1 User data analysis

The data used for this master thesis is Geolife dataset from Microsoft [1] [2] [3]. The dataset contains 182 users’ GPS trajectory data for the period of five years. The trajectory data contains the latitude, longitude, date, time, altitude information which is tracked every 1 to 5 seconds. Majority of the dataset was created in China with few exceptions of USA and Europe. This included several different types of users, few with a lot of trajectory data over years and few only for few weeks. For instance, user 17 has 1026179 trajectory points and user 72 has only 81.

A total of 73 users have also labelled their transportation mode while recording the GPS trajectories. The Figure 27 explains the distance travelled altogether using different transportation modes.

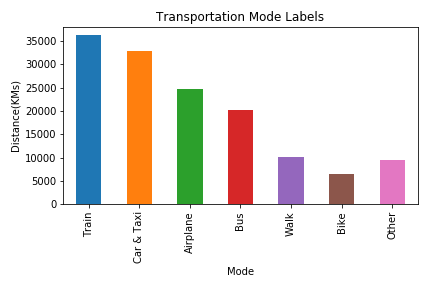


Figure 27 User transportation mode

The transportation labels summary helped us to understand that most users were travelling while recording their GPS trajectories. The average trajectory speed, which is calculated per trajectory file for each user in the dataset, is depicted in the box plot in Figure 28. The median travel speed was found to be 5.73 km/h. This indicate that most users tracked the GPS trajectories outdoor.

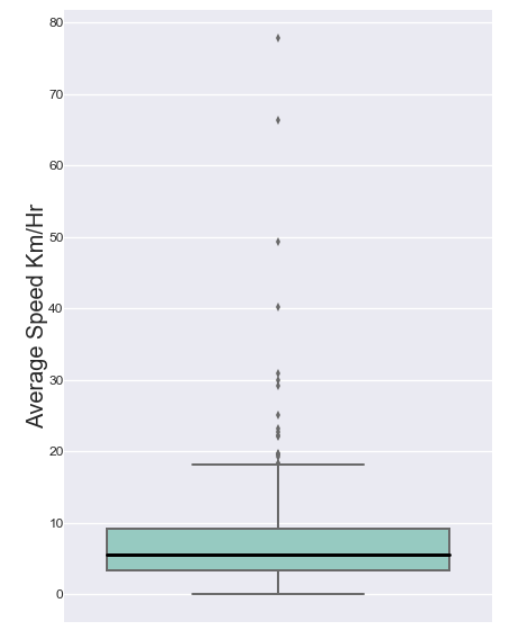


Figure 28 Geolife dataset trajectory average speed

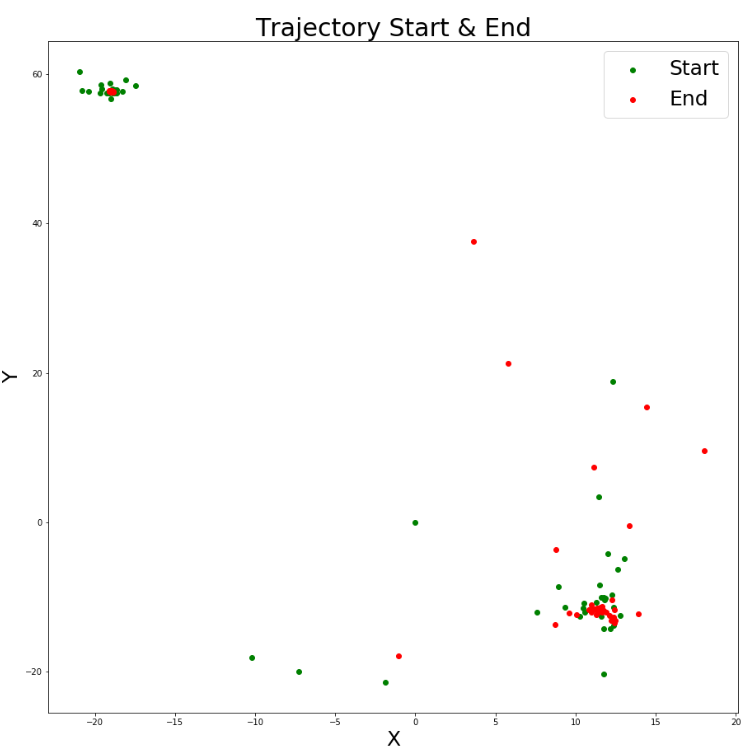


Figure 29 User 1 start and end points from each trajectory

The analysis of each user also shows some important patterns. For instance, the first and last GPS coordinates of each trajectory for user 1 is depicted in the Figure 29. The green dots indicating the start of each trajectory and the red dots indicting the end of each trajectory. This also shows a pattern indicating that the trajectory data was mostly recorded outdoor. For example, the trajectory was starting at home and ending at work and repeated in cycles. This analysis motivated us adding the starting and the ending point of each trajectory as stay-points. The approach is explained further in the chapters before.

## 6.2 Evaluation and Results

In this section, we detail evaluate each individual component of the algorithm.

### 6.2.1 Stay-points Evaluation

The algorithm is applied to Geolife dataset. The user files are read in an online manner to simulate the GPS location points received on a mobile device. The stay-points distance and time threshold values are important to be appropriate. If the distance threshold is very large, a lot of insignificant locations will be part of stay-point cluster and if the time threshold is very small, the short duration stays will also be regarded as stay-points. The Figure 30 shows the stay-point count on a user trajectory trace for varying the distance and time threshold values in range 50m-350m and 10min-30min respectively. It is evident from the figure that the number of stay-points are very high with small time threshold and reduces as the time threshold increases. The time threshold value at both the extremes results either in very less stay-points or many stay-points, hence we select a value from the middle. Performing several tests analysis, we chose 200 meters and 20 minutes for distance and time threshold respectively. This threshold values can vary with the type of data.

The stay-points found for user 1 for November 2008 as shown in the Figure 31. The trajectory is shown with the green line and the red arrows indicate the stay-points. These stay-points represent the locations with semantic meaning behind it like “home”, “work”, “restaurant”. This clearly depicts that a lot of noise in the trajectory data is removed at this step and only the significant stays are extracted.

The stay-points algorithm considers the start and end trajectory points and smoothens the data between two stay-points, based on the distance between and time elapsed. The result of this approach is compared against the stay-point extraction within the trajectory points. The results are shown in Figure 32. The data used is from Geolife dataset for user 1 for the first few days in November. The x-axis shows the time-slots and the y-axis shows the days. The top of the Figure 32 shows the black marked rectangles which represents the raw trajectory data. The figure indicates that a lot of time-slots are missing data, especially the data after hour 23 till hour 7 in the morning. The middle portion of the figure indicates the stay-points found only within the trajectory points. The time-slotted data here does not represent any regular pattern and misses the important locations. The bottom of the figure depicts the stay-points found using our approach. The pattern “home-work-home” is now very evident. Hence, this approach does not miss the most important states like “home” and “work”.

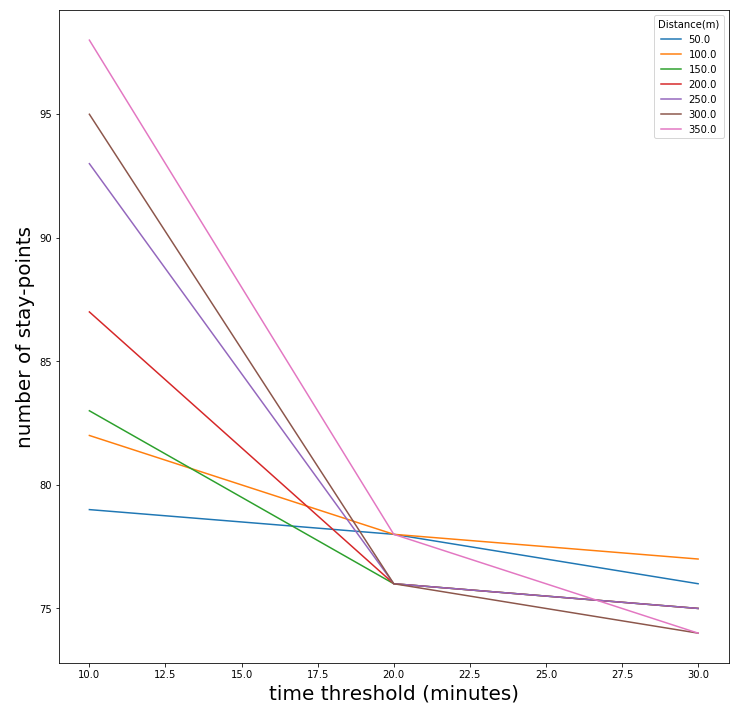


Figure 30 Stay-points count for different time and distance thresholds

### 6.2.2 Time-slotted States

The hourly weights calculate the time-slotted data which is used as the base for markov chain model.

The Geolife dataset date and time are represented in GMT, hence to have the correct visualization, the date and time must be adjusted to the local time in the trajectory data. The Figure 33 below depicts the hourly weighted state data for user 1 for November 2008. The x-axis represents each hour of the day from 0 to 24 and the y-axis represent the days. Each rectangle represents a state where the width of the rectangle represents the weight of the state in the corresponding hour. For example, state 1 is the first state between 9 am and 10 am on day 0. The distribution of the states over the hours and days makes some hints about the semantic meaning behind the locations. For instance, state 3 is most likely user’s home location as on most days’ user is at this location from 9 pm till next day 7 am and state 6 is most likely user’s work location as on most days’ user is at this location from 8 am till 8 or 9 pm. There are many other locations like 13, 22, 2, 16 and so on which could represent the supermarket, shopping mall, fitness club and so on.

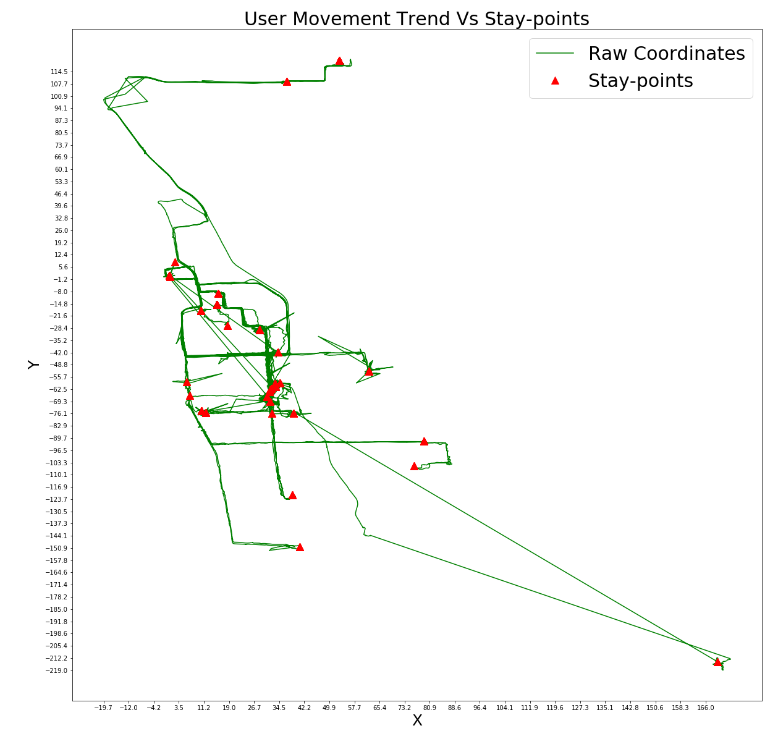
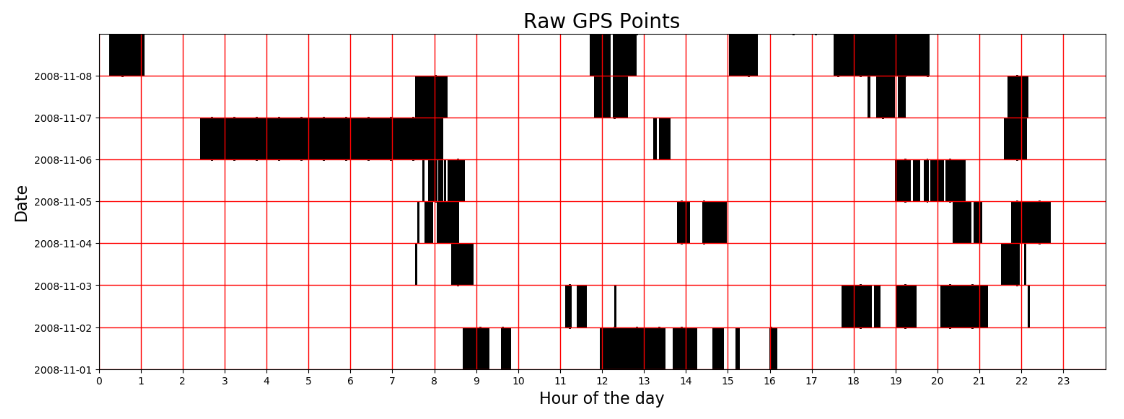
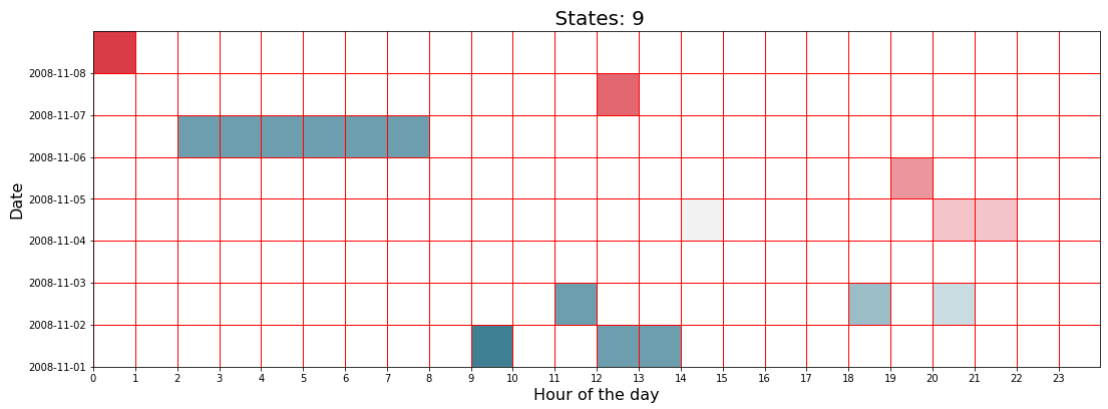


Figure 31 User 1 raw trajectory data vs stay-points extracted





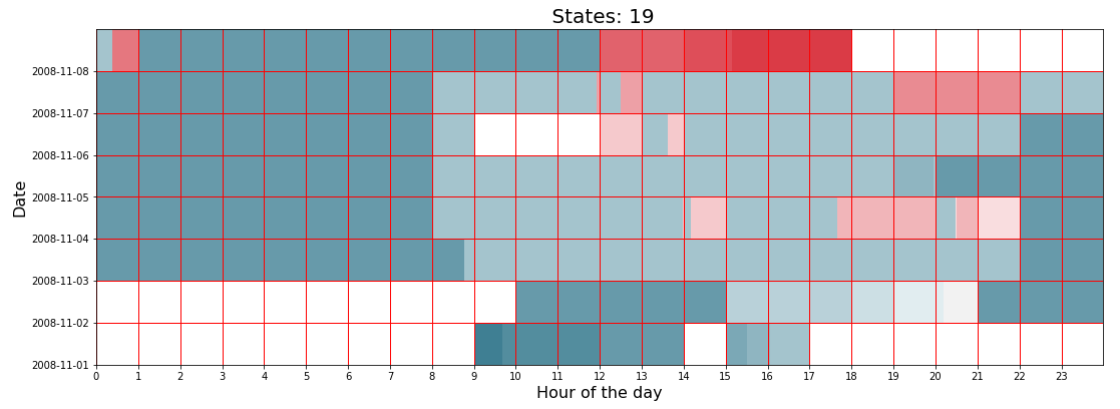


Figure 32 User 1 eight days’ time-slotted; 1. Top: Raw Trajectory Data, 2. Middle: Stay-points within Trajectories only, 3. Bottom: Stay-points with our algorithm

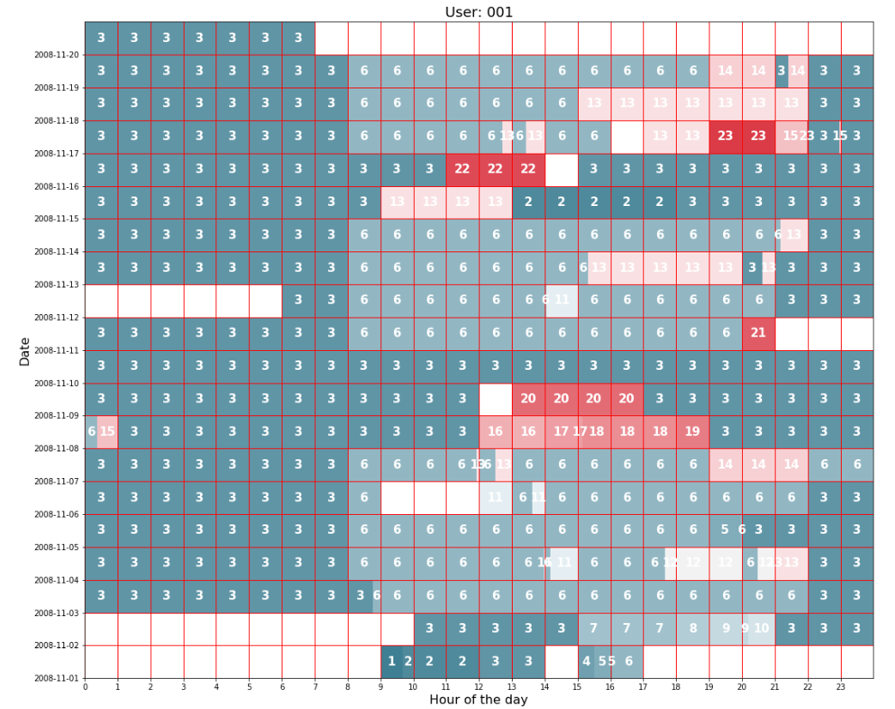


Figure 33 User 1 time-slotted data for the November 2008

### 6.2.3 Path Prediction Evaluation

The algorithm takes the current hour, location and minimum threshold as input and uses the markov model to predict the several paths representing the several locations in consecutive time-slots with their confidence percentages. The Figure 34 shows the paths predicted for user 1 for a known state 3 at hour 7 with minimum confidence 0.1. The x-axis denotes the hour of the day and each path starts in a new line. There exist two paths. The circle with the number denotes the states predicted and the color of each circle is to indicate the confidence of the prediction. The first predicted state at hour 8 in path 1 is 6. Similarly, the first predicted states at hour 8 in path 2 is state 3. The states with darker color indicate a high confidence and the states with lighter color indicates the lower confidence. The confidence reduces as we go further away from the starting hour i.e. 7. The path continues until the confidence drops below the threshold confidence or the day has ended. There is an additional state shown after the drop of confidence below threshold. This additional state is shown to see the drop in the confidence of the next state after the minimum confidence.

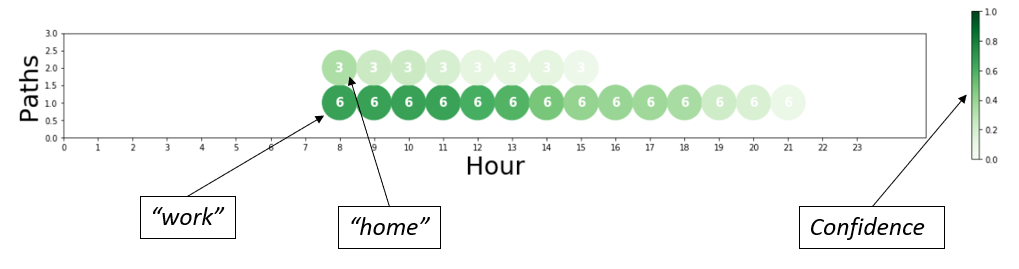


Figure 34 Path prediction for user 1

The path prediction is done to aware the user about the different paths which are predictable based on his/her location data collected. The known location was only state 3 at hour 7, which is used to predict the paths for several hours.

#### 6.2.3.1 Survey for Memory-loss Factor

The predicted paths follow a very strict markov chain model. The model is counting each transition. The application users would not expect such strict behavior and tend to forget many previous visits. As proposed in the earlier chapters, we can apply this behavior to our model using a memory-loss factor. The memory-loss factor can be calculated using the feedback from the users directly.

Consider the example of user 1 data for November 2011 for first few days as depicted in Figure 35. The x-axis denotes the hour of the day and the y-axis denotes the days. During the evening the user has been visiting many new places. The transition from hour 18 and 19, as marked in the Figure 35, is interesting to focus. The state 6 seems to be user’s “work” location. The user has visited location 5 and 14 on day 5 and 7 respectively at hour 19, after state 6 “work” location at hour 18. On other days like 3 and 6, he stays at “work” location 6 at hour 19. The markov chain will remember these transitions. This in turn reduces the probability of user to be at state 6 “work” location at hour 19.

As a result of this, the path predicted if the known location is at state 6 “work” at hour 14 is shown in Figure 36. Because of several possibilities at hour 19, the confidence drops instantly.

But humans do not consider each detail while thinking about the transitions. It is easy to forget the places we have visited 2 weeks before after work. To improve the path prediction algorithm, it is interesting to forget some minor transitions like humans do. This can be done using the inputs from the user, hence a small survey is conducted.

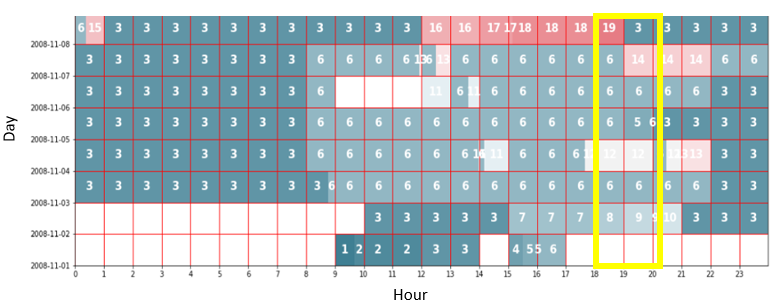


Figure 35 User 1 November 2011 hourly distributed data

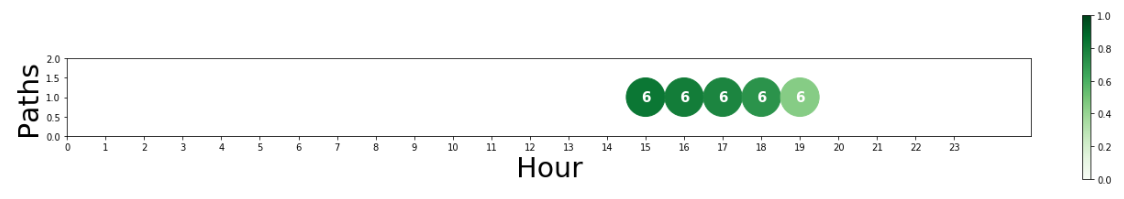


Figure 36 Path prediction user 1 with minimum confidence of 50%

*Survey:*

The survey contained 10 participants with the median age of 26.5 years. The participants are shown the time-slotted data as in Figure 33 for 1 minute of time. Before showing the time-slotted data, the participants are explained that the x-axis represents the hour of the day and the y-axis represents several days the data is collected for. Each rectangle represents a location with the id inside the rectangle. They are also told to focus on user’s working hours and his movements trends. After this, all participants are asked a same question.

*“How long will the user stay at location “Work” if he was observed at location “Work” at x hour.”*

The variable x is replaced with an actual hour value based on the data. The question is to understand how the participants forget the minor transitions and expects the user to be at “work” for longer hours. One of the Geolife user data is shown to the participants which is the Figure 33. The x in the question was 8am. The average answer from participants reported that the user will stay till 8:00pm at state 6 “work” location. The path prediction from our algorithm resulted in Figure 37. The prediction is very strict which says user will stay at state 6 “work” only till 14 hours. The minimum confidence considered here is 80% and the prediction of state 6 at hour 14 is already below the minimum confidence. The way participants have observed the movements and the how markov chain has calculated the transitions, are not same. Most participants ignored the infrequent and minor transitions like from state 6 to state 11 from hour 13 to 14 on day 3, and so on. But markov model recorded these transitions and this in turn reduced the probability of staying at state 6 at hour 14. Hence, markov model also need to forget these minor transitions. This is called as applying memory-loss factor to the markov model.

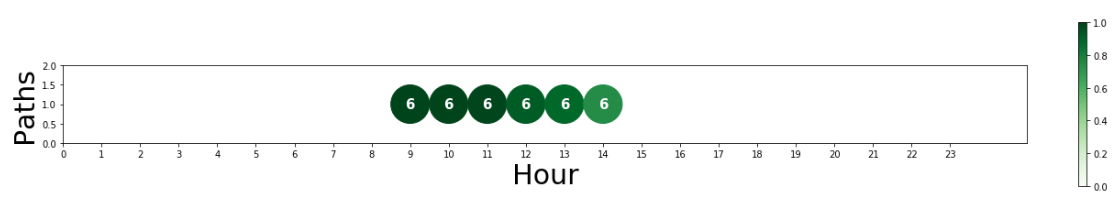


Figure 37 Path prediction result for user 1 without changes in the algorithm

The memory-loss factor is calculated for each participant. The factor is simply calculated by changing the markov model to produce the same output as the participant has suggested. For example, if the participant suggested that the user will stay at state 6 till 9pm, the same output should be produced with our path prediction. This can be implemented if the probabilities are reduced by a factor and then normalized again.

After application of a memory-loss factor on user 1, we get the following result as shown in the Figure 38. The memory-loss factor of 0.22 changes the suggestion to stay at state 6 till 21 hours.

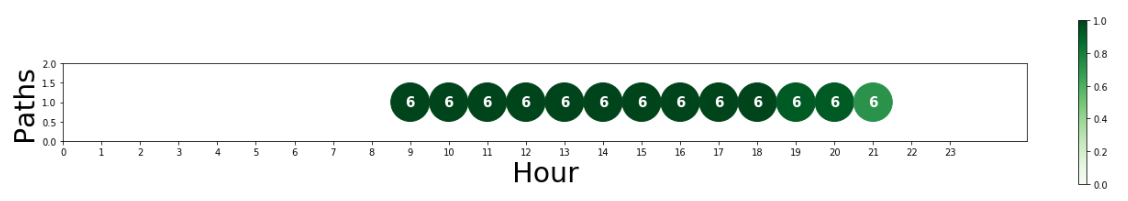


Figure 38 Path prediction result for user 1 with memory-loss factor of 0.22

Similarly, the memory-loss factor is calculated for each participant, based on their answer, to calibrate the algorithm. The average memory-loss factor is reported to be 0.17.

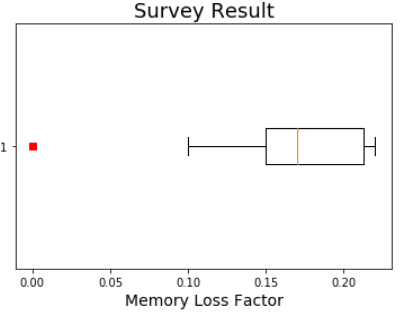


Figure 39 Memory-loss factor result from survey

The survey was only conducted with 10 participants. The results can change with more participants and with different age groups. More research must be done in this direction.

### 6.2.4 Prediction Performance

The location prediction algorithm implemented on Geolife dataset for 182 users. The data is divided for training and test purposes. The user month, with most trajectory points, is selected as the training month and the closest month available to the training month is selected as the test month. This is done to build the model for the month where highest data is available. The closest available month, next to training month, is selected as the test month as this is most likely to have the same pattern as the training month.

#### 6.2.4.1 Metric

The prediction algorithm is evaluated for three parameters, accuracy correctness Ac, accuracy precision Ap and cosine similarity C.

The accuracy correctness Ac is the defined as the ratio of total correct predictions Pc to the total prediction Pt:

The prediction is counted to be made only from the known locations. The known locations are the ones which are present in the markov model. A prediction is counted as correct, if the predicted vector with highest weight is also the one visited (present in ground truth vector) in the next hour slot, else an incorrect prediction. Consider an example as shown in Figure 40. On the left of the Figure 40, from the training model, it is suggested to transition from state st1 to st2 and st3 from hour 5 to hour 6. The actual transition is made from st1 to st2. In this case the prediction is considered as correct as the prediction suggested to be at state st2 with highest weight. In the middle of the Figure 40, the prediction is made from st1 to st3 from hour 5 to hour 6 and the actual transition is made from st1 to st2 from hour 5 to hour 6. Hence, this is considered as the incorrect prediction. On the right, the prediction is neither correct or incorrect as there is no stay-point found in the hour 6. The predictions are rated only for the time-slots where an actual transition has been occurred.

The accuracy precision Ap is defined as:

Where *P* is the prediction vector, *GT* is the ground truth vector. From the Figure 40, the left most scenario yields the prediction vector states as P = [st2, st3] and the ground truth vector contains states as GT = [st2, 0].

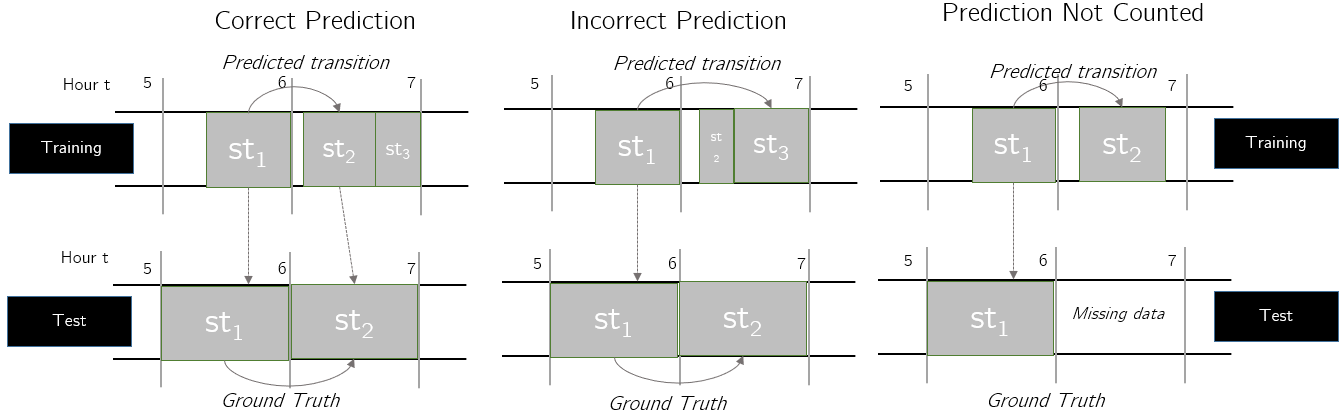


Figure 40 Correct(left), Incorrect(middle) or none(right) prediction scenario

The cosine similarity C is calculated as:

The numerator is the dot product of the prediction vector *P* and ground truth vector *GT* and the denominator is the magnitude product of the two vectors. The cosine similarity C = 1 represents the vectors are exactly same, which means the prediction is 100% matching as the actual transitions. A cosine similarity C = 0 represents orthogonality, which means that the prediction is completely wrong.

The probability of transition from the current state to all the other state in the next time-slot is fetched from the markov chain model as *p = {p­1, p2, …, pn}* where *p­i* indicates the probability of state *i*. The true stay in the next time-slot is also stored as a vector *gt = {gt1, gt2, …, gtn}* where *gti* represents the true or actual stay of state *i*. These two vectors are compared to determine the cosine similarity. Hence, in contrast with the accuracy correctness Ac and accuracy preciseness Ap, the cosine similarity C considers the probability of prediction from markov chain and the true duration of the stay. It means, even if the predicted transition and ground truth are same, if the prediction suggests very less duration stay in the next hour but the ground truth is for longer duration in the next hour, the cosine similarity will be smaller. Let us consider the example shown in the Figure 40 on the left. The transition from st1 to st2 is predicted and the ground truth is also the same. The prediction is correct but to what extend? The prediction suggests a very short duration stay at state st2 in the next hour, whereas the actual stay at hour 6 at state st2 is longer. This will result in smaller cosine similarity value. Hence, the cosine similarity C will be close to 1 only if the duration of stay in ground truth matches with the prediction, otherwise it will be close to 0. Therefore, the cosine similarity close to 1 suggests precise predictions and the cosine similarity close to 0 suggests bad predictions.

#### 6.2.4.2 Results

The first evaluation done is to find out the accuracy correctness and accuracy precision. For a total 182 users, the box plots are shown in Figure 41. The mean accuracy correctness Ac for 182 users is found to be 92.7%. The accuracy precision Ap for 182 users is found to be 63.7%.

Another evaluation is done using cosine similarity. The cosine similarity mean for 182 users is found to be 0.725 as shown in the Figure 41.

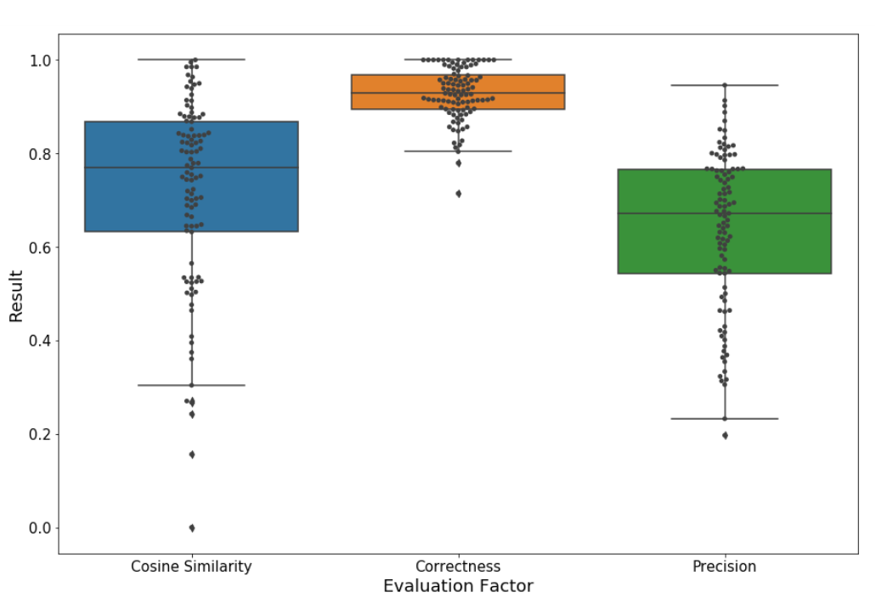


Figure 41 Evaluation for Geolife dataset

### 6.2.5 Android Application Performance

# **Conclusion and Future Work**

In this chapter we summarize the thesis contribution and work, the overall evaluation results and possible aspects of future work are discussed.

## 7.1 Summary

The Location Based Applications LBAs and social media applications, keep collecting the user location-based data. Based on this collected location details, the third parties can easily prediction user’s whereabout. To help user understand the consequences of sharing location details with the third parties, a location prediction model is built. The markov chain model is suggested for location prediction on mobile devices.

The steps for refining the raw trajectories are suggested and implemented. For instance, stay-points are extracted from the raw trajectory data and the noise like travelling GPS coordinates and short stay locations are removed. Based on the stay-points, several states are formed, which combines the geographically close-by stay-points into one state. The states represent the location like “home”, “work”, “restaurants”, etc. The state weights are calculated at each hour of the day. Hence, the state coordinate locations add the spatial feature and distributing the states over time, add the temporal feature. Based on the states, a markov model is created for each hour of the day. This markov model is used for future location predictions and path prediction. The algorithm is built in an online manner to simulate the actual use case scenario.

The prediction model is used for path predictions. For instance, if user’s know location is “home” at hour 6, which paths will he/she take after this known location. The several paths are predicted with a confidence percentage. The prediction probabilities from markov chain are used to determine this confidence. This helps to understand the predictability based on a known location and a time-slot. The algorithm is implemented, and a visualization approach is recommended for the android application users. An improvement of the path prediction algorithm based on the user inputs are suggested. This is done by a survey with 10 participants to understand their expectations of predictability. A factor of memory-loss is calculated for each participant to suggest the application of user expected behavior in path prediction.

The approach is tested and evaluated on real-life data Geolife dataset from Microsoft [1] [2] [3]. The dataset contains 182 user GPS trajectory data over the period of 5 years. The approach is implemented on python and android. The algorithm resulted in 92.7% mean accuracy correctness, 63.7% mean accuracy preciseness and average cosine similarity of 0.725 for 182 different users. The evaluation results suggest that the approach can predict basic movements with an above average accuracy percentage.

## 7.2 Discussion

In this thesis, we presented an algorithm for location prediction using markov chain model. The prediction approach is less complex and applicable for most users. The markov model can be built during the run-time on a mobile device. The GPS coordinate points can be received in the background and time-slotted data can be created. It could be possible to improve the accuracy and cosine similarity by adding more features to the markov model e.g., separate weekend/weekdays/public holidays markov chain, track peak travelling hours for each user, user calendar entries and call/SMS logs from mobile phone to estimate next movements. It is obvious that the addition of more features increases the complexity of the algorithm.

The thesis work had several challenges as it dealt with real-life data. It was difficult to generate a stay-point clustering algorithm as many users have missing data for several days in between. The basic clustering algorithm did not result into a pattern. It was an important investigation that the most GPS trajectories are tracking only outdoor movements. As a result of this investigation, the start and end points of trajectories are considered as stay-points. Since, the users were real, user evaluation was an important section.

The survey for improving the path prediction had only 10 participants. The survey must be conducted on more participants to be able to learn the user behavior on large scale.

## 7.3 Future Work

The model is implemented only with one temporal feature which is the hour information for each location data. The prediction model can have better prediction accuracy with more spatio-temporal. For instance, building separate markov models for weekdays and weekends may increase the accuracy for the users who have very distinctive movement patterns on weekdays and weekends. Another aspect which can be tested is, weighting the transition probabilities based on geographical distance between the states. The transition from one state to another state is more likely to happen if the states are geographically close to each other. Hence, weight the transition probabilities more if they are close to each other and weight less if they are very far apart. The algorithm can also be tested in future to accept more data like calendar entries or call/SMS log. As suggested by the paper [7], it can reveal important information about user’s next movement. Even though, adding new features will increase the complexity of the algorithm, it could be interesting to see the effect on the prediction model cosine similarity and path prediction.

The idea is prototyped on the android to showcase the basic approach of the application. The next location prediction model on android device uses the smoothened data due to the limited duration of time of this thesis. This can be extended in future to accept the raw GPS points as input on the android device and create and update the markov model regularly. Currently, the hourly state weights or time-slotted data is used as the input for the android application. The markov chain and path prediction algorithms are implemented on android application independently. The python code can be used as an example to extend the android implementation. The finished android application can be published on Google Playstore to collect user feedback.

Another important direction of research could be to estimate a generalized memory-loss factor as introduces in the section 4.4.6.2 . Once the application is completed and published on Google Playstore, user feedback on path predictability can be received. This feedback can be applied to the markov model to generalize the path predictions and forget short, infrequent transitions. The user memory-loss factor can be then collected for several users. I will be interesting to see if the there exists a general memory-loss factor for most users or it vary a lot. This can also help to improve the user interface and calculate the memory-loss factor for each user.

Bibliography

|  |  |
| --- | --- |
| [1] | L. Z. X. X. W.-Y. M. Yu Zheng, "Mining Interesting Locations and Travel Sequences from GPS Trajectories," *Proceedings of the 18th International,* 2009. |
| [2] | Q. L. Y. C. X. X. W.-Y. M. Yu Zheng, "Understanding Mobility Based on GPS Data," in *In Proceedings of ACM conference on Ubiquitous Computing ACM Press: 312-321*, Seoul, Korea, 2008. |
| [3] | X. X. W.-Y. M. Yu Zheng, " GeoLife: A Collaborative Social Networking Service among User, location and trajectory," in *Invited paper, in IEEE Data Engineering Bulletin. 33, 2, pp. 32-40*, 2010. |
| [4] | *Cunningham V. New York State Department Of Labor (Challenging Warrantless Gps Tracking Of State Employee's Personal Car),* 2013. |
| [5] | A. H. W. W. T. Andreas Gorlachl, "Survey on Location Privacy in Pervasive Computing," *Robinson P., Vogt H., Wagealla W. (eds) Privacy, Security and Trust within the Context of Pervasive Computing. The International Series in Engineering and Computer Science, vol 780. Springer.* |
| [6] | W. W. B. S. G. B. Jong Hee Kang, "Extracting Places from Traces of Locations," *Proceedings of the 2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots,* October 2004. |
| [7] | S. S. N. L. C. M. Anastasios Noulas, "Mining User Mobility Features for Next Place Prediction in Location-based Services," *Computer Laboratory, University of Cambridge,* no. IEEE, 10-13 December 2012. |
| [8] | C. P. S. K. Joao Bartolo Gomes, "Where will you go? Mobile Data Mining for Next Place Prediction," *Institute for Infocomm Research (I2R), A\*STAR, Singapore,* August 2013. |
| [9] | N. M. P. J. H. A. K. S. B. A. T. Mitra Baratchi, "A Hierarchical Hidden Semi-Markov Model for Modeling Mobility Data," *UBICOMP '14,* pp. 13 - 17, SEPTEMBER 2014. |
| [10] | W. K. S. S. Paul Baumann, "The Influence of Temporal and Spatial Features on the Performance of Next-place Prediction Algorithms," *UbiComp’13,* September 8–12 2013. |
| [11] | P. R. Center, "Americans’ complicated feelings about social media in an era of privacy concerns," 2018. |

# Declaration

I hereby declare that I have done the thesis work independently with the help of my supervisors. Only the sources listed in the bibliography are used.

Shashank Sharma