ON THE INFLUENCE OF SPATIO-TEMPORAL DATA ANALYSIS ON CLUSTERING AND RECOMMENDATION

by

MADHURI DEBNATH

DISSERTATION

Submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy at

The University of Texas at Arlington

December, 2017

Arlington, Texas

SUPERVISING COMMITTEE:

Ramez Elmasri, Supervising Professor,

Leonidas Fegaras,

Gautam Das,

David Levine

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my Ph.D. supervisor Professor Ramez Elmasri. It has been a great honor to be his Ph.D. student. I would like to thank him for his continuous support of my Ph.D. study and research. His immense knowledge, motivation and guidance helped me in all the time of my research. The enthusiasm he has for his research was the biggest motivation for me, even during the tough times in the Ph.D. pursuit. I could not have imagined having a better supervisor and mentor for my Ph.D. study. I would like to extend my gratitude to the rest of my Ph.D. committee members Dr. Leonidas Fegaras, Dr. Gautam Das and Mr. David Levine. I appreciate their interests in my research and helpful comments throughout my Ph.D. journey.

I would like to thank the members of MAST (Mining and Analysis of Spatio-Temporal Data) lab. Especially, I recall countless discussions with Dr. Praveen Kumar Tripathi, which were really encouraging for me surviving through the tough time of the Ph.D. track.

My time at UT Arlington was made more enjoyable due to many friends. My heartiest gratitude to them for always being there in my happy time and hard time. Special thanks to Mohammad Fakrul Islam for being the company in our most memorable mountain hiking trips.

My heartiest gratitude to my father Ranjit Debnath and mother Beauty Debnath for their unconditional love and encouragement. My sister Shimul Debnath and my brother Rajib Debnath were always a great source of inspiration for me.

Finally, my loving and caring husband, Dr. Ashis Kumer Biswas. You are my best friend since we were freshmen at the University of Dhaka. I cannot thank you enough for being the constant source of support and encouragement throughout the half of my entire life. I strongly believe I am the luckiest person in the whole world to have you in my life.

November 13, 2017

To my parents.

RELATED PUBLICATIONS

- **Madhuri Debnath**, Praveen Kumar Tripathi, and Ramez Elmasri, "Preference-Aware Successive POI Recommendation with Temporal and Spatial Influence", *The 8th International Social Informatics Conference (SocInfo) 2016*, Washington, USA, 2016
- Praveen Kumar Tripathi, **Madhuri Debnath**, and Ramez Elmasri, "A Direction Based Framework for Trajectory Data Analysis", *The 9th ACM International Conference on Pervasive Technologies Related to Assistive Environments (PETRA), 2016*, Corfu Island, Greece, June 29 - July 01, 2016.
- **Madhuri Debnath**, Praveen Kumar Tripathi, and Ramez Elmasri, "Preference-Aware POI Recommendation with Temporal and Spatial Influence", *Proceedings of the 29th International Florida Artificial Intelligence Research Society Conference (FLAIR) (pp 548-553)*, Florida, USA, 2016.
- Praveen Kumar Tripathi, **Madhuri Debnath**, and Ramez Elmasri, "Directional Analysis of Trajectories Based on Trajectory Smoothing," *5th International Workshop on Mobile Entity Localization and Tracking in GPS-less Environment, MELT 2015*, Seattle, WA, USA, 2015.
- **Madhuri Debnath**, Praveen Kumar Tripathi, and Ramez Elmasri, "K-DBSCAN: Identifying Spatial Clusters with Differing Density Levels", *International Workshop on Data Mining with Industrial Applications, DMIA 2015.*, Paraguay, September 14-16, 2015
- Praveen Kumar Tripathi, **Madhuri Debnath**, and Ramez Elmasri, "Extracting Dense Regions from Hurricane Trajectory Data", *First International ACM*

Workshop on Managing and Mining Enriched Geo-spatial Data, GeoRich 2014. 5:1-5:6., UTAH, USA, 2014.

- **Madhuri Debnath**, Praveen Kumar Tripathi, and Ramez Elmasri, "A Novel Approach to Trajectory Analysis using String Matching and Clustering", *ICDM Workshops 2013: 986-993*, Dallas, Texas, USA, 2013.

ABSTRACT

ON THE INFLUENCE OF SPATIO-TEMPORAL DATA ANALYSIS ON CLUSTERING AND RECOMMENDATION

Madhuri Debnath

The University of Texas at Arlington, 2017

Supervising Professor: Ramez Elmasri

In this dissertation, we propose efficient frameworks to analyze spatio-temporal data. In the first part of the dissertation, we use a clustering based method to mine useful information from trajectory data. Existing trajectory clustering algorithms have focused on geometric properties and spatial features of trajectories. In contrast to existing algorithms, we propose a new framework to cluster sub-trajectories based on a combination of spatial and non-spatial features.

In the second part of dissertation, we propose a unified framework to build recommendation systems by analyzing human movement data. We propose recommendation frameworks to recommend POI locations and travel routes that use a combination of spatial, temporal and content features. POI recommendation method aims to provide users with a list of recommendation of POI locations within a geo-spatial range that should match their temporal activities and categorical preferences. In travel route recommendation method, we propose to recommend time-aware and preference-aware travel routes consisting of a sequence POI locations with correspond-

ing time information. This method helps users to plan the entire trip under a specific time constraint. The recommended travel routes tell users where to visit and when to visit. For all the problems, we provide extensive experiments with real world spatio-temporal data available in public domains. The performance evaluation validates the utility and the effectiveness of the proposed methods over baseline approaches.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Spatio Temporal Data Analysis

Spatio-temporal data deals with spatial and temporal aspects of data. Trajectory data is an example of spatio-temporal data. A trajectory is a sequence of time-stamped GPS locations (latitude, longitude). Due to the advances in location-acquisition technologies, a massive amount of spatial trajectory data has been generated. This data represents the mobility of a diversity of moving objects, such as humans, animals or vehicles. Such trajectories offer us a lot of useful information to understand moving objects and locations. Analysing trajectory data has many other applications such as managing the traffic pattern of vehicles, monitoring and predicting weather conditions, examining wild animal behaviour and movement, as well as analysing the spread of disease. A number of attempts have been made in this domain to analyse these kinds of data sets.

Another example of spatio-temporal data is user check-in data collected by mobile GPS. "Check-in" is a process by which a user posts his/her presence or arrival to a physical location, Users' movement data with location and time information provide us better knowledge about their activities and interests. The availability of such information opens up an array of new research problems and various real world applications. POI recommendation systems and Travel Route recommendation systems are the examples of such real world applications.

In this dissertation, we present novel frameworks and algorithms to analyze spatio-temporal data. Here we provide a high level overview of our contributions:

1. **A Novel Approach to Trajectory Analysis Using String Matching and Clustering**: Our first contribution is to propose a new framework to cluster sub-trajectories based on a combination of their spatial and non-spatial features. This algorithm combines techniques from grid-based approaches, spatial geometry and string processing. Given a set of sub-trajectories, this framework clusters the sub-trajectories to group them based on the combination of their spatial and non-spatial features. In this work, we first convert each trajectory into a representative sequence that captures the trajectory direction and location. We identify common sub-trajectories from all the sequences using a modified string matching algorithm. Then, we extract non-spatial features from the common sub-trajectories. Finally, we present a density-based clustering algorithm to cluster the sub-trajectories. This work is presented in Chapter 2.
2. **Identifying Spatial Clusters With Differing Density Levels**: Spatial clustering is a very important tool in the analysis of spatial data. In this contribution, we propose a novel density based spatial clustering algorithm called *K-DBSCAN* with the main focus of identifying clusters of points with similar spatial density. This contrasts with many other approaches, whose main focus is spatial contiguity. The strength of *K-DBSCAN* lies in finding arbitrary shaped clusters in variable density regions. Moreover, it can also discover clusters with overlapping spatial regions, but differing density levels. The goal is to differentiate the most dense regions from lower density regions, with spatial contiguity as the secondary goal. *K-DBSCAN* works in two phases: first, it divides all data objects into different density levels to identify the different natural densities present in the dataset; then it extracts the clusters using a modified version of DBSCAN. This work is presented in Chapter 3.

3. **Preference-Aware POI Recommendation with Temporal and Spatial Influence**: POI recommendation is a very important application in Location Based Social Network (LBSN) that provides users personalized location recommendation. It helps users to explore new locations and filters uninteresting places that do not match with their interests. Multiple factors influence users to choose a POI, such as user's categorical preferences, temporal activities and location preferences as well as the popularity of a POI. In this contribution, we use user movement data and define a unified framework that takes all these factors into consideration. This method aims to provide users with a list of recommendation of POIs within a geospatial range that should match with their temporal activities and categorical preferences. This work is presented in Chapter 4.
4. **Preference-Aware Successive POI Recommendation with Spatial and Temporal Influence**: Successive POI recommendation refers to the problem of recommending users the very next location based on his current location and the current time. Traditional POI recommendation cannot suggest where a user may go the next day or next hour based on their current location or status. In this contribution, we consider the task of personalized successive POI recommendation, recommending to a user the very next location where he might be interested to go next based on his current location. Multiple factors influence users to choose a POI, such as user's categorical preferences, temporal activities and location preferences, the popularity of a POI as well as sequential patterns of a user. In this work, we define a unified framework that takes all these factors into consideration to build a better successive POI recommendation model. We use a real-world user check-in dataset collected from Foursquare. This work is presented in Chapter 5.

5. **Preference Aware Travel Route Recommendation with Temporal Influence**: Travel route recommendation is one of the recent and most important applications in the LBSN services. Travel route recommendation provides users a sequence of POIs (Point of Interests) as a route to visit. In this contribution, we propose to recommend time-aware and preference-aware travel routes consisting of a sequence of POI locations with corresponding timestamps. It helps users not only to explore interesting locations in a new city but also it will help to plan the trip accordingly with those locations with timestamp information in a specific time constraint. First, we find the interesting POI locations that consider the following factors: User's categorical preferences, temporal activities and popularity of location. Then, this work proposes an efficient solution to generate travel routes with those locations. The travel routes will tell users where to visit and when to visit. This work is presented in Chapter 6.

1.2 Dissertation Organization

In Chapter 2, we presented our method of clustering sub-trajectories [1]. In this work, we aim to cluster trajectories considering both spatial features and non-spatial features. The main motivation behind this is to find groups of spatial dense regions of trajectories with similar non-spatial attribute behaviour. Our proposed framework has 4 major phases: 1) dimension reduction, 2) trajectory segmentation, 3) non-spatial feature extraction and 4) clustering. In the first phase, each trajectory is mapped from high dimensional (usually 2 or 3) space to one-dimensional space. This mapping simplifies trajectory representation and their comparison in later stages of our framework. The second phase exploits the string matching concept to identify common sub-trajectories among all trajectories. In this work we use a modified version of Longest Common String Matching (LCS) algorithm. The third phase deals with

identifying significant non-spatial features from second phase. In the final phase, the sub-trajectories are clustered using *DBSCAN* algorithm based on the combination of spatial and non-spatial features. We performed experiments using our proposed method with real world hurricane trajectory data. Experimental results show that our framework correctly discovers groups of similar sub-trajectories with their similar non-spatial features.

In Chapter 3, we present a new density-based spatial clustering algorithm *K-DBSCAN*. The motivation of this algorithm is to analyse spatial data that can handle data with different density levels. Unlike the DBSCAN [2] algorithm, it does not depend on the global ϵ parameter to calculate neighbourhood, rather each data point dynamically generates its own parameter to define its neighbourhood. Hence, it has less sensitivity to the user specified parameter. Our proposed method works in two major steps: 1) K Level Density Partitioning and 2) Density Level Clustering. In the first phase, we calculate the density of each data point based on its distance from its nearest neighbouring data points. Then we partition all the data points into K groups based on their density value. In this phase, we introduce a modified version of DBSCAN algorithm that works on different density levels. Our proposed K-DBSCAN algorithm can be utilized in several applications. For example, it can be used to find spatial clusters with differing population density levels, even when these clusters are overlapping. We experimented the proposed method with both synthetic data and real world data. Experimental results demonstrates the effectiveness of our algorithm [3].

In Chapter 4, we present a preference-aware, location-aware and time-aware POI recommendation system that offers a particular user a set of POI locations incorporating time information and geo-spatial range. In this framework, we incorporate time dimension to model time-specific user preferences, so our recommendation

model aims to recommend POI locations that match the time-specific preferences of individual user. We further exploit user's spatial behaviour using location histories to generate spatial-aware location preferences. Our recommendation model uses the popularity factor of individual locations by calculating both time-specific popularity and regional popularity. We model personal preferences of users based on the category information of their location histories. We estimate the similarity between two users by computing similarity between their personal preferences rather than using user's location vector. There are 2 main reasons behind this. First, it handles the data sparsity problem of user-location matrix. Second, two users who do not visit the exact same venue may still share common interest if their preferences are the same. Then We evaluate our system with a real-world dataset collected from Foursquare. The extensive experimental results with evaluation show that our method combining multiple factors (temporal, spatial, popularity, preferences) provide users better and effective recommendations than other baseline approaches.

In Chapter 5, we present a preference-aware, location-aware and time-aware successive POI recommendation system. However traditional POI recommendation systems consider all check-ins as a whole and generate recommendations [4, 5, 6, 7, 8]. They do not consider the users' sequential movement information. Therefore, they cannot suggest where a user may go in the next few hours based on their current location or status. In this work, we consider the task of personalized successive POI recommendation. Successive POI recommendation refers to the problem of recommending users the very next location based on his current location and current time. This task recommends those locations that a user may not visit frequently or before, but he/she may like to visit at successive timestamps [9]. For example, successive POI recommendation can suggest a user location to have fun after dinner, or a location for outdoor activities in a nearby park after his work. We develope a successive POI rec-

ommendation model **PLTSRS** (**P**reference-Aware, **L**ocation-Aware and **T**ime-Aware **S**uccessive POI **R**ecommendation **S**ystem), which jointly considers user's personalized sequential movement information, temporal categorical preferences, location preferences and popularity of POIs.

In Chapter 6, we present an efficient framework to generate preference-aware and time-aware travel route recommendation system. Our goal is to recommend top-K travel routes with the combination of interesting locations that will match with users' time specific interests. Each location of a route will be associated with the corresponding approximated time information. By this recommendation, the user will know not only where to go, but also when to go. First, incorporate time dimension to model time-specific user preferences. We estimate the similarity between two users based on the user-preference vector rather than the user-location vector. We build a model to estimate the uncertain transition time between two locations. We propose a novel framework *"PTTR-Reco"* (**P**reference-Aware, **T**ime-Aware **T**ravel **R**oute **Reco**mmendation) to recommend top-K travel routes to users.

Finally, Chapter 7 summarizes the contributions made in this dissertation. It also outlines several short-term and long-term research goals on the foundations developed in the dissertation.

CHAPTER 2

A Novel Approach to Trajectory Analysis Using String Matching and Clustering

2.1 Introduction

Because of vast improvements in GPS and current sensor technologies, large scale trajectory data are available. The data provided by the technologies are raw data. Hence it becomes significant to discover important and meaningful information by analysing them. Examples of trajectory data includes vehicle position data, hurricane track data, animal or pedestrian movement tracking data, radio frequency identification (RFID), among many other examples.

Trajectories are represented as a sequence of spatio-temporal points. Analysing trajectory data has many other applications such as managing the traffic pattern of vehicles, monitoring and predicting weather conditions, examining wild animal behaviour and movement, as well as analysing the spread of disease. A number of attempts have been made in this domain to analyse these kind of data sets. Some of these analyses can be found in [10],[11],[12],[13],[14].

Existing trajectory clustering algorithms have focused on spatial proximity and spatial features (latitude and longitude) of trajectories. Similarity of non-spatial attributes of trajectories has not been considered. Example of non-spatial attributes of trajectories include wind speed, length, area coverage, frequency of stops between movements and so on. These attributes can be significantly different from each other. For example, two hurricane trajectories or their sub-trajectories may have common

tracking patterns based on spatial location and direction, but they may have different wind speed and wind pressure or time-span of two hurricane can be different.

Example

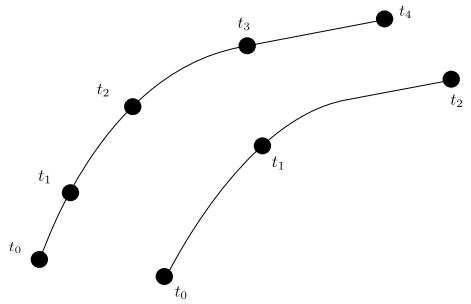

Figure 2.1: An example of two trajectories

Consider the two trajectories in Fig. 1. The trajectory data that we use it from the hurricane dataset [15]. In this dataset, time points in a trajectory are 6 hours apart. Trajectory that starts at t_0 and ends at t_4 has total duration of 24 hours. Another trajectory starting at t_0 and ending at t_2 has total duration of 12 hours. It is obvious that average wind speed of the second trajectory is higher than the first one, because they have the same approximate length but a different number of points.

In this chapter, we aim to cluster trajectories considering both spatial features and non-spatial features. The main motivation behind this is to find groups of spatial dense regions of trajectories with similar non-spatial attribute behaviour. Here we give one example to illustrate that discovering common sub-trajectories considering both type of feature is useful.

1. Storm trajectory analysis has an important application in forecasting hurricane landfall information [11]. Storms with high wind speed and intensity is more significant than those of low speed and intensity value. So, analysing these

attributes will be useful to predict the storm locations with high wind speed and intensity.

The rest of the chapter is organized as follows. In Section 2.2, we review some related work. We describe our proposed algorithm in Section 2.3. Section 2.4 presents experimental results of our algorithm. Finally, Section 2.5 concludes the chapter.

2.2 Related Work

In this section, we briefly describes some works that are most relevant to this one. In [16], the authors proposed a model based clustering algorithm, where a set of trajectories are represented using a regression mixture model. EM algorithm is used to determine cluster membership. They consider the whole trajectory as a basic unit of clustering.

In [11], the authors propose a partition and group framework to cluster trajectories. In the partitioning phase, trajectories are divided into some line segments. This division has been done using the notion of characteristic points which reflect the most significant points in the trajectory. In the grouping phase, these line segments are clustered using DBSCAN algorithm [17]. The obvious drawback of this algorithm is that only the line segments are being clustered.

In [10], the authors proposed a modular approach to cluster sub-trajectories. This approach is based on combination of techniques from computational geometry, string processing and data mining.

In [18], the authors have proposed a technique for mining a spatio temporal pattern called the flocking behaviour in an online fashion. The flocking patterns refer to the set of the trajectories that remain close to each other for some reasonable time interval. The authors consider both the time information along with the spatial attributes in mining the flocking behaviour.

In [14], the authors propose a non-parametric approaches to cluster spatial trajectories. This article deals with trajectory clustering and the post analysis of the cluster results. In this approach, they propose a clustering algorithm that uses a randomized hill climbing technique to find some local maxima of the density function. The clusters finally result when the trajectories belonging to the same local maxima are grouped together. The post processing of the obtained spatial clusters is performed to get more domain specific knowledge.

Another important analysis of sptio-temporal tracking data has been proposed in [19]. The authors have given a framework for mining the sequential patterns from the spatio temporal data. This task gets very important when studying the evo- lution of some phenomena in spatial and temporal domain. They proposed a sequence index that is important in identifying the significant spatio-temporal sequential patterns from the spurious ones. A novel algorithm called Slicing-STS-Miner has been proposed to use the given sequence index in order to efficiently obtain the spatio-temporal sequential patterns.

In [20], the authors extended their previous work on trajectory clustering [11] and proposed a new algorithm of trajectory classification. They proposed two types of clustering: 1) region-based clustering and 2) trajectory based clustering. The motivation is to arrive at the discriminative features, which are very vital in generating the classifier model for the classification task. The first level of the clustering, which is the region level, identifies the higher level, region based features of the trajectories, ignoring the movement based features at this stage. The second level of the clustering identifies the lower level movement based features. These two clustering collaboratively identify the high-quality features for the classification task.

2.3 PROPOSED FRAMEWORK

Our proposed framework has four phases which are, 1) dimension reduction, 2) trajectory segmentation, 3) non-spatial feature extraction and 4) clustering.

In the first phase, we map each trajectory from high dimensional (usually 2 or 3) space to one-dimensional space. This mapping simplifies trajectory representation and their comparison in later stages of our framework.

The second phase deals with the identification of common sub-trajectories among the dataset. This phase exploits the string matching concept to identify common sub-trajectories among all trajectories. In this work we use a modified version of Longest Common String Matching (LCS) algorithm [21].

In the third phase, we extract non-spatial features from the sub-trajectories obtained from the second phase. Some examples of non-spatial features are wind speed, trajectory length etc.

In the fourth phase, we cluster the sub-trajectories based on the combination of their spatial and non-spatial features. We use DBSCAN algorthim for clustering.

2.3.1 Representation of Trajectory

Each trajectory is represented as a sequence of n spatial locations with time information viz., $(t_0, l_0), (t_1, l_1), (t_2, l_2), (t_3, l_3), ...(t_n, l_n)$. Here, n is the trajectory length. Each location l_i is a 2-dimensional point. The length of one trajectory can be different from another one. At the same time, the shape and movement of each trajectory is different. We can consider one trajectory as $n \times 3$ dimensional matrix, where n = number of points in the trajectory.

$$\text{Trajectory } \mathrm{T}_1 = \begin{bmatrix} t_0 & x_0 & y_0 \\ t_1 & x_1 & y_1 \\ t_2 & x_2 & y_2 \\ .. & .. & .. \\ t_n & x_n & y_n \end{bmatrix}$$

In this phase of our algorithm, we focus on mapping a trajectory from two dimensional space to one dimensional space. There are several techniques to accomplish this [22]. The most prominent ones include the Z-order, Gray Codes and Hilbert Curve. In spatial database management systems, these mapping techniques are used to store and index location information on disk, as disk storage is logical one dimensional device [22]. In this chapter, we aim to simplify the trajectory data to facilitate recognition of moving pattern and to compare it with other trajectory data. For this task we propose our novel algorithm.

We divide the whole problem space domain into $M \times N$ grid cells, where each grid cell has equal length and width. We consider each grid cell as a spatial region. Each region is identified with a unique identification number viz., *id*. The identification number is assigned in row major order. For example, in the first row, $1..N$ column is identified with numbers from 1 to N. In the second row, all columns are identified with $N + 1$ to $N + N$ and so on (see **Algorithm 1**).

2.3.1.1 Example

Let us consider the following example of one trajectory.

$$\text{Trajectory } \mathrm{T}_1 = \begin{bmatrix} 0 & 1.5 & 1.5 \\ 6 & 1.8 & 2.5 \\ 12 & 3.5 & 2.8 \\ 18 & 5.5 & 4.8 \\ 24 & 6.2 & 5.5 \\ 30 & 4.8 & 5.8 \end{bmatrix}$$

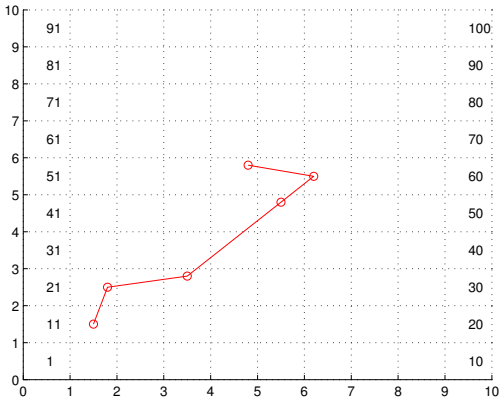

Figure 2.2: Trajectory T_1

Let us, divide the problem space domain into 10×10 square grid cells, with each grid cell representing 1 spatial region. Now we assign each grid cell a unique identification number from 1 to 100, as we have a total of 100 grid cells. In first row, columns are identified from 1 to 10, in the second row, they are identified from 11 to 20. In order to map the trajectory T_1 in this grid, it's first point which is viz., $(0, (1.5, 1.5))$ is mapped to grid location number 12. Fig. 2 shows the first and last grid cell number in each row.

We assume that the interval between two successive points is the same (as is given in the dataset) [15]. For example, trajectory represented as $T_1 = (12, 22, 24, 46, 57, 55)$

(see Fig. 2). Hence, a trajectory of length n is a sequence of $T[n] = (\ l_1, l_2, l_3.....l_n\ )$, where l_i is the grid location that approximately represent a trajectory point.

The advantage of this approach is its simplicity and it takes linear time to map each point to an one-dimension space. We can also do the reverse mapping from grid cell representation to approximate 2-D position of a point. For example, if grid $id = l_o$ and we have N columns, then x coordinate can be retrieved as $l_o \mod N$ and y coordinate can be retrieved as $\lceil l_0/N \rceil$ (see **Algorithm 2**). Note that, because of the approximate mapping of the 2-D points to the grid *ids* the reverse mapping does not guarantee the exact 2-D points, only approximate location.

For example, grid location $l_0 = 12$ is mapped to 2-D point $(2, 2)$ and $l_1 = 22$ is converted to 2-D point $(2, 3)$. To measure distance between two grid locations, we measure euclidean distance between 2-D approximate points of two grid *ids* (see **Algorithm 3**).

Algorithm 1 Transformation of 2-D point to 1-D value

Input: (i) x, (ii) y, (iii) N

1. $p = ceil(x + floor(y) * N)$
2. **return** p

Algorithm 2 Transformation of 1-D value to approximate 2-D point

Input: (i) l_0, (ii) N

1. $x = l_0 \mod N$
2. $y = \lceil l_0/N \rceil$
3. **return** (x, y)

Algorithm 3 Grid distance between two grid location

Input: (i) l_i, (ii) l_j, (iii) N

1. $< x_i, y_i >=$ Two-D-Transform (l_i,N)
2. $< x_j, y_j >=$ Two-D-Transform (l_j,N)
3. $d = \sqrt{(x_i^2 - x_j^2) + (y_i^2 - y_j^2)}$
4. **return** d

2.3.2 Segmentation Algorithm

In the previous section, we focused on the approximate and simplified representation of a trajectory. In this section, we define an approach to segment a trajectory into sub-trajectories. We compare two trajectories and find the approximate common segments between them. This idea originates from Longest Common Substring (LCS) matching algorithm [21].

2.3.2.1 Longest Common Approximate Trajectory Segments (LCATS)

Given two trajectory sequence S of length m (s_1,s_2,s_3,.....s_m) and T of length n (t_1,t_2,t_3,.....,t_n). Let X (x_1,x_2,...,x_p) be a sub-sequence of S and Y (y_1,y_2,..y_p) be a sub-sequence of T. X and Y are called approximate common trajectory segments of S and T if for each points of X and Y, grid-distance(x_i,y_i) $\leq \epsilon$. In LCATS Problem, we wish to find the longest common trajectory segments between two trajectories. We have used dynamic programming approach to find the LCATS between two trajectories.

Example

Consider the following example (see Fig. 3). Here, S, T, V are three different trajectories. (a, b, c, d) and (h, i, j, k) are LCATS of trajectory S and T. (m, n, o) and (p, q, r) are LCATS of trajectory S and V.

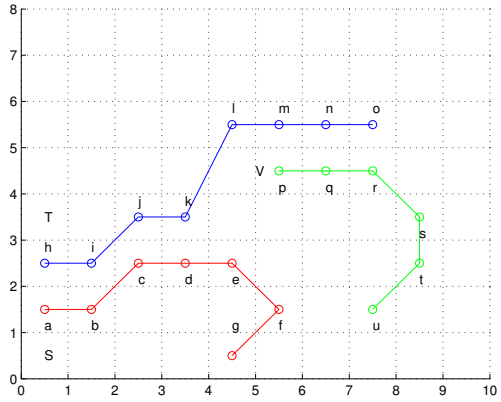

Figure 2.3: Longest Common Approximate Sub-trajectories

2.3.2.2 Identifying sub-trajectories

We consider all trajectories to find the common patterns with other trajectories. To reduce the cost of comparing a trajectory sequence with another trajectory sequence, we come up with the idea to use the notion of Minimum Bound Rectangle (MBR). MBR is a popular technique for indexing spatial objects. It is also used to indicate approximate spatial positions of spatial objects.

2.3.2.3 Example

Consider the example in Fig. 4. Minimum bounding rectangle of trajectory 1 and 2 intersects with each other. So, there is a possibility that they have some common segments between them. But MBR of trajectory 3 is not intersecting with any one of them. So, we do not consider trajectory 3 to compare with trajectory 1 and 2.

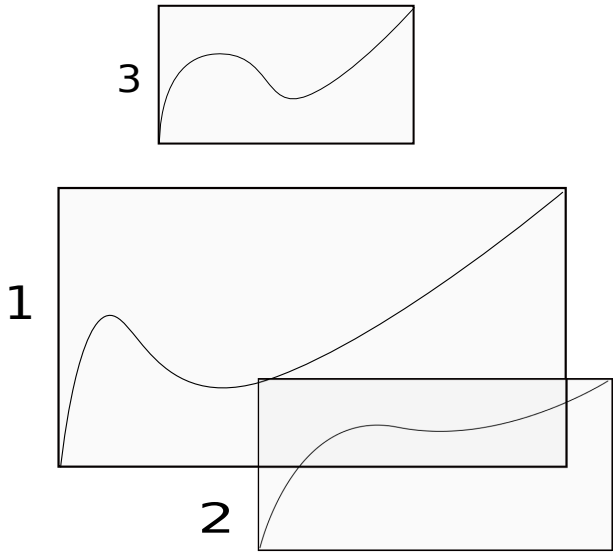

Figure 2.4: MBR of three trajectories

2.3.3 Non-spatial feature extraction

After the segmentation phase described in the previous section, we obtain a set of sub-trajectories. In this section, we identify some non-spatial features associated with them. Non-spatial features are application dependent. In different application domain, different non-spatial features provide significant information. For example, In storm trajectory data some significant features are:

1. Average wind speed during the storm
2. Time-length of each storm
3. Wind pressure
4. storm intensity
5. Area coverage

2.3.4 Density-based Clustering

In this section, we present the sub-trajectory clustering algorithm to group them based on the combination of their spatial and non-spatial features.

Algorithm 4 Identifying Sub-trajectories

Input: (i) T : a set of trajectories $(T_1, T_2, ...T_n)$

(ii) S : a set of sub-trajectories $(s_1, s_2, ...s_m)$

1. $S = \emptyset$
2. **for** $i = 1 \rightarrow n$ **do**
3. **for** $j = i + 1 \rightarrow n$ **do**
4. **if** MBR of T_i intersects MBR of T_j **then**
5. U =LCATS(T_i,T_j)
6. **if** is not Null **then**
7. $S = \{U\}$
8. **end if**
9. **end if**
10. **end for**
11. **end for**

The aim of density-based clustering algorithm is to identify dense regions that are separated by low-density region. Two most important characteristics is that, they can identify clusters with arbitrary shape and they can find outliers [17].

In this phase we consider two issues,

1. We define two different distance functions. First distance function $dist_s$ aims to find the spatial proximity between two sub-trajectories. The second distance function $dist_{ns}$ is to find their non-spatial attribute similarity.
2. We cluster them considering both distance functions.

2.3.4.1 Distance function $dist_s$ (spatial distance)

We define a new distance measure to find the spatial proximity of two sub-trajectories. This distance function is based on the aggregated nearest neighbor distance between the points of the trajectories.

We are given two sub-trajectories S_a and S_b. Here $|S_a|$ denotes the length of S_a and $|S_b|$ denotes the length of S_b. If $|S_a| \geq |S_b|$, the spatial distance between two sub-trajectories S_a and S_b is defined as

$$dist_s(S_a, S_b) = \sum_{i=1}^{|S_a|} \min_{j=\{1,2,\cdots,|S_j|\}} \Big\{ d(S_{ai}, S_{bj}) \Big\} \tag{2.1}$$

That is, for each point in S_a, we find the minimum distance to a point in S_b, and then sum these distances.

Example

Consider the two sub-trajectories S_a and S_b in Figure 5. Here $|S_a| \geq |S_b|$. Hence,

$dist_s(S_a, S_b) = d(s_{a1}, s_{b1}) + d(s_{a2}, s_{b1}) + d(s_{a3}, s_{b1}) + d(s_{a4}, s_{b2}) + d(s_{a5}, s_{b3}) + d(s_{a6}, s_{b3})$

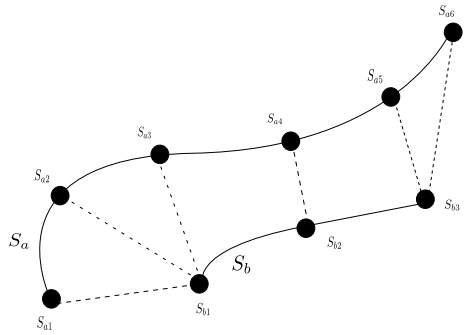

Figure 2.5: Spatial distance between two sub-trajectories.

2.3.4.2 Distance function $dist_{ns}$ (non-spatial distance)

In Section (C), we describe some non-spatial features. In this chapter, we use euclidean distance method to measure non-spatial attribute similarity among sub-trajectories.

Now we summarize the important notation required for density-based clustering algorithm. This approach is inspired by ST-DBSCAN algorithm [23]. Let D denote the set of all sub-trajectories in the database. This algorithm is based on three threshold : ϵ_s, ϵ_{ns}, $MinS$. The notations we used in the algorithm is:

ϵ_s-Neighborhood: The ϵ_s-Neighborhood of a sub-trajectory $N_{\epsilon_s}(S_i)$ is defined by $\{S_j \in D | dist_s(S_i, S_j) \leq \epsilon_s \}$.

ϵ_{ns}-Neighborhood: The ϵ_{ns}-Neighborhood of a sub-trajectory $N_{\epsilon_{ns}}(S_i)$ is defined by $\{S_j \in D | dist_{ns}(S_i, S_j) \leq \epsilon_{ns} \}$.

Neighborhood: The Neighborhood of a sub-trajectory $N(S_i)$ is defined by $\{S_j \in N_{\epsilon_s}(S_i) \cap N_{\epsilon_{ns}}(S_i) \}$.

Core object: A sub-trajectory S_i is considered as a core object if $|N(S_i)| \geq MinS$.

Border sub-trajectory: A sub-trajectory is considered as a border object if it is not a core object but density reachable from at least one core object.

Directly-density-reachable: A sub-trajectory S_i is directly-density-reachable from another sub-trajectory S_j with respect to ϵ_s, ϵ_{ns} and $MinS$ if 1) $S_i \in N(S_j)$ and 2) $|N(S_j)| \geq MinS$.

Density-reachable: A sub-trajectory S_i is density-reachable from another sub-trajectory S_j with respect to ϵ_s, ϵ_{ns} and $MinS$ if there are a chain of objects S_j, S_{j-1},S_{j-2},$\cdots\cdots$ $S_{i+1}, S_i \in D$ such that S_k is directly density reachable from L_{k+1} with respect to ϵ_s, ϵ_{ns} and $MinS$.

Density-connected: A sub-trajectory S_i is density-connected to a sub-trajectory S_j with respect to ϵ_s, ϵ_{ns} and $MinS$ if there is sub-trajectory $S_k \in D$ such that both S_i and S_j are density reachable from S_k with respect to ϵ_s, ϵ_{ns} and $MinS$.

Density-based cluster: A cluster C of sub-trajectories is a non-empty subset of D satisfying the following condition:

- $\forall S_i$,S_j: if $S_i \in C$ and S_j is density-reachable from S_i, then $S_j \in C$.
- $\forall S_i, S_j \in C$: S_i is density-connected to S_j with respect to ϵ_s, ϵ_{ns} and $MinS$.

2.3.4.3 Clustering Algorithm

In Algorithm 5, we present the clustering algorithm. Given a set D of sub-trajectories, this algorithm generates a set of clusters based on three threshold values: ϵ_s, ϵ_{ns} and $MinS$. ϵ_s determines object's spatial neighborhood area and ϵ_{ns} determines it's non-spatial neighborhood area. Based on these two threshold, this algorithm determines the neighbor sub-trajectories for each sub-trajectory.

2.4 Experiments and Result Evaluation

In this section, we evaluate the effectiveness of our clustering algorithm. We use a real trajectory data set viz., hurricane tracking data [15]. The data comprises Atlantic hurricanes from 1950 to 2000 (50 years). It contains 496 trajectories and 15,998 points. Each track in the data set consists of a sequence of hurricane data sampled at 6-hours intervals. The sample for each instance in a particular trajectory track has latitude, longitude, wind speed and wind pressure as its attributes. We did the experiments using MATLAB because of its better visualization.

In our experiments we used latitude and longitude as spatial attributes, and wind speed as non spatial attribute within a particular hurricane trajectory. The

Algorithm 5 Density-based Clustering

Input: (i) D: a set of trajectory segments

(ii)ϵ_s: Maximum distance to find spatial neighbor

(iii)ϵ_{ns}: Maximum distance to find non-spatial neighbour

(iv)$MinS$: Minimum number of sub-trajectories necessary to form a cluster

(v) Output: C : a set of clusters

1. $cId = 1$
2. **for** $i = 1 \rightarrow n$ **do**
3. **if** S_i is not visited and not clustered yet **then**
4. Mark S_i as visited
5. X = GET-NEIGHBOUR(S_i)
6. **if** $|X| < MinS$ **then**
7. S_i is marked as noise
8. **else**
9. assign clusterId to $\forall S \in X$
10. Insert all X into the Queue Q
11. **while** Q is not Empty **do**
12. pop the current object S_j
13. $Y = getNeighbour(S_j)$
14. **if** $|Y| \geq MinS$ **then**
15. **for** $\forall s \in Y$ **do**
16. **if** S_j is not noise **then**
17. **if** S_j is not in Cluster **then**
18. assign S_j in CId
19. **end if**
20. **end if**
21. **end for**
22. **end if**
23. **end while**
24. Increment CId by 1
25. **end if**
26. **end if**
27. **end for**

Algorithm 6 GET-NEIGHBOUR

Input: (i) D: a set of trajectory segments

(ii)ϵ_s: Maximum distance to find spatial neighbor

(iii)ϵ_{ns}: Maximum distance to find non-spatial neighbour

(iv)$MinS$: Minimum number of sub-trajectories necessary to form a cluster

(v) S : a set of sub-trajectories

1. X = Spatial Neighbours with respect to ϵ_s and $MinS$
2. Y = Non-spatial Neighbours with respect to ϵ_{ns} and $MinS$
3. $N = X \cap Y$
4. **return** N

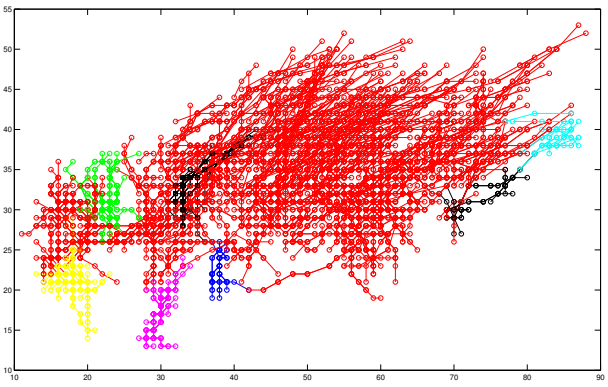

Figure 2.6: Clustering results considering spatial distance $\epsilon_s = 0.002$, $MinS = 7$

first stage of our framework, which is identifying sub-trajectories, resulted in 4044 sub-trajectories.

For these sub-trajectories, the average wind speed for each sub-trajectory is used as its non-spatial attribute. This is mainly motivated by the fact that wind speed is a key characteristic of the storm.

Figs 2.6, 2.7 and 2.8 show the clustering results obtained using only the spatial neighbourhood parameter (viz., ϵ_s), whereas the $MinS$ parameter is 7 throughout the experiments. The parameter ϵ_s determines the size of the spatial neighborhood

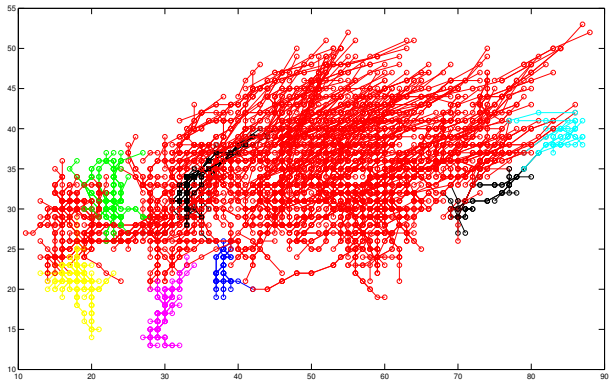

Figure 2.7: Clustering results considering only spatial distance with $\epsilon_s = 0.00195$, $MinS = 7$

to be evaluated for the density of a particular point. From Figs 2.6 and 2.7 it can be seen that changing the value of ϵ_s from 0.002 to 0.00195 did not change the clustering result significantly. Whereas, it can be seen from the result in Fig 2.8 for the value of $\epsilon_s = 0.0018$, that the clustering result changed significantly from those in the Figs 2.6 and 2.7 respectively.

After the experiments with only the spatial neighbourhood parameter, we incorporated the non spatial neighbourhood parameter also. Figs 2.9, 2.10, 2.11 show the results of our clustering algorithm using different parameter values of ϵ_s, ϵ_{ns}. Since we incorporate non spatial attributes also in the clustering process, we used ϵ_{ns} parameter corresponding to the non spatial attribute (viz., wind speed) for the neighborhood criteria. These parameters are very vital for the performance of the DBSCAN algorithm. For that reason we provide the results for three different sets of these parameters. This parameter specifies the threshold for the dense regions, i.e., a data point will be considered core (dense) only if it has at least $MinS = 7$ data points in its neighborhood.

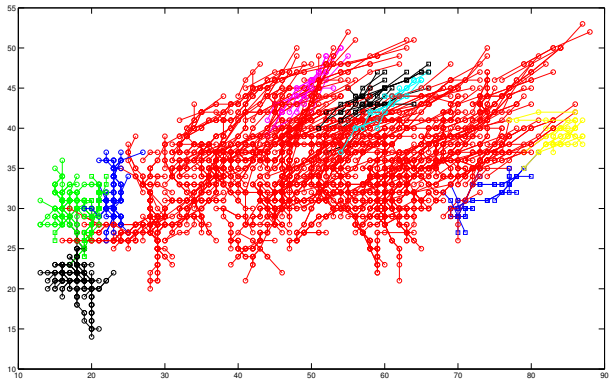

Figure 2.8: Clustering results considering only spatial distance with $\epsilon_s = 0.0018$, $MinS = 7$

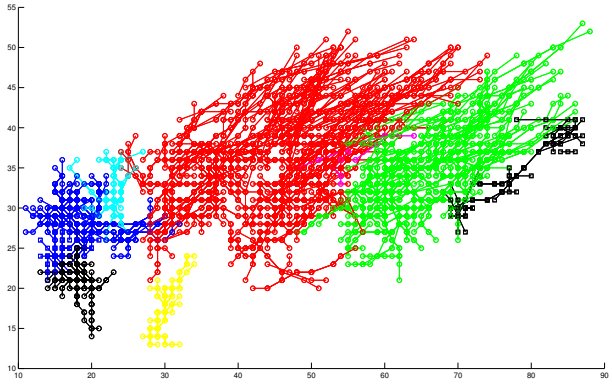

Figure 2.9: Clustering results with $\epsilon_s = 0.002$, $\epsilon_{ns} = 0.2$, $MinS = 7$

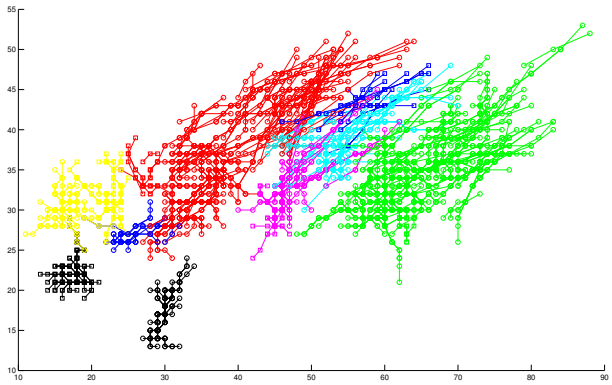

Figure 2.10: Clustering results with $\epsilon_s = 0.00195$, $\epsilon_{ns} = 0.15$, $MinS = 7$

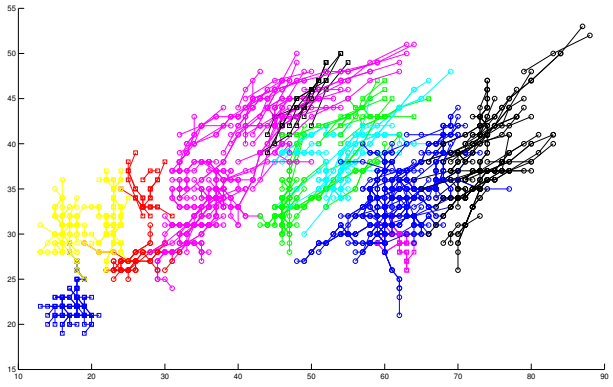

Figure 2.11: Clustering results with $\epsilon_s = 0.0018$, $\epsilon_{ns} = 0.15$, $MinS = 7$

Fig 2.9 uses $\epsilon_s = 0.002$, $\epsilon_{ns} = 0.2$ and $MinS = 7$. For these set of parameters the result shows four major clusters. The biggest cluster of the trajectories is shown in *red* and is in the central position with respect to the other clusters. This signifies that this particular region witnessed more hurricane activities. Other than this cluster we can see the next significant cluster shown in *green*. The third most significant cluster for this particular parameter set is the one in dark *blue*.

Fig 2.10 uses $\epsilon_s = 0.00195$, $\epsilon_{ns} = 0.15$ and $MinS = 7$. For these set of parameters the resulting clusters are very different from the ones given in Fig 2.9. Here the number of clusters is more than the former case and also there is no clear dominating clustering region. The clusters also seem to be overlapping to some extent.

Finally, Fig 2.11 uses $\epsilon_s = 0.0018$, $\epsilon_{ns} = 0.15$ and $MinS = 7$. Now the resulting clusters are looking better from the ones given in Fig 2.10. Here the number of clusters is reduced and the clusters are also more concrete with comparatively less extent of overlapping. Comparing this result with the one in Fig 2.9, it can be seen that the number of clusters is more in Fig 2.11. Also, the size of the clusters is comparatively smaller for the current set of parameters.

From the above set of experiments it can be seen that the results of the clustering are very susceptible to the choice of the parameters for the clustering algorithm. The changing behavior of the clustering result from Fig 2.6 to Fig 2.11 is natural as with the reduction in the values of ϵ_s, as well as ϵ_{ns}, we are restricting the size of the clusters to include only the more dense regions. In particular, the drastic improvement in the clustering result can be noticed between Fig 2.8 and Fig 2.9. This is because of the incorporation of the non spatial neighbourhood parameter which results in more compact clusters.

2.5 FUTURE WORK AND CONCLUSION

In this chapter, we have proposed a novel framework to cluster trajectories. In the first phase, we map each trajectory to one-dimensional space. This dimension reduction approach helps to simplify trajectories representation and reduce the cost of comparison with other trajectories. In the second phase, we present our own algorithm to identify all common sub-trajectories based on spatial proximity. In the third phase, we extract significant non-spatial features from them. Finally, we present a nearest neighbour based approach to measure spatial distance of sub-trajectories. Then we cluster all sub-trajectories based on the combination of spatial and non-spatial features.

We have performed our experiment on real life hurricane track data. We then compared the clustering that combines the spatial and non spatial attributes, with the clustering based on spatial attributes only.

Spatial datasets are quite large and it is an open research issue to handle big data efficiently. In our future work, we plan to design an algorithm in a distributed framework. We will analyse some other trajectory data sets like vehicle and animal

movement. In this approach we do not consider temporal information. We plan to include temporal information during clustering.

CHAPTER 3

K-DBSCAN: Identifying Spatial Clusters With Differing Density Levels

3.1 Introduction

Clustering is the process of grouping a set of objects into classes or clusters so that the similarity between the objects within the same cluster is maximized. For researchers who work with geographical and other types of spatial data, data mining has offered many useful and promising tools for data analysis. Spatial clustering is one of these tools [24].

Spatial Clustering has a wide range of applications. Some of them include crime hot-spot analysis, identification of similar land usage, earthquake analysis, agricultural environment analysis and merging of regions with similar weather patterns.

Spatial databases have some unique challenges. So, in order to choose a clustering algorithm that is suitable for a particular spatial application, some important issues need to be considered [25].

- Clustering algorithms should identify irregular shapes. Partitioning algorithms like K-means [26] or K-medoids [27] can discover clusters with spherical shapes and similar size. Density-based clustering algorithms like DBSCAN [2] are more suitable to find arbitrary shaped clusters.
- The algorithms should not be sensitive to the order of input. That means, clustering results should be independent of data order. For example, cluster quality and efficiency in K-means [26] depends on the choice of initial seeds, while cluster results in DBSCAN [2] do not depend on the data order.

- Algorithms should handle data with outliers. Density-based algorithms like DBSCAN [2] and OPTICS [28] can handle noise, while K-means [26] cannot.
- Algorithms should not be too sensitive to user specified parameter. For example, existing density-based algorithms like DBSCAN [2], DENCLUE [29] and OPTICS [28] need a careful choice of threshold for density, because they may produce very different results even for slightly different parameter settings.
- Lastly, clustering algorithms should handle spatial data with varying density. DBSCAN [2] fails to cluster this kind of data.

Motivated by these challenges, we propose a new density-based spatial clustering algorithm K-DBSCAN to analyse spatial data that can handle data with different density levels. Unlike the DBSCAN [2] algorithm, it does not depend on the global ϵ parameter to calculate neighbourhood, rather each data point dynamically generates its own parameter to define its neighbourhood. Hence, it has less sensitivity to user specified parameter.

Our proposed K-DBSCAN algorithm can be utilized in several applications. For example, it can be used to find spatial clusters with differing population density levels, even when these clusters are overlapping. Spatial analysis of regions based on population has important application in urban planning, healthcare and economic development. Population density levels of different regions are different.

The rest of the chapter is organized as follows. In Section 3.2, we review some related works. We describe our proposed algorithm in Section 3.3. Section 3.4 presents experimental results of our algorithm and compares the quality of the clustering result with three other well-known algorithms. In Section 3.5, we present a practical application of our algorithm with a real-world spatial dataset. Finally Section 3.6 concludes the chapter.

3.2 Related Work

Spatial Clustering algorithms can be partitioned into four general categories: Partitioning, hierarchical, density-based and grid-based.

Partitioning algorithms divide the entire dataset into a number of disjoint groups. Each disjoint group is a cluster. K-means [26], EM (Expectation Maximization) [30] and K-medoid [27] are three well-known partitioning based clustering algorithms. These use an iterative approach and try to group the data into K clusters, where K is a user specified parameter. The shortcoming of the algorithms is that they are not suitable for finding arbitrary shaped clusters. Further, they are dependent on the user specified parameter K.

Hierarchical clustering algorithms use a distance matrix as an input and generates a hierarchical set of clusters. This hierarchy is generally formed in two ways: bottom-up and top-down [27]. The top-down approach starts with all the objects in the same cluster. In each successive iteration a bigger cluster is split into smaller clusters based on some distance measure, until each object is in one cluster itself. The clustering level is chosen between the root (a single large cluster) and the leaf nodes (a cluster for each individual object). The bottom-up approach starts with each object as one cluster. It then successively merges the clusters until all the clusters are merged together to form a single big cluster. The weakness of the hierarchical algorithms is that they are computationally very expensive.

BIRCH [31] and CURE [32] are hierarchical clustering algorithms. In BIRCH, data objects are compressed into small sub-clusters, then the clustering algorithm is applied on these sub-clusters. In CURE, instead of using a single centroid, a fixed number of well scattered objects are selected to represent each cluster.

Density-based methods can filter out the outliers and can discover arbitrary shaped clusters. DBSCAN [2] is the first proposed density-based clustering algorithm.

This algorithm is based on two parameters: ϵ and $MinPts$. Density around each point depends on the number of neighbours within its ϵ distance. A data point is considered dense if the number of its neighbours is greater than $MinPts$. DBSCAN can find clusters of arbitrary shapes, but it cannot handle data containing clusters of varying densities. Further, the cluster quality in DBSCAN algorithm depends on the ability of the user to select a good set of parameters.

OPTICS [28] is another density based clustering algorithm, proposed to overcome the major weakness of DBSCAN algorithm. This algorithm can handle data with varying density. This algorithm does not produce clusters explicitly, rather computes an augmented cluster ordering such that spatially closest points become neighbours in that order.

The DENCLUE [29] algorithm was proposed to handle high dimensional data efficiently. In this algorithm density of a data object is determined based on the sum of influence functions of the data points around it. DENCLUE also requires a careful selection of clustering parameters which may significantly influence the quality of the clusters.

The Shared Nearest Neighbour (SNN) [33] clustering algorithm was proposed to find clusters of different densities in high dimensional data. A similarity measure is based on the number of shared neighbours between two objects instead of traditional Euclidean distance. This algorithm needs 3 parameters (k, ϵ, $MinPt$).

Grid-based clustering algorithm divides the data space into a finite number of grid cells forming a grid structure on which operations are performed to obtain the clusters. Some examples of grid based methods include STING [34], Wave-Cluster [35] and CLIQUE [36]. The STING [34] algorithm calculates statistical information in each grid cells. The Wave-Cluster [35] algorithm applies wavelet transformation to the feature base. Input parameters include the number of grid cells for each dimen-

sion. This algorithm is applicable for low dimensional data space. The CLIQUE [36] algorithm adopts a combination of grid-based and density-based approaches and this algorithm can detect clusters in high-dimensional space.

3.3 Proposed Algorithm

In this section, we focus on the basic steps of our proposed algorithm. We propose K-DBSCAN algorithm, which works in two phases.

- K Level Density Partitioning: In this phase, we calculate the density of each data point based on its distance from its nearest neighbouring data points. Then we partition all the data points into K groups based on their density value.
- Density Level Clustering: In this phase, we introduce a modified version of DBSCAN algorithm that works on different density levels.

3.3.1 K-DBSCAN Phase 1 - K level Density Partitioning

In real world spatial datasets, different data objects may be located in different density regions. So, it is very difficult or almost impossible to characterize the cluster structures by using only one global density parameter [37].

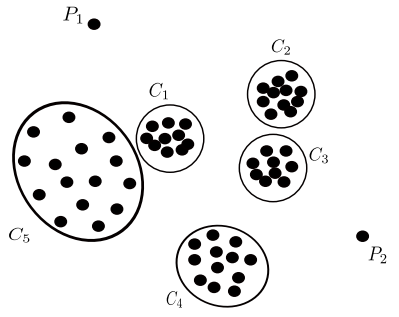

Figure 3.1: Points in different density regions

Consider the example from Figure 3.1. In this example, points in clusters C_1, C_2 and C_3 represents very dense neighbourhoods. Points in cluster C_4 represents a less dense region, while points in cluster C_5 represent a sparse neighbourhood. Point P_1 and P_2 should be considered as noise or outliers. As different data points are located in different density regions, it is impossible to obtain all the clusters simultaneously using one global density parameter. Because, if we consider the density estimation for points located in C_1, we have to choose a smaller ϵ value and we will find clusters C_1, C_2 and C_3. All other points will be considered as outliers. However, if we want to discover cluster C_5, we have to choose a larger value of ϵ, but this may result in a bigger cluster by including most of the data points as its neighbour (for example, C_1 may merge with C_5 forming a bigger cluster).

Consider another example in Figure 3.2. Here cluster C_1 is surrounded by another cluster C_2. Points in C_2 represent a less dense region, while points in C_1 are in a high density region. It is also difficult to identify both clusters using *DBSCAN* [2].

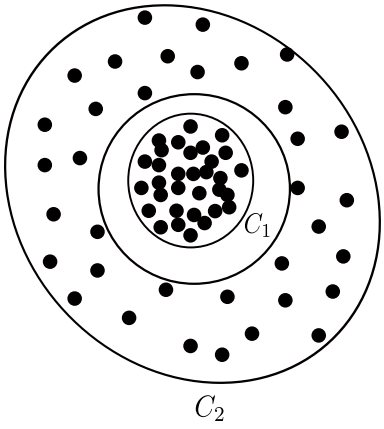

Figure 3.2: One cluster surrounding another cluster

To overcome these problems, we aim to partition all the data points based on their density values. The density value of a point depends on the distance of the point from its nearest neighbours.

There are different measures to find the density of a point. In *DBSCAN* [2], density of a point is defined as the total number of neighbours within a given radius(ϵ) of the point. But this does not work well in dataset with varying densities. A large number of points may be considered as noise due to lack of significant neighbourhood, in terms of global ϵ. SNN [33] clustering algorithm used the same definition for density measure, but they did it in terms of shared nearest neighbour similarity, which is relatively insensitive to variation of density. Still we have to estimate a global radius parameter, ϵ.

In this chapter, we have defined a new measure to find density. Here we introduce some important definitions required for this algorithm.

Definition 1: l-nearest neighbour distance

The l-nearest neighbour distance of a point P, denoted as $dist_l(P)$ is the distance between the point P and its l^{th} nearest neighbour. Where, $l \geq 1$. If $l\text{-}set(P)$ is the set of l closest neighbours of point P and d_i is the distance from point P to its i^{th} closest neighbour, then

$$dist_l(P) = \max\{d_i\} \tag{3.1}$$

Definition 2: l-density

The l-density of a point P, denoted as l-density(P) is defined as follows

$$l\text{-}density(P) = \frac{1}{l}\sum_{i=1}^{l} d_i \tag{3.2}$$

3.3.1.1 Partitioning

We divide the data points into $\boldsymbol{K}$ groups based on their density values. For each point P, we calculate l-density(P) using equation (2). Let $P = \{p_1, p_2, ...p_n\}$ be the set of n points. Let $D = \{d_1, d_2, ...d_n\}$ be the set of density values of the data points, where l-density(p_i) $= d_i$.

We use a V-optimal histogram based approach to partition the data points into $\boldsymbol{K}$ groups. In Database System, *Histogram* is a popular method to represent and summarize data. It also provides an estimation of the probability distribution of a continuous variable. The most basic histogram is *Equi-width* histogram, where each bucket represents the same range of values. *V-Optimal* histogram defines bucket boundaries in an optimal way, where the main goal is to minimize the errors over all point queries.

V-optimality is a partition rule which states that the bucket boundaries are to be placed as to minimize the cumulative weighted variance of the buckets. A v-optimal histogram is based on the concept of minimizing a quantity which is called the weighted variance in this context. The objective function is defined as

$$W = \sum_{k=1}^{K} n_k V_k \tag{3.3}$$

Here, K is the number of bins or buckets, n_k is the number of items contained in the k-th bin and V_k is the variance between the values associated with the items in the k-th bin.

Given the number of buckets K, our goal is to find the bucket boundaries for each bucket that will minimize the sum of weighted variances of the l-density values of points in buckets. Computing *V-Optimal* histogram is more expensive than the

basic histogram. We use a greedy based approach to find the optimal histogram with K buckets.

First, we sort the points based on their l-density values. Let $P' = \{p'_1, p'_2, ...p'_n\}$ be the set of n points, where l-density$(p'_i) \leq l$-density(p'_{i+1}). We maintain an additional array to keep track of the partitions. Let $L = \{\chi_1,\chi_2,...\chi_K\}$ be the set of K partitions. Here, χ_i represents the i-th partition. For each i, $\chi_i \subset P'$.

Algorithm 7 V-OPTIMAL-PARTITION

Inputs:

D: a set of data points ($P_1, P_2, P_3, ...P_n$)

K: number of buckets

Outputs:

L: set of partition ($\chi_1, \chi_2, ..., \chi_K$)

Input: (i) D: a set of data points ($P_1, P_2, P_3, ...P_n$)

(ii) K: number of buckets

(iii) Outputs: L: a set of partition ($\chi_1, \chi_2, ..., \chi_K$)

1. L = [D] /*Initially L has one partition*/
2. **for** i=1 to $(K-1)$ **do**
3. /* Partition the dataset D into $(i+1)$ partitions */
4. g = number of partitions in L
5. **for** j=1 to g **do**
6. χ_j = partitions that has maximum weighted variance
7. **end for**
8. $[\chi_{j1}][\chi_{j2}]$ = Split χ_j into two groups
9. $L = L$ - $[\chi_j]$
10. $L = L \cup [\chi_{j1}][\chi_{j2}]$
11. **end for**

Initially L has only one partition that includes all the points in set P' (Line 1). The algorithm takes (K-1) iteration to divide the dataset into K partitions (Line 2). At each iteration i, the algorithm divides the dataset into (i+1) partitions. For example, at first iteration, P' is divided into two non-empty partitions χ_1 and χ_2.

Let $\chi_1 = \{p_1, p_2, ...p_x\}$ and $\chi_2 = \{p_{x+1}, p_{i+2}, ...p_n\}$ be the partition with minimum weighted variance. Now $L = \{\chi_1, \chi_2\}$.

For each partition χ_i in L, we calculate their weighted variance W_i (Line 6). To further divide the points into one more partition, we pick the partition that has the maximum weighted variance among all. Let, W_1 is the weighted variance of χ_1 and W_2 is the weighted variance of partition χ_2. If $W_1 \geq W_2$, then we pick χ_1 in next iteration and further divide χ_1 into 2 more partitions χ_{11} and χ_{12} (Line 8). Now L contains three partitions $L = \{\chi_{11}, \chi_{12}, \chi_2\}$. Thus at ($K$-1) iterations, this process will divide the whole data points into K partitions.

Split(χ) function divides a dataset into two optimal partitions where the summation of weighted variance is minimum. To find that, it iterates over the all points. It considers all possible partitions and pick the partition where the summation of weighted variance is minimum. Runtime of Split(χ) function is $O(n^2)$, where n is the number of points in partition χ. V-OPTIMAL-PARTITION function takes (K-1) iterations. So, the overall runtime of the partitioning algorithm is $O(K.n^2)$.

Definition 3: Density-Level

Density-Level of a point P, denoted by *density-level*(P) is an integer number, labeled by partitioning algorithm described in Section 3.3.1.1. If a point p_i is in j-th partition, the density level of p_i is labeled as j. For two points p and q, if they are in the same partition, their density levels are same. Note that, density-level is only a numerical value.

Consider the example of a sample dataset in Figure 3.3. We implement the partitioning algorithm in the dataset with $K = 2$. The sorted l-density values of all the points are shown in Figure 3.4 with two density levels.

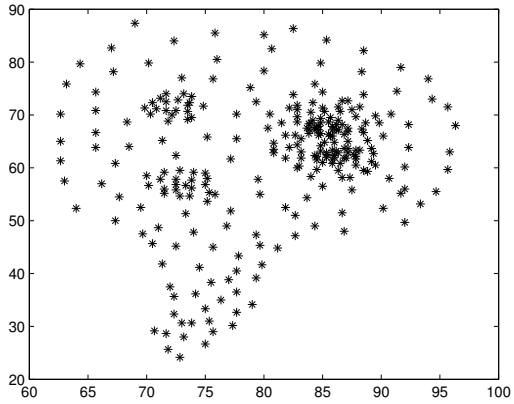

Figure 3.3: Sample dataset

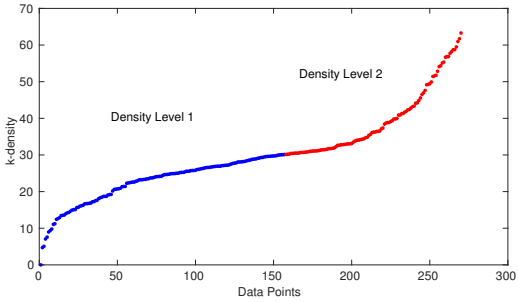

Figure 3.4: Sorted k-density values with 2 density level ($K = 2$)

We use $K = 3$ and implement the V-OPTIMAL-PARTITION algorithm. Figure 3.5 shows the 3 partitions of the dataset.

We plot the datapoints based on the density levels in Figure 3.6. Different color means different density levels. Smaller density level value indicates the points are in the higher density regions. Blue points represents the points in high density region (density level 1) and red points represent the points in lower density region (density level 2).

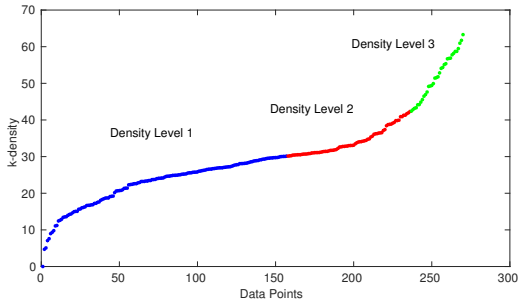

Figure 3.5: Sorted l-density values with 3 density level ($K = 3$)

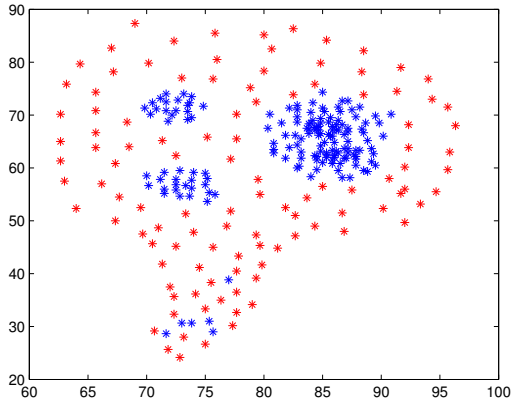

Figure 3.6: Datapoints with two density levels

3.3.2 K-DBSCAN Phase 2 - Density Level Clustering

Step 1 partitions the data points into different density levels. Step 2 is a modified version of DBSCAN algorithm, which considers spatial distance as well as the density level difference between points while clustering.

the K-DBSCAN algorithm does not depend on the single global parameter ϵ, rather each point defines its neighbourhood region dynamically based on its density value. We introduce the idea of *Density level neighbourhood* of a point. *Density level neighbours* of a point P_i are the points that reside inside the neighbourhood region of P_i and that have the same density level as that of P_i.

Consider the example in Figure 3.7. We assume that, density level of all points in C_1 is 1 and density level of all points in C_2 is 2. So, point P_1 is assigned with density-value 2. Only the *blue* points inside P_1's neighbourhood radius are defined as *Density level neighbourhood* of P_1.

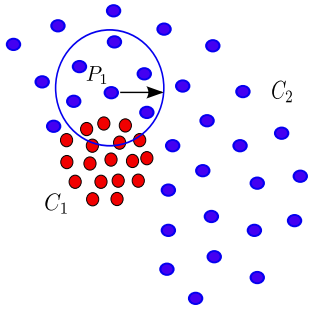

Figure 3.7: Neighbourhood of a Point

Definition 4: Neighbourhood radius of a point (ϵ_i)

The neighbourhood radius of a point P_i is defined as ϵ_i is $\epsilon_i = dist_l(P_i)$.

Definition 5: Density Level Neighbourhood of a point $N(P_i)$

This is defined by $N(P_i) = \{Q \in D \mid dist(P_i, Q) \leq \epsilon_i$ and density-level$(P_i) =$ density-level$(Q)\}$.

The following definitions from 6 to 9 are similar to the DBSCAN definitions except for the differences in neighbourhood that take density level into account in definitions 6 and 7.

Definition 6: Directly density reachable

A point q is directly density reachable from point P_i wrt. ϵ_i if $q \in N(P_i)$.

Definition 7: Density reachable

A point q is density reachable from point P_i if there is a chain of points $p_1.....p_n, p_1 = P_i$ and $p_n = q$ such that p_k is directly density reachable from point p_{k+1}.

Definition 8: Density connected

A point P is density connected to point Q if there is an intermediate point o such that both P and Q are density reachable from point o.

Definition 9: Cluster

A cluster C is a non-empty subset of the whole dataset of points satisfying the following conditions:

1. $\forall\ p, q$: if $p \in C$ and q is density reachable from p, then $q \in C$.
2. if $p, q \in C$: p is density connected to q.

Definition 10: Outliers

A cluster must have at least $MinPts$, which is a user specified parameter. If the number of points in a cluster is less than the threshold $MinPts$, we consider the points as outlier, that do not belong to any cluster.

Clustering Algorithm

In this section, we present our density-based clustering algorithm. The structure of K-DBSCAN algorithm is given in Algorithm 8, which invokes ExpandCluster (see Algorithm 9) method and RegionQuery (see Algorithm 10) method. Input D is the set of data points. Another input DL is an array containing the *density-levels* of all

points generated by a K-means [26] algorithm, that we described in the Partitioning phase (see Section 3.1.1).

Algorithm 8 K-DBSCAN

Input: (i) D: a set of data points $(P_1, P_2, P_3, ...P_n)$

(ii)DL: a set of density level of the corresponding data points $(dl_1, dl_2, dl_3....dl_n)$

(iii)Output: CL: a set of clusters

1. $C = 0$ /* C is cluster id */
2. **for** each unvisited point P_i **do**
3. mark P_i as visited
4. Calculate ϵ_i /* see Definition 4 */
5. $dl_i = DL[P_i]$
6. $NeighborPts = regionQuery(P_i, dl_i, \epsilon_i)$
7. $C = C + 1$
8. $ExpandCluster(P_i, NeighborPts, C, DL)$
9. **end for**

RegionQuery method returns all the density level neighbours of a point (see Definition 5). ExpandCluster method does the cluster formation. If a point P is assigned to a cluster C, its density level neighbours are also part of the same cluster C. This process continues until all the density connected (see Definition 8) points are found (see Algorithm 8).

3.4 Experiments and Comparison with other methods

In this section, we evaluate the effectiveness of our clustering algorithm. We used three different datasets. Dataset-1 is a synthetic dataset, whereas, dataset-2 and dataset-3 are real-world spatial datasets. We compare our method with three well

Algorithm 9 ExpandCluster

Input: (i) D: a set of data points $(P_1, P_2, P_3, ...P_n)$

(ii)DL: a set of density level of the corresponding data points $(dl_1, dl_2, dl_3....dl_n)$

(iii)Output: CL: a set of clusters

1. Assign P_i to Cluster C
2. **for** each point p_j in $Neighbor$ **do**
3. Calculate ϵ_j
4. $dl_j = DL[p_j]$
5. $NeighboursPts_j = regionQuery(p_j, dl_j, \epsilon_j)$
6. **for** all points p_k in $NeighbourPts_j$ **do**
7. **if** DL$[P_i]$ = DL$[p_k]$ **then**
8. $NeighbourPts = NeighbourPts \cup p_k$
9. **end if**
10. **end for**
11. **if** p_j is not yet assigned to any cluster **then**
12. assign p_j to cluster C
13. **end if**
14. **end for**

Algorithm 10 RegionQuery

Input: (i) P_i: i-th data point

(ii)dl_i: density level of point P_i

(iii)ϵ_i: Neighbourhood radius of point P_i

(iv)Outputs: S: a set of neighbour points

1. Return all points within P_i's ϵ_i-neighbourhood (including P_i) and that has the same density-level as dl_i

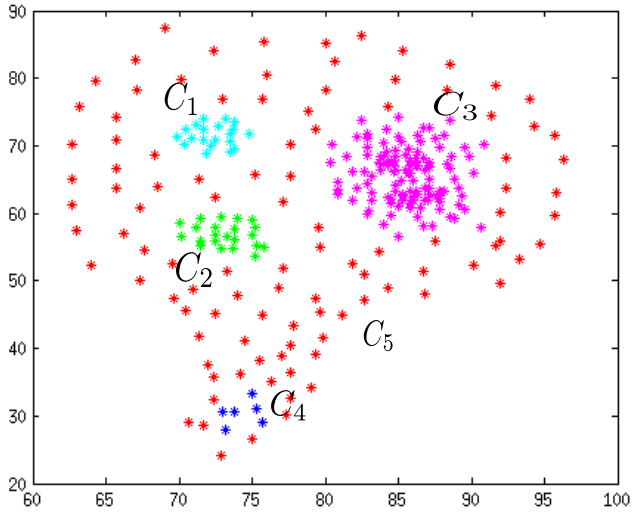

Figure 3.8: K-DBSCAN clustering result, K = 2

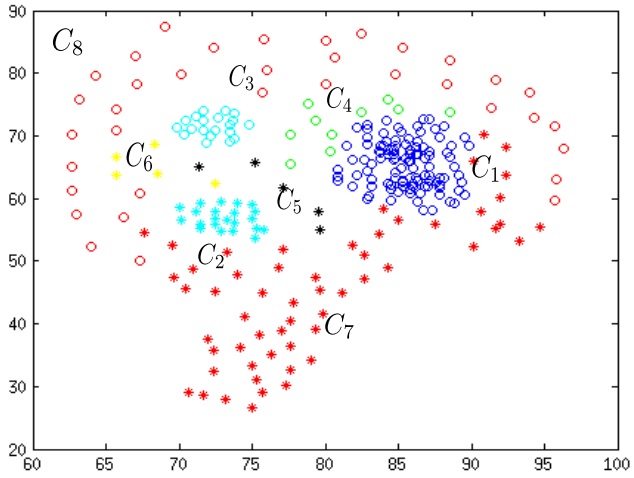

Figure 3.9: K-DBSCAN clustering result, K = 3

known density based clustering algorithm, DBSCAN [2], Shared Nearest Neighbour (SNN) [33] and OPTICS [28].

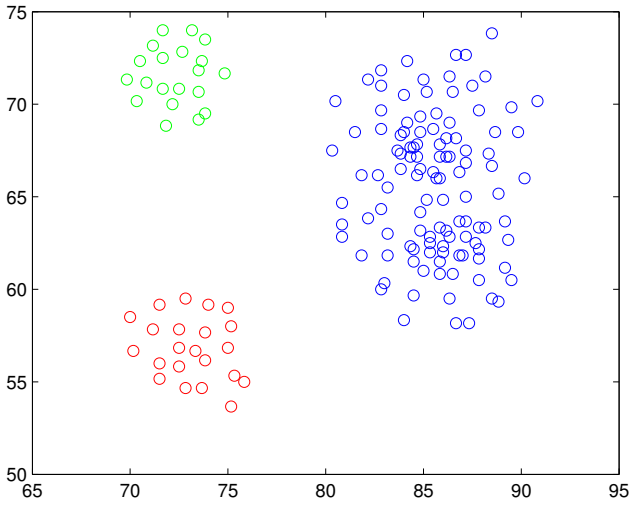

Figure 3.10: DBSCAN clustering result, $\epsilon = 2$

3.4.1 Experiment 1

Dataset-1 is a synthetic dataset, that was also used in [37] (see Figure 3.3). We use K = 2 (2 density levels) and $MinPts = 5$. The result of K-DBSCAN is shown in Figure 3.8. In this result, different colors indicate different clusters. We get 5 clusters. cluster C_5 is the big cluster with lower density level that surrounds the other clusters C_1, C_2, C_3, C_4.

We change the value of K to 3 (3 density levels). Figure 3.9 shows the result. Here we get 8 clusters. Cluster C_8 represents the lowest density level. Cluster C_4, C_5 and C_6 and C_7 are at the middle density level. Whereas cluster C_1, C_2 and C_3 are at the highest density level.

Figure 3.10 shows the result of DBSCAN algorithm with parameters $\epsilon = 2$ and $MinPts = 5$. We got 3 clusters here (red, blue and green points). A large number of points are not being clustered as they are considered as outliers. We change the value of $\epsilon = 4$ and Figure 3.11 shows the result. We get only one big cluster (red points). Obviously one big cluster does not contain any meaningful information.

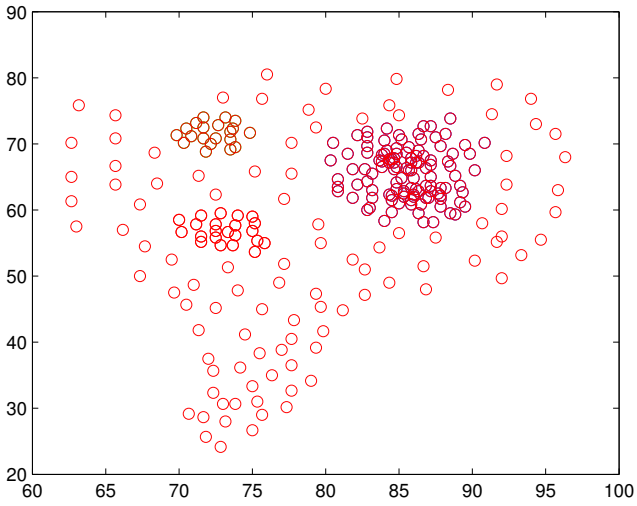

Figure 3.11: DBSCAN clustering result, $\epsilon = 4$

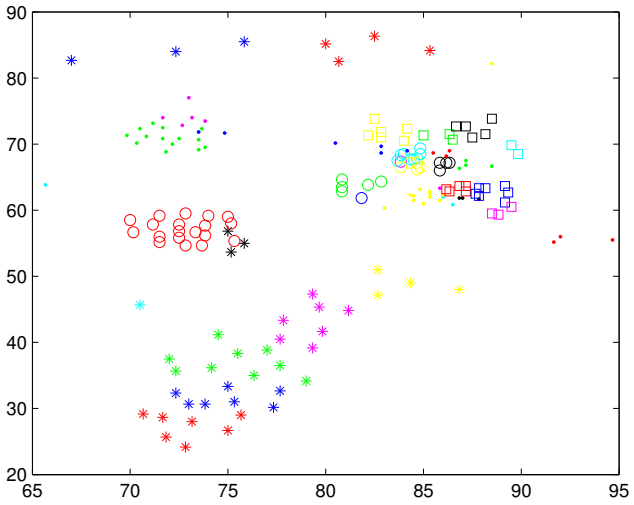

Figure 3.12: SNN clustering result, $k = 15$

Figure 3.12, Figure 3.13 and Figure 3.14 show the results of *Shared Nearest Neighbour* [33] algorithm on the same dataset. In this algorithm, the value nearest neighbour list size, k, is important to determine the granularity of clusters. If k is too small, even a uniform cluster will be broken up into multiple clusters, and the

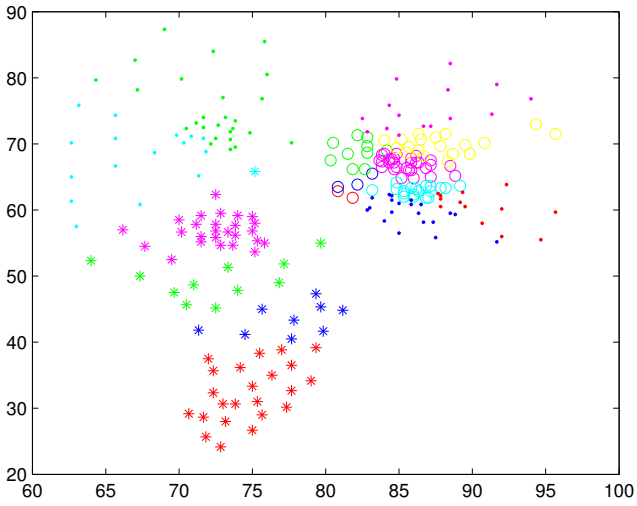

Figure 3.13: SNN clustering result, $k = 30$

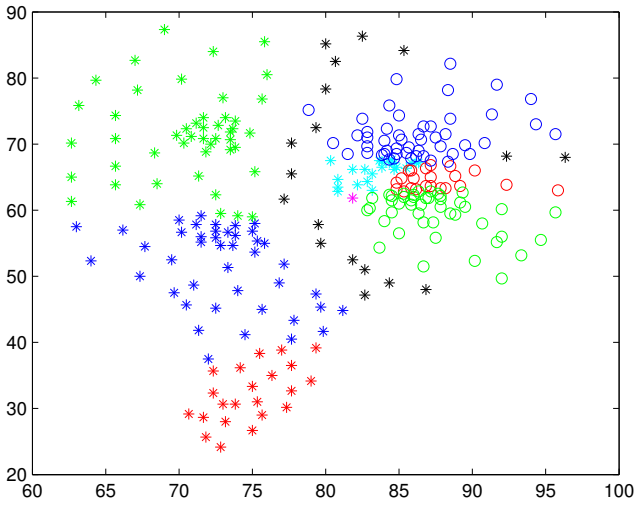

Figure 3.14: SNN clustering result, $k = 50$

algorithm will produce a large number of small clusters. On the other hand, if k is too large, the algorithm will generate only a few, well separated clusters [33]. We use 3 different values for k and compare the results with our algorithm.

Figure 3.12 shows the result of SNN algorithm with $k = 15$. Here this algorithm produces 37 small clusters, with a lot outliers.

Figure 3.13 shows the clustering result with $k = 30$. We get 18 clusters. We can see that cluster C_3 (From Figure 3.10) has been broken into multiple different clusters. Similarly, cluster C_5 (From Figure 3.10) has also been broken down into multiple clusters. Also, we find the points in different density levels are mixed together and form a single cluster.

Figure 3.14 shows the result with $k = 50$. Here, we get 8 clusters. But still one of the dense clusters (Cluster C_1 from Figure 3.9) has been broken into multiple small sized clusters. Further, some points from different density levels are merged together to form a single cluster.

Now, we compare our method with OPTICS [28] algorithm. From a set of points, OPTICS [28] generates an ordering of points and corresponding reachability values [28]. Using the reachability plot, clusters of different densities can be obtained. Figure 3.15 shows the reachability plot obtained by OPTICS algorithm for Dataset-1. In this plot, x-axis displays the order in which OPTICS visits the points. Whereas, y-axis displays the reachability distance of corresponding points. Each valley represents a cluster. The deeper the valley, the more dense the cluster.

Extracting clusters can be done manually by selecting a threshold on the y-axis. We can clearly see the threshold value = 4.5 will give us a good result. Figure 3.16 shows the clustering result. It is clear that, clusters contain points of varying densities, but the big overlapping cluster is missing here (cluster C_5 from Figure 3.8).

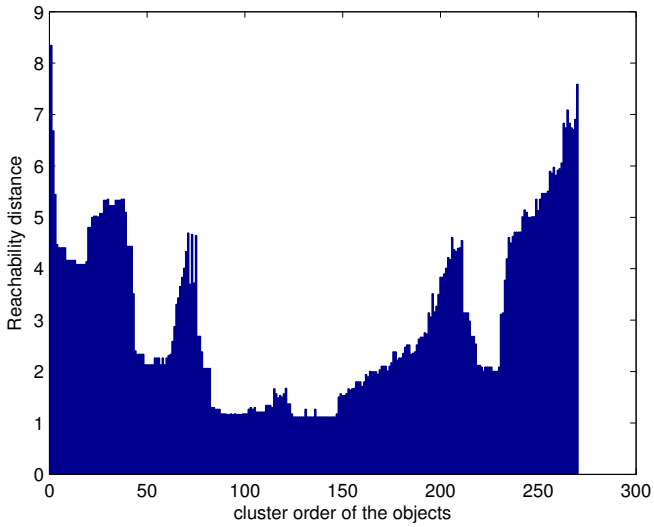

Figure 3.15: Reachability distance

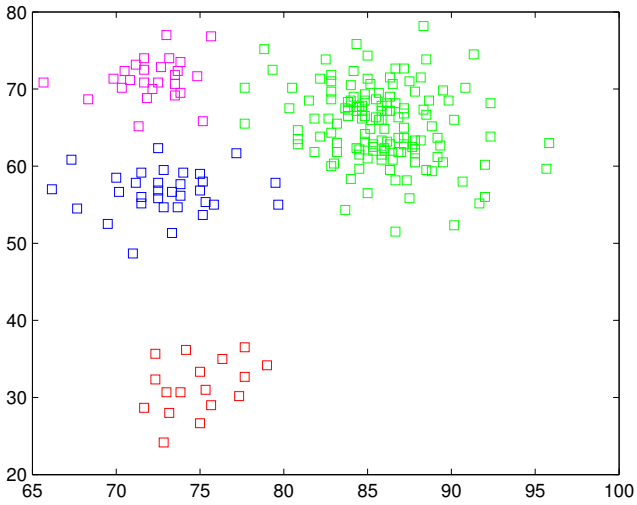

Figure 3.16: OPTICS clustering result

3.4.2 Experiment 2

In the second experiment, we use a real-world spatial dataset (Dataset-2), generated from OpenStreetMap[1], which covers a small area of Dhaka city in Bangladesh

[1]http://www.openstreetmap.org

(see Figure 3.17). We parsed all house or apartment locations in this region. This dataset contains 325 points. Figure 3.18 shows the K-DBSCAN clustering result using K = 2 (2 density levels). Our algorithm generates a total of 8 clusters. Cluster C_1 to C_7 are the most dense clusters, whereas cluster C_8 represents the lower density.

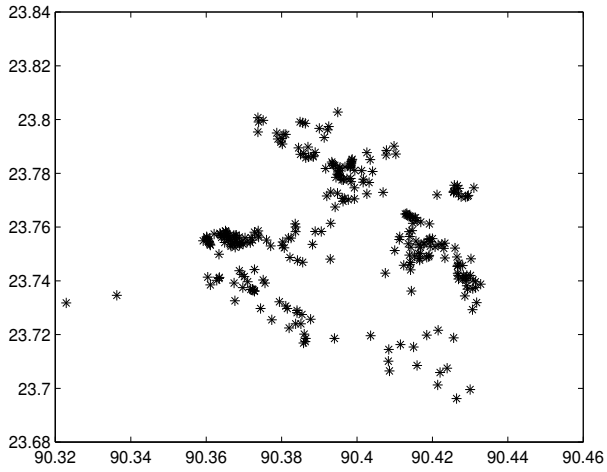

Figure 3.17: Dataset-2

To compare with DBSCAN, we first use value $\epsilon = 0.004$ and $MinPts = 5$ (see Figure 3.19). Points that formed cluster C_8 in our algorithm (see Figure 3.18) are not being clustered using these parameters, because they are considered as outliers. We increase the value of ϵ to 0.009 and implement DBSCAN again (see the result in Figure 3.20). Here we find the same cluster(magenta points) as cluster C_8 (Figure 3.18), but clusters C_1 to C_5 merged together to form a single cluster (red points).

We implement Shared Nearest Neighbour (SNN) [33] algorithm on dataset 2. We use nearest neighbour size $k = 50$. Figure 3.21 shows the result. We get a total of 8 clusters. Points that formed cluster C_8 in our algorithm (see Figure 3.18) are not being clustered in this result.

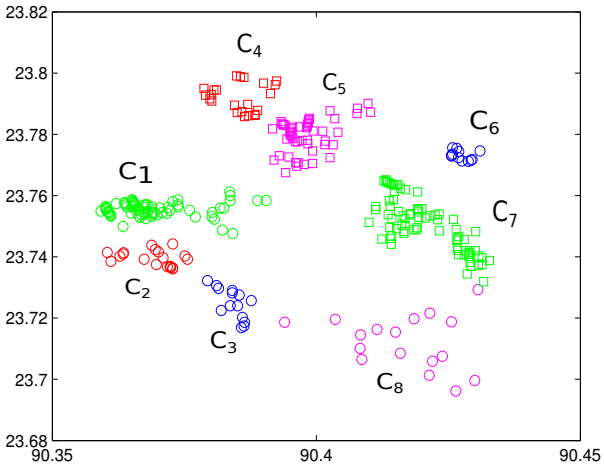

Figure 3.18: K-DBSCAN, K=2

Next we implement OPTICS [28] algorithm on dataset 2. We get a total of 4 clusters. Still the points that formed cluster C_8 in our algorithm (see Figure 3.18) are not being clustered in this result (see Figure 3.22).

3.4.3 Experiment 3: Population Dataset

We have used a real world geographical dataset Dataset-3. This dataset gives the population density for Texas state for year 1990. The dataset is gridded with 0.25×0.25 degree resolution. Each grid cell contains the count of the number of people inside it. For the experiments in this chapter, we represent each population count of 50 persons by 1 point. Within each grid, if there were n people residing in it, we generated $n/50$ locations (latitude, longitude) so that each point represents 50 persons.

We use *K-DBSCAN* algorithm on dataset-3. We use density level $K = 4$. We obtain total 69 clusters (see Figure 3.24). There are 16 clusters, that represents the highly densely populated area (highest density level). Whereas, 29 clusters are in

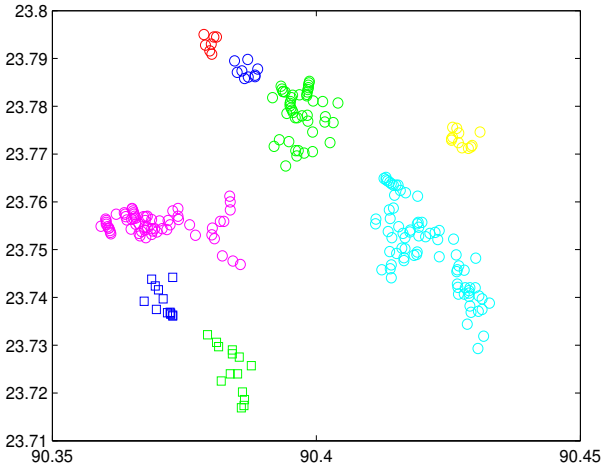

Figure 3.19: DBSCAN, $\epsilon = 0.004$

second highest density levels. In third highest density level (2nd lowest density level) has 17 clusters and 7 clusters are in the lowest density levels. Note that, we get some overlapping clusters, that surrounds other dense clusters.

Figure 3.25 shows the result of DBSCAN algorithm on the same dataset. This algorithm generates total 29 clusters.

Figure 3.26 shows the result of OPTICS algorithm on the same dataset. This algorithm generates total 18 clusters.

Figure 3.27 shows the result of SNN algorithm on the same dataset. This algorithm generates total 123 clusters.

3.4.4 Qualitative Measure of Clustering Results:

We attempt to measure the qualitative measure of clustering result of K-DBSCAN algorithm. Our goal is to differentiate the most dense regions from lower density regions. Motivated by this goal, we define a cluster quality measure that is based on density variation of the points in the clusters. First, we calculate the average density

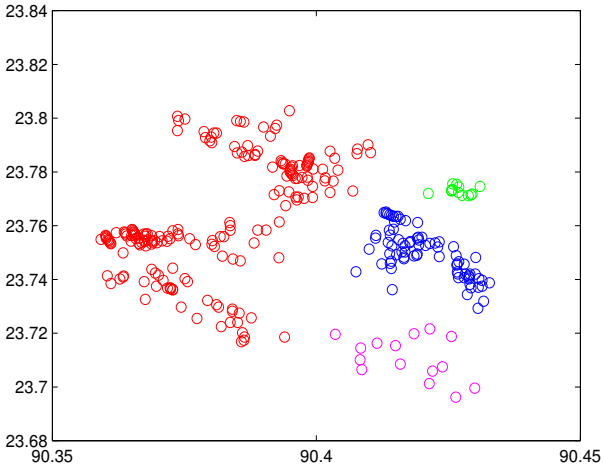

Figure 3.20: DBSCAN, $\epsilon = 0.009$

of all points of a cluster. Then we get the standard deviation of density values of that cluster. In addition to this, we consider the noise penalty to penalize an excessive number outliers. The formulas for these density quality measures are given in (3.4) - (3.8).

$$a_i = \frac{1}{|C_i|} \sum_{x \in C_i} density(x) \tag{3.4}$$

$$\sigma_i = \sqrt{\frac{\sum_{x \in C_i} (a_i - density(x))^2}{|C_i|}} \tag{3.5}$$

$$NoiseP = \frac{Total\ number\ of\ outliers}{Total\ number\ of\ Points} \tag{3.6}$$

$$MDV = \frac{1}{num_{cluster}} \sum_{i=1}^{num_{cluster}} \sigma_i \tag{3.7}$$

$$QMeasure = MDV + NoiseP \tag{3.8}$$

Table 3.1 compares the qualitative measure of our clustering algorithm on Dataset-1 with the other different algorithms, whereas, Table 3.2 and Table 3.3 show the qualitative measure on Dataset-2 and Dataset-3. The lower the QMeasure value,

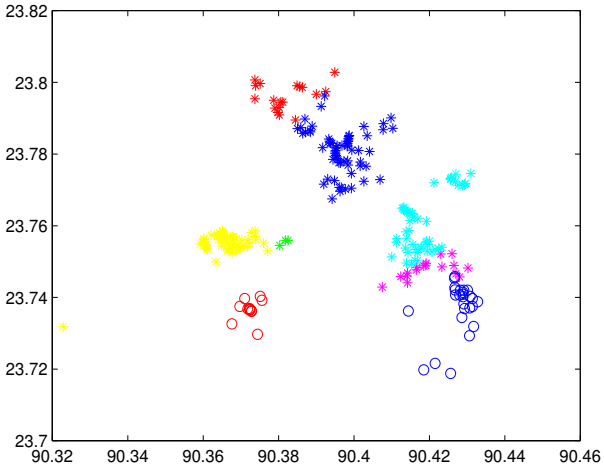

Figure 3.21: SNN, $k = 50$

the better the result. We can clearly see that, our algorithm performs better on all three cases than all other algorithms.

We then incorporate a measure of spatial contiguity of clusters in addition to the density and noise measures, by using SSE (Sum of squared error) [38], normalized to 0 to 1 range, (3.9) and (3.10).

$$SSE = \sum_{i=1}^{num_{cluster}} \left(\frac{1}{2|C|} \sum_{x \in C_i} \sum_{y \in C_i} dist(x, y)^2 \right) \tag{3.9}$$

$$Total = MDV + NoiseP + SSE \tag{3.10}$$

Table 3.4, 3.5 and 3.6 show the results for Dataset-1, Dataset-2 and Dataset-3 respectively. Smaller value indicates better result. Dataset-1 gives us better result for $K = 3$ or $K = 4$. With 2 density levels ($K = 2$), we get one big overlapping cluster (see Figure 3.8). That is why we get a larger value of SSE. As the density levels increase, the big overlapping cluster is broken into multiple well separated clusters, which gives us a better result (see Figure 3.9). Qualitative results of DBSCAN

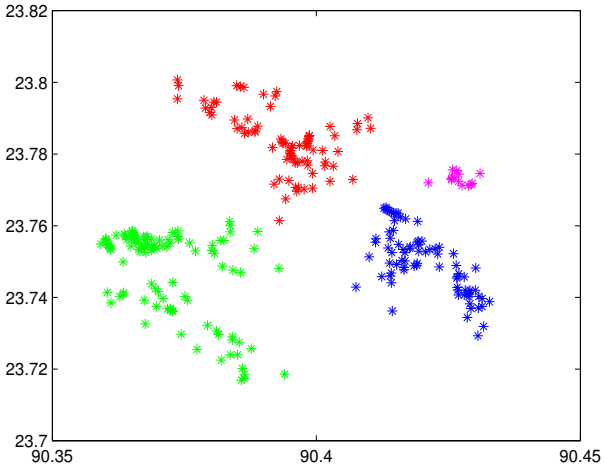

Figure 3.22: OPTICS

change dramatically for change of parameter values. Whereas, our method gives comparatively consistent results when changing the parameter value (K).

Our method gives us the best result in Dataset-2. As there are no overlapping clusters, we get 8 well separated clusters (see Figure 3.18). Our method gives us better results for Dataset-3 also. In this result we get both well separated clusters and overlapping clusters. DBSCAN and SNN ends up with large number of outliers (larger value of NoiseP). Whereas OPTICS creates clusters merging points from various density levels (larger value of MDV).

3.5 Practical Applications of K-DBSCAN

3.5.1 Analysis of Earthquake Data

In order to illustrate the practicability of our *K-DBSCAN* method, we use a real-world spatial dataset of earthquakes. The data used in this chapter is obtained from Northern California Earthquake Data Center (NCEDC), which is located at the Berkeley Seismological Laboratory. The data center is a joint project of Berkeley

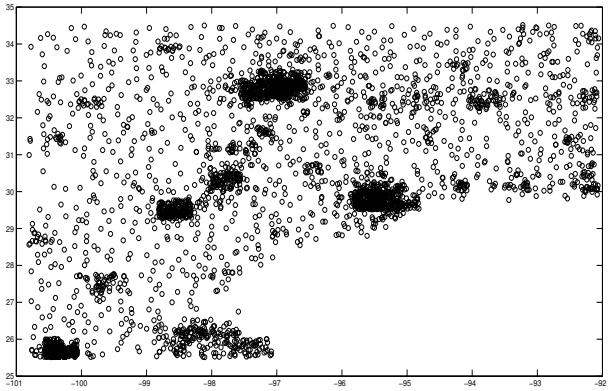

Figure 3.23: *Dataset-3*

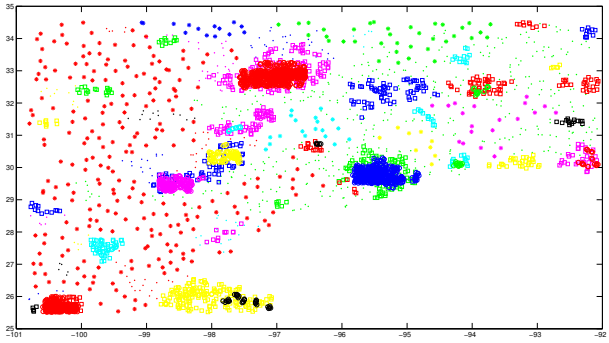

Figure 3.24: K-DBSCAN, K = 4

Seismological Laboratory and U.S. Geological Survery (USGS). The dataset is accessible for public use since 1992. We collect data from 1990 to 2015. The earthquake locations that has magnitude equal to or larger than 3 are considered as complete earthquakes in this chapter. We only use the complete earthquakes in this experiment.

The clustering results of earthquake data can benefit us to recognize the active faults or understanding the changing trend of earthquakes. In this chapter, we implement *K-DBSCAN* method in this earthquake dataset to find clusters in different density levels. Clusters in the highest density level indicates the regions are more

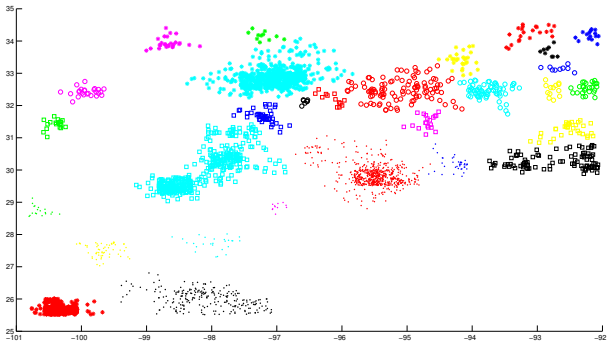

Figure 3.25: DBSCAN, ϵ=0.18

Algorithm	MDV	NoiseP	QMeasure
K-DBSCAN (K=2)	0.3503	0.0037	0.354
K-DBSCAN (K=3)	0.3080	0.029	0.337
K-DBSCAN (K=4)	0.2451	0.085	0.330
SNN (k=15)	0.4354	0.263	0.699
SNN (k=30)	0.4966	0.103	0.597
SNN (k=50)	0.7328	0	0.733
OPTICS	0.6364	0.222	0.859
DBSCAN ($\epsilon = 0.2$)	0.2278	0.333	0.561
DBSCAN($\epsilon = 0.4$)	0.8861	0.0111	0.897

Table 3.1: QMeasure for Dataset-1

prone to earthquakes. *DBSCAN* and other spatial clustering algorithms failed to do that because they find clusters only in single density level.

Figure 3.28 shows the result of *K-DBSCAN* algorithm with density level $K = 3$. The algorithm generates 18 clusters. We get 6 clusters in high density levels. There are 8 clusters in the second highest density levels and 4 clusters in the lowest density levels.

We compare this result with *DBSCAN* method. We use $\epsilon = 0.25$ and $MinPts = 5$ (see Figure 3.29). We choose this value for ϵ based on the *k-distance* heuristic method that was proposed in [2]. It generates 4 clusters. We can see that, it generates a big cluster (red color) combining most of the points in the dataset.

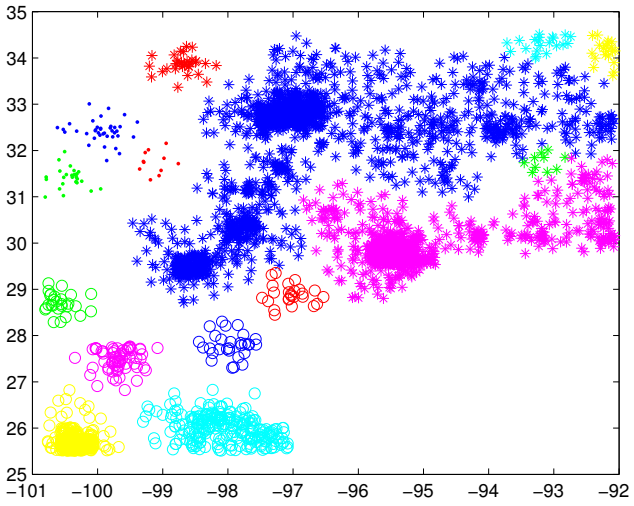

Figure 3.26: OPTICS

Algorithm	MDV	NoiseP	QMeasure
K-DBSCAN(K=2)	0.00083	0.0565	0.0573
SNN(k=50)	0.0017	0.0598	0.0615
OPTICS	0.0011	0.664	0.675
DBSCAN($\epsilon = 0.004$)	0.0006	0.698	0.6986

Table 3.2: QMeasure for Dataset-2

3.5.2 Analysis of Crime Data

Crime dataset is a classic example of spatial data. Determination of clustered crime region can be utilized to recognize crime hot spot regions. It may help police departments to plan the investigation against crime. Crime data analysis may also benefits police to predict crime. People may utilize the analysis result to find a safe neighborhood.

The crime data we used in this chapter is obtained from Chicago data portal.This dataset is provided by Chicago city police. It reflects reported incidents of crime that occurred in city of Chicago. Data is extracted from the Chicago Police

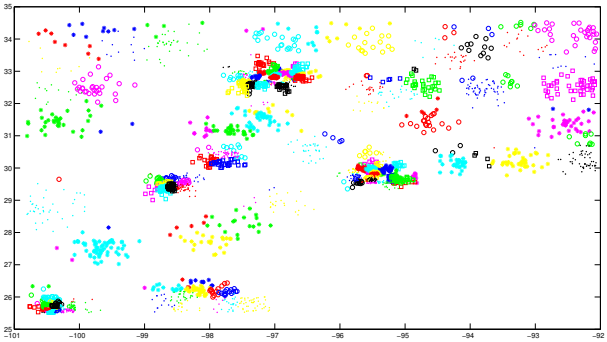

Figure 3.27: SNN, k=50

Algorithm	MDV	NoiseP	QMeasure
K-DBSCAN (K=4)	0.0350	0.0379	0.0729
SNN (k=50)	0.0224	0.2014	0.2238
OPTICS	0.0511	0.0727	0.1238
DBSCAN ($\epsilon = 0.2$)	0.0387	0.1518	0.1905

Table 3.3: Qualitative Measure on Dataset-3 based on density variation and Noise

Department's CLEAR (Citizen Law Enforcement Analysis and Reporting) system. Data is accessible for public use.

In this chapter, we implement *K-DBSCAN* algorithm in real world crime dataset to find clusters in different density levels. Our motivation behind implementing *K-DBSCAN* algorithm in this dataset is to discover the crime hotspot regions. Instead of finding hotspots in a single density level, our algorithm can generate crime hotspots regions in multiple density levels. Some regions have higher crime rate, whereas some regions have lower crime rate. Identifying regions based on crime rate density can benefit police department, because they may increase enforcement in higher density crime regions rather than lower density crime regions. So, it is important to identify the crime zone in different levels.

Algorithm	MDV	NoiseP	SSE	Total
K-DBSCAN (K=2)	0.350	0.004	0.404	0.758
K-DBSCAN (K=3)	0.308	0.029	0.299	0.636
K-DBSCAN (K=4)	0.245	0.085	0.273	0.603
SNN (k=15)	0.435	0.263	0.004	0.703
SNN (k=30)	0.497	0.103	0.0512	0.649
SNN (k=50)	0.733	0	0.0512	0.784
OPTICS	0.636	0.222	0.0570	0.916
DBSCAN ($\epsilon = 0.2$)	0.228	0.333	0.024	0.585
DBSCAN ($\epsilon = 0.4$)	0.887	0.011	0.511	1.401

Table 3.4: Total Qualitative Measure for Dataset-1

Algorithm	MDV	NoiseP	SSE	Total
K-DBSCAN (K=2)	0.0008	0.057	0.020	0.078
SNN (k=50)	0.0017	0.059	0.045	0.106
OPTICS	0.0011	0.664	0.021	0.697
DBSCAN ($\epsilon = 0.004$)	0.0006	0.698	0.131	0.830

Table 3.5: Total Qualitative Measure for Dataset-2

In this dataset, the crime incidents are classified into 13 types. Theft and Burglary are one of the most occurred crime incident. In this chapter, we use theft crime data and burglary crime data for our experiment.

3.5.2.1 Theft Data

The Theft dataset reflects the geolocations of reported incidents of theft crime of Chicago city. We use the data from January to April of year 2014. Dataset used in this chapter contains $17,108$ geo-location points (latitude, longitude).

In the first experiment with theft dataset, we use density level parameter $K = 3$. Point distribution in different density levels are shown in Table 3.7. Table shows number of points, number of clusters and maximum *k-distance* (in km) in each density level. Figure 3.30 shows the *K-DBSCAN* clustering result. our method generates 83 clusters. The algorithm identifies 39 crime hot spot regions as the highest density

Algorithm	MDV	NoiseP	SSE	Total
K-DBSCAN (K=4)	0.0350	0.0379	0.049	0.122
SNN (k=50)	0.0224	0.2014	0.0023	0.226
OPTICS	0.0511	0.0727	0.0023	0.126
DBSCAN ($\epsilon = 0.2$)	0.0387	0.1518	0.0069	0.197

Table 3.6: Total Qualitative Measure on Dataset-3

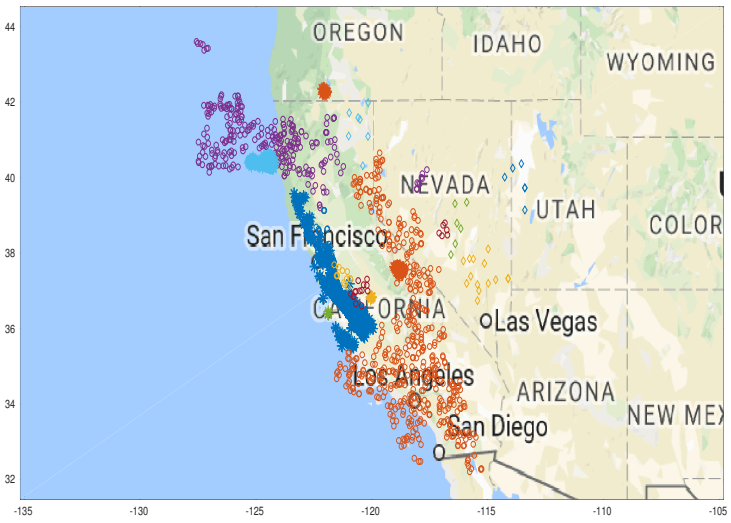

Figure 3.28: *K-DBSCAN clustering result,* $K = 3$

regions. In the second highest density level, there are 32 clusters. We find 12 clusters are in the lowest density level.

In the second experiment, we change the value of density level parameter K to 4. Point distribution and the number of clusters in each density levels are shown in table 3.8. We can see from the result that, the highest dense region from Figure 3.30 has been divided into two density level regions. Figure 3.31 shows the result. Method generates 116 clusters. It identifies 45 hot spots in highest density level regions, 4 in second highest density levels. There are 62 clusters in the third highest density level and 5 hot spots in the lowest density level.

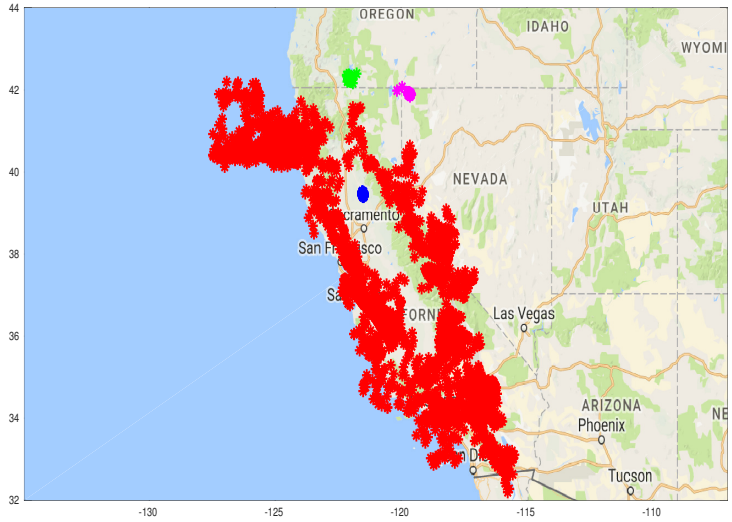

Figure 3.29: *DBSCAN clustering result with EarthQuake Data*

Density Level	Total Points	No. of clusters	k-distance(km)
1	7245	39	6.7055
2	2411	32	12.3478
3	5978	12	22.5335

Table 3.7: Point distribution of theft data, $K = 3$

We compare our results with *DBSCAN* method. *DBSCAN* Algorithm generates only clusters with one density level. Figure 3.32 shows the *DBSCAN* clustering result with Theft data. We use $\epsilon = 0.003$, $MinPt = 10$. We get 34 clusters. Algorithm generates 1 big cluster (blue cluster). In this experiment, we choose the value for ϵ by using a *k-distance* graph. We plot the distance to the $k = 5$ nearest neighbor ordered from the largest to the smallest value. Good values of ϵ are where this plot shows an *elbow*.

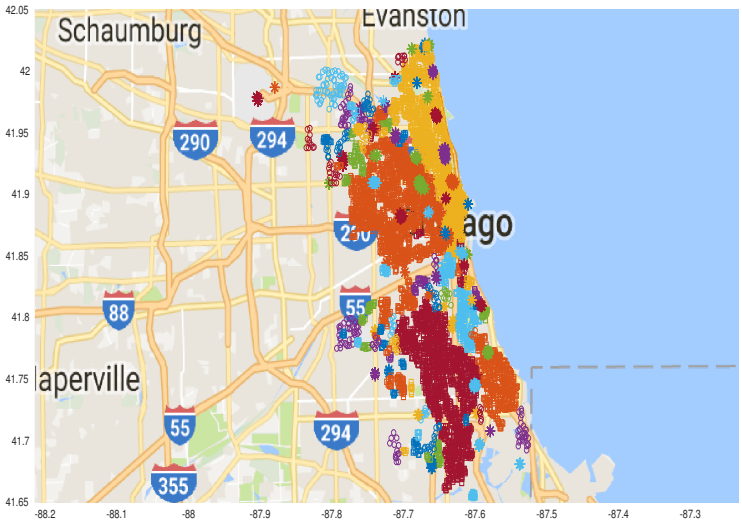

Figure 3.30: *K-DBSCAN result with theft Data, K = 3*

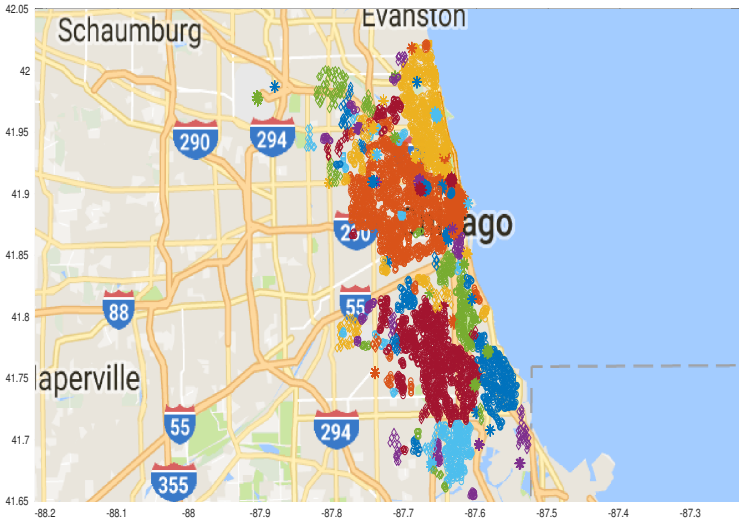

Figure 3.31: *K-DBSCAN result with theft Data, K = 4*

Density Level	Total Points	No. of clusters	k-distance(km)
1	7245	45	6.7055
2	2411	4	12.3478
3	5978	62	22.5335
4	1471	5	186.3751

Table 3.8: Point distribution of theft data, $K = 4$

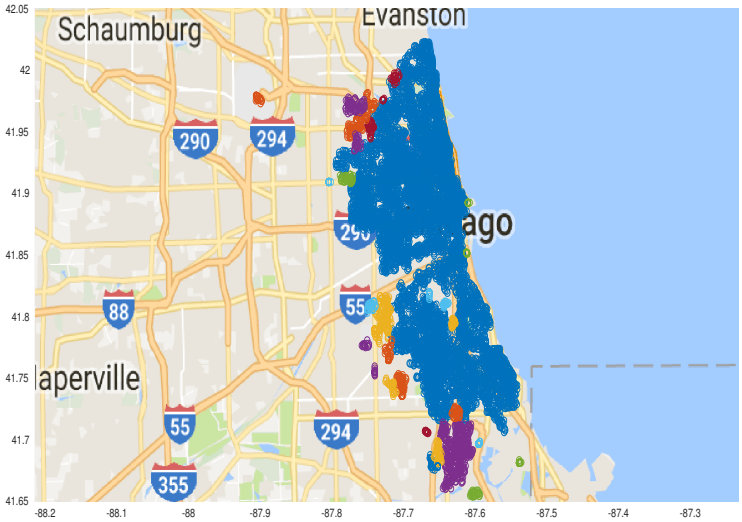

Figure 3.32: *DBSCAN result with theft Data,* $\epsilon = 0.003$

3.5.2.2 Burglary Data

The burglary dataset reflects the location of reported burglary incident of Chicago city. In this chapter, we use burglary data of year 2014. Dataset used in this chapter contains $14,194$ geo-location points (latitude and longitude).

In the first experiment with this data, we use the value of density level parameter $K = 3$. Point distribution and number of clusters in different density levels are shown in Table 3.9. Algorithm generates 215 clusters. 113 clusters represent the region in the highest density level. There are 83 clusters in the second highest density level. There are 19 clusters are in the lowest density level. Figure 3.33 shows the result.

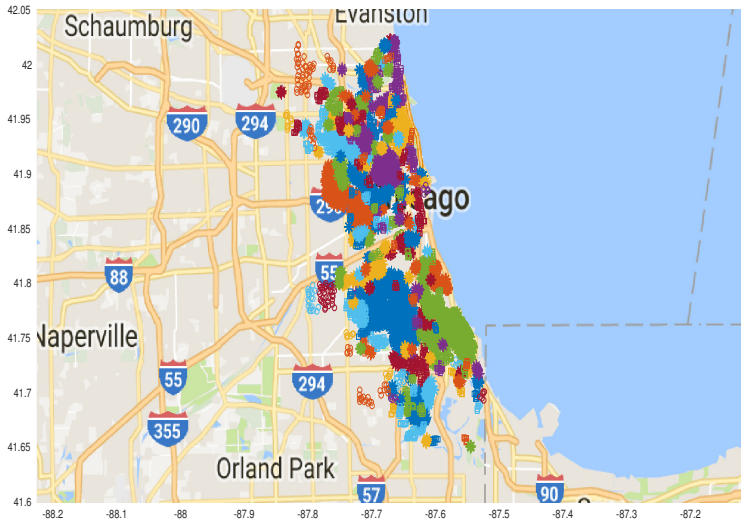

Figure 3.33: *K-DBSCAN clustering result, K = 3*

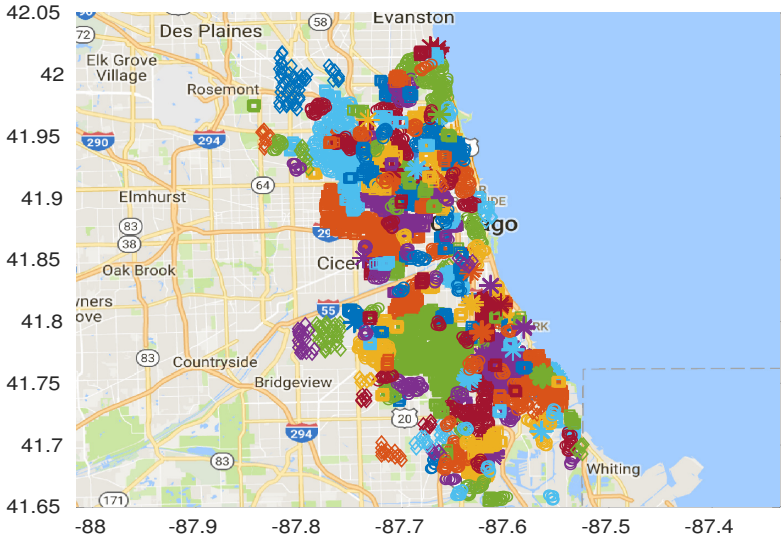

Figure 3.34: *K-DBSCAN clustering result with Burglary Data, K = 4*

Density Level	Total Points	No. of clusters	k-distance (km)
1	3703	113	10.1338
2	6589	83	16.3056
3	3112	19	25.0108

Table 3.9: Point distribution of burglary data, $K = 3$

Density Level	Total Points	No. of clusters	k-distance (km)
1	3703	33	10.1338
2	6589	106	16.3056
3	3112	85	25.0108
4	790	20	117.5724

Table 3.10: Point distribution of burglary data, $K = 4$

Next we change the value of $K = 4$. Point distribution and number of clusters in all density levels are shown in Table 3.10. We see that the points in the level 1 (from Table 3.9) has been divided into 2 density levels. 33 clusters represent the regions in the highest density level and there are 106 clusters generated in the second highest density level.

We compare our result with *DBSCAN* clustering algorithm. We choose parameter $\epsilon = 0.003$ and $MinPt = 10$. Algorithm generates 25 clusters. All clusters are in a single density level. Figure 3.35 shows the result. Here algorithm generates 2 big clusters.

3.5.3 Parameter K

In this algorithm, parameter K defines the number of density levels in the dataset. In each density level, clusters are generated based on the spatial distance between each points. Clustering result of this algorithm differ with different value of K. Note that, if we use $K = 1$, the result will be similar to *DBSCAN* algorithm.

Value of K depends on the distribution of points in the dataset in different density level. But user can choose the value of K based on their desired clustering

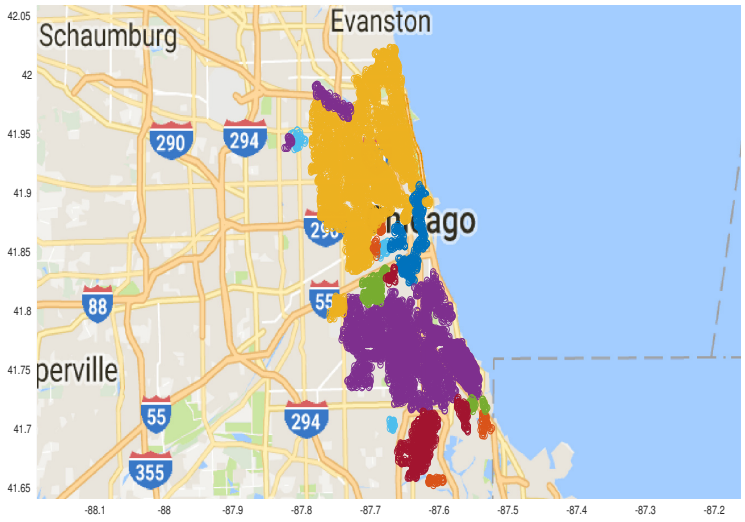

Figure 3.35: *DBSCAN clustering result with Burglary Data,* $\epsilon = 0.003$

result. For example, with population data users may want to get the regions only two density levels: regions which is high population and regions with low population. In this case, users need to use $K = 2$ and they will get their desired clustering result.

In this section, we propose a heuristic approach to find a good value of density level parameter K. We divide the data points into different density groups based on their density value. Given the number of density level, our goal is to assign the data points into the level so that the density variation of the points in each level is nearly similar. In other words, we want to minimize the density variation of points in each level.

We use the motivation of *elbow method* to determine the value of K. The idea behind the method is, we should choose the number of density levels so that adding another density level doesn't give us much better quality of the clustering result. We plot the value of K against the mean of density variation of clustering results for the value of K. Figure 3.36 shows the result. We see that with value of $K = 6$ does not

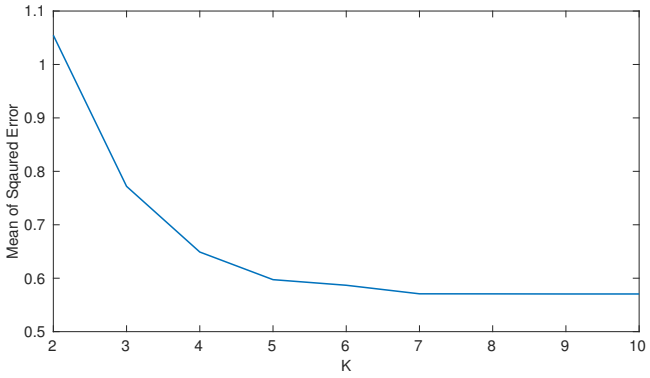

Figure 3.36: *Mean of Squared Error with different value of K*

give much better than the result with $K = 5$. After $K = 7$, cluster quality remains almost similar. In this scenario, $K = 5$ identifies the good value of K.

3.6 Conclusion

In this chapter, we propose a density-based clustering algorithm which can handle arbitrary shaped clusters and datasets with varying densities. We did experiments on both synthetic data and real-world spatial data. Unlike the *DBSCAN* algorithm, ***K***-*DBSCAN* does not depend on a single global density threshold ϵ, which is difficult to determine. Rather, we have to find a threshold for density level ***K***, which can be easily determined from the points density distribution. Experimental results show that our algorithm can correctly cluster the data points that have different density levels and different shapes. We compared the clustering quality of our algorithm with 3 other well known algorithms. We illustrated the practicability of our algorithm using two real world spatial datasets.

CHAPTER 4

Preference-Aware POI Recommendation With Temporal and Spatial Influence

4.1 Introduction

There have been vast advances and rapid growth in Location based social networking (LBSN) services in recent years. Foursquare[1], Yelp[2] and Facebook Places[3] are a few of the examples of LBSN services. LBSNs allow users to share their life experiences via mobile devices. A user posts his/her presence or arrival to a physical location, which is known as a process of "Check-in". She can also share her experiences by leaving comments or tips on that location. A *Point of Interest* (POI) location can be a "Restaurant", "Travel spot", "Park" and so on.

The task of *POI recommendation* is to provide personalized recommendation of POI locations to mobile users. Both LBSN users and POI owners can exploit the benefit from the recommendation service. Users can find better POIs and have better user experiences via the right recommendation. A POI owner could exploit this service to acquire more target customers [4]. So, a POI recommendation system is a very important application in LBSN services.

User's movement data with location information provide us better knowledge about their activities and interests. For example, people who often visit a gym, must be interested in physical exercise. Also, people who visit the same place may share

[1] `www.foursquare.com`

[2] `www.yelp.com`

[3] `www.facebook.com/places`

the same interest. Location histories and opinions of one user can be exploited to recommend an unvisited location to another user if they share the same interest.

A POI recommendation system provides users with list of locations that should match their personal interests within a geospatial range [39]. Here are some factors that may influence a user to make a decision to choose a POI.

1) *Personal preferences*: Personal preferences of different users are different. For example, a food lover is more likely to be interested in exploring better quality restaurants, whereas, a health conscious user may be interested in finding a better place for walking or running.

2) *Spatial Influence*: Geographical position of a POI location plays an important role. People often tend to visit nearby places. The probability of a user visiting a place is inversely proportional to the geographical distance of the POI from the current location of the user [6].

3) *Temporal Influence*: User activities are significantly influenced by time [6]. For example, users are more likely to visit a restaurant rather than a bar at noon. Parks or other recreational places attract a lot of visitors in the weekend rather than weekdays.

4) *Popularity*: Choosing a POI can be influenced by the popularity or rank of a POI. People may visit a far away place if the place is very popular.

In this chapter, we propose a preference-aware, location-aware and time-aware POI recommendation system that offers a particular user a set of POI locations incorporating time information and geo-spatial range. The contribution of this chapter can be summarized as follows:

1) We incorporate time dimension to model time-specific user preferences, so our recommendation model aims to recommend POI locations that match the time-specific preferences of individual user.

2) We further exploit user's spatial behaviour using location histories to generate spatial-aware location preferences.

3) Our recommendation model uses the popularity factor of individual locations by calculating both time-specific popularity and regional popularity.

4) We model personal preferences of users based on the category information of their location histories. We estimate the similarity between two users by computing similarity between their personal preferences rather than using user's location vector. There are 2 main reasons behind this. First, it handles the data sparsity problem of user-location matrix. Second, two users who do not visit the exact same venue may still share common interest if their preferences are the same.

5) We evaluate our system with a real-world dataset collected from Foursquare. The extensive experimental results with evaluation show that our method combining multiple factors (temporal, spatial, popularity, preferences) provide users better and effective recommendations than other baseline approaches.

The rest of the chapter is organized as follows. In Section 4.2, we review some related work. We present the basic *Preference-Aware location recommendation* model in Section 4.3. In Section 4.4, we proposed a method to incorporate time influence in basic preference-aware location recommendation. In Section 4.5, we describe the method of utilizing spatial influence for further enhancement of our method. Section 4.6 presents experimental results of our algorithm. Finally, Section 4.7 concludes the chapter.

4.2 Related Work

There have been many studies to design POI recommendation algorithms. Two popular approaches are Collaborative Filtering algorithm and Non-Negative Matrix Factorization algorithm.

Collaborative Filtering (*CF*) algorithms are divided into two major categories. 1) *Memory-based CF* and 2) *Model-based CF*. *Memory-based CF* methods are further divided into two categories. 1) *User-based CF* and 2) *Item-based CF*.

In [40], the *User-based CF* approach considers a combination of social influence and spatial influence. Their experiments report that geographical influence has a significant impact on the accuracy of POI recommendation, whereas the social friend link contributes little. Their results also indicate that *user-based CF* works much better than *Item-based CF*. In [6], the authors exploit spatial influence as well as temporal influence for building a recommendation model. They incorporate time factors in the basic *CF based* model by computing similarity between two users by considering check-in information at a specific time t, rather than that of all times.

User similarity is computed based on check-in location history of two users. User's categorical preferences have not been considered in these works. In general, two users who do not visit the same venue may have similar preferences. Also, the large scale of user-location data suffered from data sparsity problem, which is a big challenge for CF based algorithm.

In [7], authors explore user preferences with social and geographical influence for POI recommendation. They model user preferences using predefined categorical information of location data. In [4], the authors propose a geographical probabilistic factor analysis framework for recommendation that takes various other factors into consideration, viz. user-item preferences, POI popularity and geographical preferences of individual users. In [41], the authors proposed a friendship based collaborative filtering (FCF) approach for POI recommendation.

4.3 Problem Definition

The problem of personalized POI recommendation is to recommend a set of POIs to a user. In this chapter we used four key data structures: 1) User, 2) POI location, 3) Check-in and 4) Category hierarchy.

1) Each user u is represented by a unique id. Let $U = \{u_1, u_2, u_3, ...u_n\}$ be a set of users.

2) Each POI location is associated with a unique POI id, geographical position (latitude and longitude) and category information. Let $L = \{l_1, l_2, l_3, ..., l_m\}$ be set of POI locations.

3) "*Check-in*" is a process by which a user u announces his physical arrival or presence at a venue in location based social network. Let $Ch_{ij} = \{u_i, l_j, t\}$ be a *check-in* tuple, which represent that user u_i checked in POI l_j at time t.

4) Each POI location is associated with a category which represent its functionality. For example, a location can be a "Restaurant", "Museum", or "School" etc. In this chapter, we use two level category hierarchy obtained from Foursquare[4]. In Foursquare, there are 8 primary categories. Each primary category includes other sub-categories. For example, "Food" is a primary category, it includes 78 sub-categories, such as "Chinese Restaurant", "Indian Restaurant", "Cafe" etc. Let $CT = \{ct_1, ct_2, ..., ct_8\}$ be a list of primary categories. Let $SCT = \{sct_1, sct_2, sct_3, ...sct_k\}$ be a list of sub-categories. Each sct_i is associated with only one primary category ct_m. Each POI location l_j is associated with exactly one primary category and one sub-category.

Table 4.1 shows the location category hierarchy defined by Foursquare [42].

[4] `https://developer.foursquare.com/categorytree`

Primary Category	Number of sub-categories
Arts and Entertainments	17
College and University	23
Food	78
Great Outdoors	28
Home, Work, Other	15
Nightlife Spot	20
Shop	45
Travel Spot	14

Table 4.1: Location category hierarchy by Foursquare

4.3.1 User-based Collaborative Filtering

User-based CF first finds similar users based on their interests or ratings on items using a similarity measure. Then the recommendation score for an item is computed by the weighted combination of historical ratings on the item from similar users [43].

Given a user $u \in U$, the recommendation score that u will check-in a POI l that she has not visited yet is computed with the following equation,

$$R_u(l) = \frac{\sum\limits_{v \in U} w_{uv}}{|v|} \tag{4.1}$$

Here $v \in U$ are list of users who has visited the same location l and w_{uv} is the similarity score between u and v.

4.3.2 Preference-Aware Location Recommendation

Large-scale check-in data often faces data sparsity problem, as a user only visits a limited number of locations [44]. For example, in user location matrix of NYC Foursquare data, data sparsity is 99.46%. To overcome the data sparsity problem, authors proposed Preference-Aware location recommendation in [7]. In this chapter, we

leverage the temporal properties on LBSNs with baseline *Preference-Aware Location Recommendation* method.

Preference-Aware location recommendation method works in three major steps.

4.3.2.1 Step 1: Personal Preference Discovery

In this step, we learn each individual user's categorical preferences from his/her check-in history and predefined *Category Hierarchy*. Categorical preference of a user u is a numerical score, denoted as $CP_{u,c'}$. It represents u's affinity as well as willingness to visit a venue with category c'. As we have two level *Category Hierarchy*, we calculate user's preference on two levels (Primary categories and their sub-categories).

In [7], authors used *TF-IDF* approach to calculate user preference. *TF-IDF* (*Term Frequency-Inverse Document Frequency*) is widely used in text mining, which reflects how important a word is in a corpus of documents [45]. Motivated by this idea, we use the approach, where user's location history is regarded as a document and categories are considered as terms in the document. We denote this approach as *CF-ILF (Category Frequency-Inverse Location Frequency)*.

$CF(c', u.L)$ is the measure of how many times user u has visited the locations with a category c'. Intuitively, a user would visit more locations belonging to a category if he likes it. Here $u.L$ is the location set visited by u. ILF handles the *Rare-Item* problem [45]. Some locations are not visited by a user very often. For example, the number of visits to a restaurant is generally more than that of a museum. If a user visits location of a category that is rarely visited by other users, it means that the user could like this category more prominently [7].

CF is calculated using eq. (4.2) and ILF is calculated using eq. (4.3).

$$CF(c', u.L) = \frac{|\{u.l_i : l_i.c = c'\}|}{|u.L|} \tag{4.2}$$

$$ILF(c', L) = \log \frac{|U|}{|\{u_j.l \in L : l_j.c = c'\}|} \tag{4.3}$$

Here, $|\{u.l_i : l_i.c = c'\}|$ is user u's number of visits in category c', $|u.L|$ is the total number of user's visit in all locations. $|U|$ is the number of total users in the system. $|u_j.l \in L : l_j.c = c'|$ is the number of users who visit category c' among all users in U.

$CP_{u,c'}$ is generated using following equation:

$$CP_{u,c'} = CF(c', u.L) \times ILF(c', L) \tag{4.4}$$

Then, we generate User-Preference matrix. We generate the first matrix $A \in \mathbb{R}^{\mathrm{N}\times|\mathrm{CT}|}$ based on primary category, with A_{ij} be the $CP(u_i, pct_j)$, preference of user u_i on primary category pct_j. We generate the second matrix based $B \in \mathbb{R}^{\mathrm{N}\times|\mathrm{SCT}|}$ based on sub-categories, with B_{ij} be the $CP(u_i, sct_j)$ preference of user u_i on sub-category sct_j. Here, N is the number of users in the system. Here, $|CT| = 8$ and $|SCT| = 240$.

4.3.2.2 Step 2: User Similarity

We use *Cosine Similarity* [46] to find the similarity (w_{uv}) between two users u and v based on their categorical preferences. Let $\vec{p_u}$ be the preference vector of u over primary category and $p_{u,ct}$ be the element of p_u. User similarity between u and v based on primary categorical preference is calculated as:

$$w_{uv}^{(pc)} = \frac{\sum_{ct=1}^{|CT|} p_{u,ct} p_{v,ct}}{\sqrt{\sum_{ct=1}^{|CT|} p_{u,ct}^2} \sqrt{\sum_{ct=1}^{|CT|} p_{v,ct}^2}} \tag{4.5}$$

Let $\vec{s_u}$ be the preference vector of u over sub-category and $s_{u,sct}$ be the element of s_u. User similarity between u and v based on sub-categorical preference is calculated as:

$$w_{uv}^{(sc)} = \frac{\sum_{sct=1}^{|SCT|} s_{u,sct} s_{v,sct}}{\sqrt{\sum_{sct=1}^{|SCT|} s_{u,sct}^2} \sqrt{\sum_{sct=1}^{|SCT|} s_{v,sct}^2}} \tag{4.6}$$

User similarity between u and v is calculated as:

$$w_{uv} = \frac{1}{2} * \left\{ w_{uv}^{(pc)} + w_{uv}^{(sc)} \right\} \tag{4.7}$$

4.3.2.3 Step 3: Preference-Aware Recommendation

Given a user u, the recommendation score that u will visit location l that he has not visited yet is computed with the following equation:

$$R_u(l) = \frac{\sum_{v \in U} w_{uv}}{|v|} \times p_{u,pc_l} \times s_{u,sc_l} \tag{4.8}$$

Here v is list of users who also visited l. pc_l is primary category of location l and p_{u,pc_l} is the preference score of u and pc_l. sc_l is sub-category of location l and s_{u,sc_l} is the preference score of u and sc_l.

4.4 Enhancement over Baseline By Incorporating Temporal Influence

Human movement is significantly influenced by time [47]. For example, people tend to arrive at office at morning, check-in at restaurant for lunch at noon. Again

movement pattern on weekend is usually different than that of weekdays. For example, people generally go to a travel spot on weekend, whereas people go to school or office on weekday. It is obvious that personal preference of a user is influenced by time dimension. So a recommendation model should consider time dimension for generating efficient recommendations.

To incorporate temporal influence, we introduce the time dimension to generate time-specific *User-Preference* matrix. We split a day into multiple equal time intervals (t_s) based on hour. Then we generate temporal preference of individual user on each time segment (t_s).

4.4.1 Temporal Categorical Preference

Given a user u, time segment t_s, category c', temporal preference of user u on category c', denoted as $CP^{(t_s)}_{u,c'}$ is calculated using following equation:

$$CP^{(t_s)}_{u,c'} = CF^{(t_s)}(c', u.L^{(t_s)}) \times ILF(c', L^{(t_s)}) \tag{4.9}$$

Here $CF^{(t_s)}(c', u.L^{(t_s)})$ is the *Category Frequency* of user u for category c' at time segment t_s. $u.L^{(t_s)}$ is the location set visited by u at t_s. $ILF(c', L^{(t_s)})$ is the *Inverse Location Frequency* for category c'. $L^{(t_s)}$ is the list of all locations that has been visited at t_s by all users.

$$CF^{(t_s)}(c', u.L^{(t_s)}) = \frac{|\{u.l_i^{(t_s)} : l_i^{(t_s)}.c = c'\}|}{|u.L^{(t_s)}|} \tag{4.10}$$

$$ILF(c', L^{(t_s)}) = \log \frac{|U^{(t_s)}|}{|\{u_j.l \in L^{(t_s)} : l_j.c = c'\}|} \tag{4.11}$$

Here, $|\{u.l_i^{(t_s)} : l_i^{(t_s)}.c = c'\}|$ is the number of visits by user u at category c' at time segment t_s. $|u.L^{(t_s)}|$ is total visits by user u at time t_s. $|U^{(t_s)}|$ is the total number

of unique users in the system that has checked-in at time t_s. $|u_j.l \in L^{(t_s)} : l_j.c = c'|$ is the total number of unique users that visit at category c' at time t_s.

For each time segment, we generate two User-Categorical Preference Matrix. One is based on primary category $A^{(t_s)} \in \mathbb{R}^{\mathrm{N}\times|\mathrm{CT}|}$ and the second one is based on sub-categories $B^{(t_s)} \in \mathbb{R}^{\mathrm{N}\times|\mathrm{SCT}|}$. Here, N is the number of users in the system.

User similarity between two users is calculated based on the temporal categorical preference. If two users prefer to check in a POI with the same category during the same time, similarity between them will be high.

4.4.2 Temporal Popularity

Popularity of a location plays a significant role to attract users. People tend to visit a more popular POI for better satisfaction. However, popularity also varies over time. For example, a bar is more popular at night, whereas people tend to visit a museum during morning or afternoon. For better recommendation, we intend to calculate popularity score of each POI on each time segment. Popularity of a POI l at time t_s is calculated using following equation:

$$P^{(ts)}(l) = \frac{1}{2} * \left\{ \frac{|U^{(t_s)}(l)|}{|U(l)|} + \frac{|Chk^{(t_s)}(l)|}{|Chk(l)|} \right\} \tag{4.12}$$

Here $|U^{(t_s)}(l)|$ is the number of users that visited l at time t_s, $|U(l)|$ is the total number of users visited l. $|Chk^{(t_s)}(l)|$ is the number of check-ins at l at time t_s and $|Chk(l)|$ is the total number of check-ins at location l.

4.5 Incorporating Spatial Influence by POI Clustering

Geographical position of a POI plays a significant role to attract users [40][6]. People tend to visit nearby places. The propensity of a user to choose a POI de-

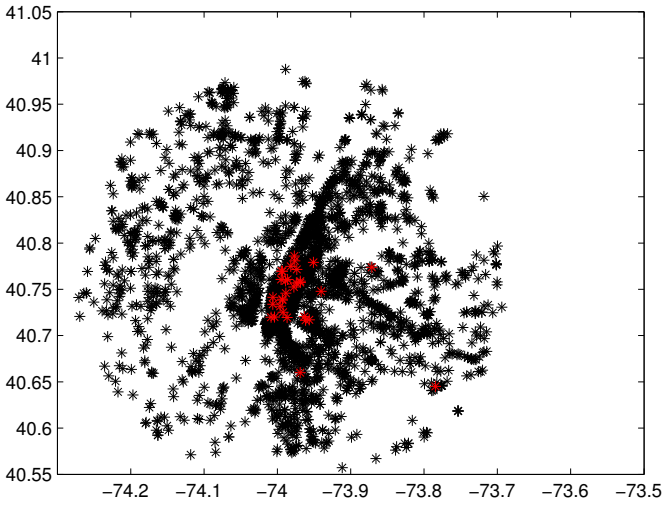

Figure 4.1: User check-in distribution in NY City

creases as the distance between user and POI increases [4]. Consider the example in Figure 4.1. Black points represent all the POI locations of NY City. Red points are the check-in distribution of a single user. It is obvious that, one person does not move all over the city, rather his movement data is limited to some geographical regions.

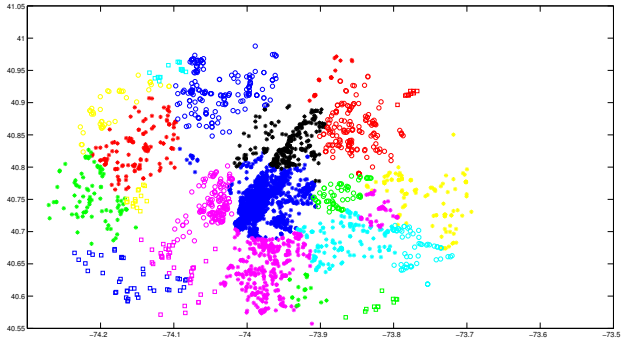

Figure 4.2: Regions of POI locations

4.5.1 Spatial-Aware Candidate Selection

To incorporate spatial influence, we cluster all the POI locations into M number of regions. we use a modified version of DBSCAN [48] algorithm for clustering. DBSCAN is a density based clustering algorithm. It requires two parameters: ϵ and $MinPts$. Density of a point is defined as the total number of neighbours within a given radius (ϵ) of the point. A data point is considered dense if the number of its neighbours is greater than $MinPts$.

The drawback of this algorithm is that, it is too sensitive to parameter ϵ. If ϵ is small, DBSCAN generates small sized clusters with a lot of outliers. If ϵ is big, a large number of points may merge together to form a big cluster. To overcome this problem, we used a modified version of DBSCAN algorithm. We introduced one extra parameter $maxD$. $maxD$ defines the possible maximum diameter of a cluster. Figure 4.2 shows the result of this algorithm on POI locations of NY City (see Figure 5.8). Algorithm generates 44 regions.

Let $G = \{g_1, g_2, g_3, ...g_m\}$ be the list of all regions. Each region g_i is a collection of POI locations. Let $G^{(u)} = \{g_1, g_2, ...g_k\}$ be the list of regions in which user u have visited. For each user, we project his check-in locations to G to generate $G^{(u)}$. All POI locations of $G^{(u)}$ are selected as a candidate POI locations for recommendation of u.

4.5.2 Regional Popularity

In this section, we intend to calculate popularity score of a location at the regional level. Two locations with the same terms can be rated differently in different

region [4]. We calculate regional popularity of POI location l, denoted as $P^{(g)}(l)$ using following equation:

$$P^{(g)}(l) = \frac{1}{2} * \left\{ \frac{|U_l|}{max_{l \in g}\{U_l\}} + \frac{|Chk_l|}{max_{l \in g}\{Chk_l\}} \right\} \tag{4.13}$$

Here, U_l is the number of people visited location l, $max_{l \in g}\{U_l\}$ is the maximum number of people visited in a location in region g. Chk_l is the number of total check-ins in location l and $max_{l \in g}\{Chk_l\}$ is the maximum number of check-ins in a location in region g.

4.6 POI Recommendation

Given a user u and check-in history of u, we first generate spatial-aware candidate location list $G^{(u)}$. Given time segment t_s, we calculate recommendation score $R_u^{(t_s)}(l)$ for each candidate location $l \in G^{(u)}$ using the following equation:

$$R_u^{(t_s)}(l) = \frac{\sum_{v \in U} w_{uv}^{(t_s)}}{|v|} \times p_{u,pc_l}^{(t_s)} \times s_{u,sc_l}^{(t_s)} \times P^{(ts)}(l) \times P^{(g)}(l) \tag{4.14}$$

Here v is the list of users who also visited the location l at time t_s.

4.7 Experiments

4.7.1 Dataset

We use the real-world check-in dataset from Foursquare[5]. Dataset includes $227,428$ check-in data from New York City, USA. The dataset has data from 12 April 2012 to 16 February 2013 (10 months). Each check-in Ch_{ij} contains user (u_i), location id (l_j) and time (t). Each location id l_j is associated with geographical position (lat,lon), primary category (pc_{l_j}) and sub-category (sc_{l_j}). It contains check-in data of $1,083$ users and $38,383$ locations. To get more effective results, we removed POIs

[5] `www.foursquare.com`

that have lower than 5 check-ins. After preprocessing, the dataset contains $4,597$ locations and $164,307$ check-ins. For each user, we randomly mark off 50% of his location histories as a training set to learn his temporal categorical preferences and location preferences. The other 50% is used as a test set.

4.7.2 Evaluation Method

To evaluate our proposed method, we use two well-established metrics: *precision* and *recall* [49]. We denote them as *Pre@N* and *rec@N* respectively.

$$pre@N = \frac{number\ of\ recovered\ ground\ truths}{total\ number\ of\ recommendations\ (N)} \tag{4.15}$$

$$rec@N = \frac{number\ of\ recovered\ ground\ truths}{total\ number\ of\ ground\ truths} \tag{4.16}$$

Here, N is the number of recommendation results. We use 3 values of N in our experiments: 5,10 and 20. Ground truth refers to the set of locations where user has visited. So, *pre@N* measures how many POIs in the top-N recommended POIs correspond to the ground truth POIs. *rec@N* measures how many POIs in the ground truths were returned as top-N recommendation. These two measures can be used together to evaluate the result, which is known as *f-measure*.

$$f\text{-}measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.17}$$

Given time segment t_s, precision and recall for t_s are denoted as *precision(t_s)* and *recall(t_s)* respectively. The overall precision and recall are calculated by averaging the value over all time slots. Here, T is the number of time slots.

$$precision = \frac{1}{T} \sum_{t_s \in T} precision(t_s) \tag{4.18}$$

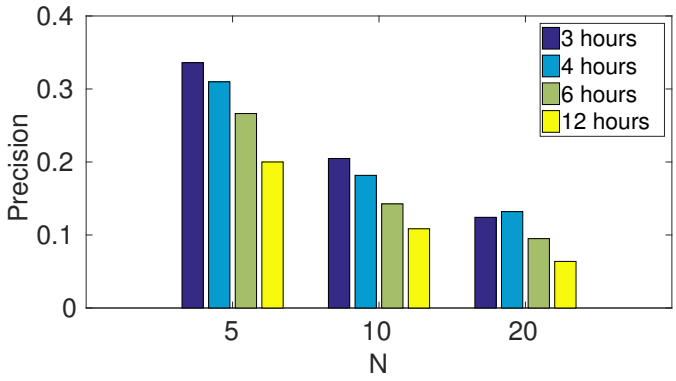

Figure 4.3: *Effects of time segment length, precision@N*

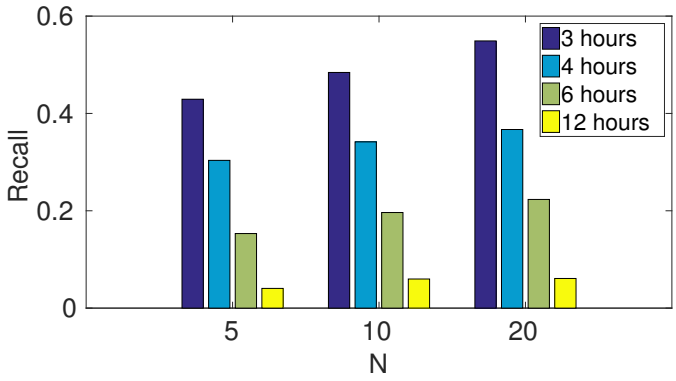

Figure 4.4: *Effects of time segment length, recall@N*

$$recall = \frac{1}{T} \sum_{t_s \in T} recall(t_s) \tag{4.19}$$

4.7.3 Experimental Results

In this experiment, we use different time slot length to study how the experimental results change on varying time slot length. The length of time slot controls the granularity of time-aware recommendations. A larger value of time slot length means that the result will be less time-specific. Figure 4.3, 4.4, 4.5 shows the *Precision@N*, *Recall@N* and *f-measure@N* with varying time slot length. We use three values of N (*i.e.*, 5, 10, 20) in our experiments.

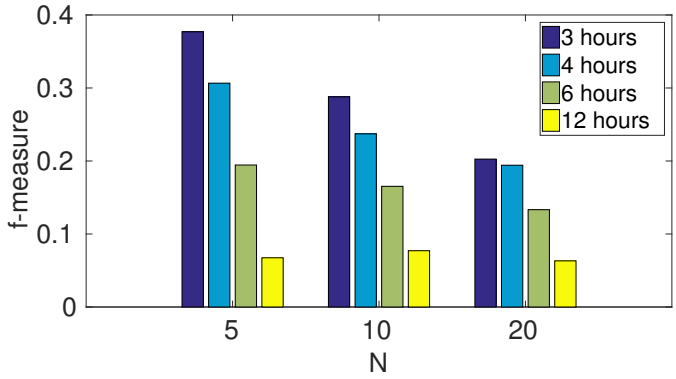

Figure 4.5: *Effects of time segment length, f-measure@N*

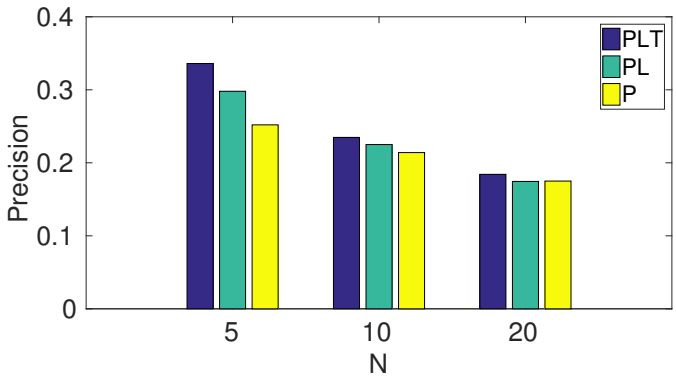

Figure 4.6: *Comparison with baseline methods, precision@N*

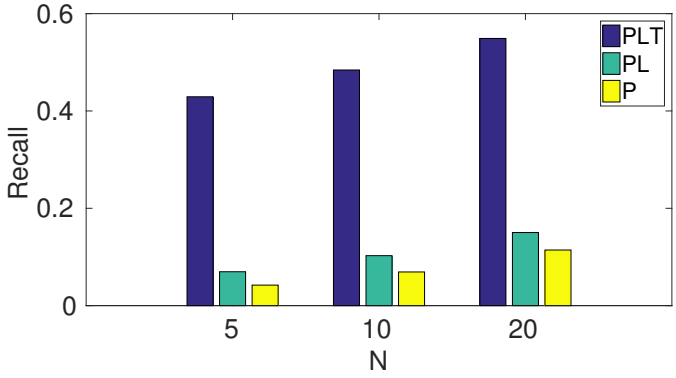

Figure 4.7: *Comparison with baseline methods, recall@N*

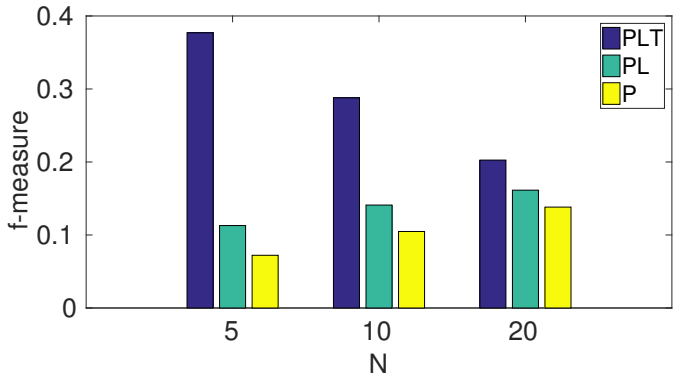

Figure 4.8: *Comparison with baseline methods, f-measure@N*

We can see from results that, as time slot length increases, precision value decreases slightly in most of the cases (see Figure 4.3). But recall value drops dramatically with the higher time slot length (see Figure 4.4). The reason is, with the lower value of time slot length, the recommendation method generates more focused and correct time-specific results. As, precision depends on N, precision gets slightly better. Because in most of the cases the number of time-specific ground truth value is less than N. But recall value gets a lot better as the length of time slot decreases. Figure 4.5 shows the *f-measure@N* value that combines *precision* and *recall* value. We can see that, for all length of N, *f-measure* value is better with lower time slot length.

We evaluate the effectiveness of our *preference-aware, location-aware and time-aware (PLT)* method by comparing with two other methods: 1) *Preference-aware (P)* method, and 2) *Preference-aware, location-aware (PL). Preference-aware (P)* method is the base-line *preference-aware* recommendation method that has been described in Section (4). *Preference-aware, location-aware (PL)* method combines the method discussed in Section (4) and Section (6).

Figure 4.6, 4.7 and 4.8 shows the *precision*, *recall* and *f-measure* value of three methods with $N = 5$, 10 and 20. We can see from results that, incorporating spatial influence gives us better results than baseline method (P) for all values of N. But the results of incorporating temporal influence with spatial influence (PLT) outperforms both of them (PL and P).

4.8 Conclusion

This chapter presents a time-aware, location-aware and preference-aware recommendation system, which provides a user time-specific location recommendation based on user's personal categorical preferences and spatial preferences. This method also considers a combination of regional popularity and temporal popularity of a particular POI. To the best of our knowledge, this is the first work that combines all the 4 factors (temporal, spatial, categorical preferences, popularity) together to generate recommendations. Experimental results show that our method combining multiple factors is better than other baseline approaches. In the future, we plan to incorporate other time dimensions (day of the week, month/season of the year) in POI recommendation.

CHAPTER 5

Preference-Aware Successive POI Recommendation With Spatial and Temporal Influence

5.1 Introduction

In recent years, location based social network (LBSN) services have gained a vast amount of attention and popularity among users. Foursquare [1], Yelp [50] and Facebook Places[2] are a few of the examples of LBSN services. LBSNs allow users to share their life experiences via mobile devices. "Check-in" is a process by which users post their arrival to a location. They also share their experiences by leaving comments or tips on that location. A *Point of Interest* (POI) location can be a "Restaurant", "Travel spot", "Park" and so on.

It was reported that there are over 30 million registered users in Foursquare. The number of check-ins posted by them by January 2013 was over 3 billion [51]. The "check-ins" contain abundant information about their daily activities as well as their preferences among the POIs. For example, people who often visit a gym must be interested in physical exercise. Also, people who visit the same place may share similar interests. Location histories and opinions of one user can be exploited to recommend an unvisited location to another user if they share a similar interest.

The task of *POI recommendation* is to provide personalized recommendation of POI locations to mobile users. The recommended locations should match their personal interests within a geospatial range [39]. Recently, POI recommendation in

[1]https://foursquare.com

[2]https://www.facebook.com/places/

LBSNs has attracted much attention in both research and industry [40], [52]. However traditional POI recommendation systems consider all check-ins as a whole and generate recommendations [4, 5, 6, 7, 8]. They do not consider the users' sequential movement information. Therefore, they cannot suggest where a user may go in the next few hours based on their current location or status.

In this work, we consider the task of personalized successive POI recommendation. Successive POI recommendation refers to the problem of recommending users the very next location based on his current location and current time. This task recommends those locations that a user may not visit frequently or before, but he/she may like to visit at successive timestamps [9]. For example, successive POI recommendation can suggest a user location to have fun after dinner, or a location for outdoor activities in a nearby park after his work.

The essential difference between traditional recommendation system and successive POI recommendation system is that the performance of successive recommendation tasks is largely influenced by users' current visiting locations [53]. Also the shift from one location to another location depends on their categorical preferences and periodic patterns. One may go to a coffee shop to grab a cup of coffee first, then head to work or university. On a weekend people often go to shopping, then go to a restaurant for dinner or lunch.

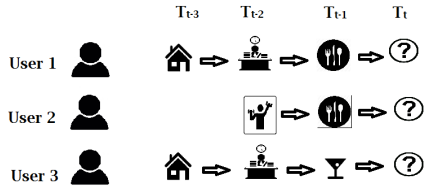

Figure 5.1: Sequential check-in data of three users

Figure 5.1 gives examples of sequential check-in data of three users. User 1 goes to dinner from the office. User 3 goes to a bar after office. If user 1 and user 3 share similar interests, user 1 also may become interested to go to a bar after office. Thus collaborative information shared by the users can be used to recommend them the possible next locations based on their current location.

In [54], we proposed a preference-aware, location-aware and time-aware POI recommendation system. The method used *User-based Collaborative Filtering* method for POI recommendation while incorporating four other factors: 1) Categorical preferences, 2) Temporal influence, 3) Geographical preferences and 4) Popularity of POIs. In this chapter, we extend this work and propose a preference-aware successive POI recommendation system with incorporating spatial and temporal influence (PLTSRS) that offers a particular user a set of POI locations based on his current location, current time and his personal interests. The contribution of this chapter can be summarized as follows:

- We model personal preferences of users based on the category information of their location histories. We further analyse the temporal influence on their activities. We incorporate time dimension to model time-specific user preferences.
- We mine sequential patterns from check-in location of each user. Then, we construct personalized *Category-To-Category* transition probability matrix using first order markov chain [55].
- We analyze users' spatial behavior and incorporate spatial influence to generate spatial-aware location recommendations.
- Our recommendation model uses popularity factor of individual location by calculating time-specific popularity.
- We develope a successive POI recommendation model **PLTSRS** (**P**reference-Aware, **L**ocation-Aware and **T**ime-Aware **S**uccessive POI **R**ecommendation **S**ystem),

which jointly considers user's personalized sequential movement information, temporal categorical preferences, location preferences and popularity of POIs. To best of our knowledge, this is the first work that uses all the factors together to build a successive POI recommendation model.

- We evaluate our proposed framework with one large scale LBSN dataset from foursquare[3].

The rest of the chapter is organized as follows. In Section 5.2, we review some related works. In Section 5.3, we discuss the data structure and data analysis results. Section 5.4 presents our proposed method in details. We present our experimental results in Section 5.5. Section 5.6 concludes the chapter.

5.2 Related Work

With the easy availability of users' check-in data in LBSN, many studies have been conducted for POI recommendation. In this section, we briefly introduce two lines of research related to our task: 1) Traditional POI recommendation, 2) Successive POI recommendation.

Traditional POI recommendation systems have been extensively studied in the last several years. Two popular approaches have been used to generate recommendation model: Collaborative Filtering algorithm and Non-Negative Matrix Factorization algorithm.

In [40], the *User-based CF* approach considers a combination of social influence and spatial influence. Their experiments report that geographical influence has a significant impact on the accuracy of POI recommendation, whereas the social friend link contributes little. Their results also indicate that *user-based CF* works much better than *Item-based CF*. In [6], the authors exploit spatial influence as well as

[3]https://foursquare.com

temporal influence for building a recommendation model. They incorporate time factors in the basic *CF based* model by computing similarity between two users by considering check-in information at a specific time t, rather than that of all times. In [7], the authors explore user preferences with social and geographical influence for POI recommendation. They model user preferences using predefined categorical information of location data.

In [4], the authors propose a geographical probabilistic factor analysis framework for recommendation that takes various other factors into consideration, viz. user-item preferences, POI popularity and geographical preferences of individual users. In [41], the authors propose a friendship based collaborative filtering (FCF) approach for POI recommendation.

In [54], the authors propose a *User Based Collaborative Filtering* method based framework which combines 4 factors: categorical preferences, temporal influence, spatial influence and popularity of a location. They incorporate time factors by generating time specific categorical preferences. Clustering method has been used to model location preference of each user. Popularity of each location has been calculated by combining both regional factor and temporal factor.

Lately a few successive POI recommendation works have been conducted. In [52], the authors propose a probabilistic model to integrate category transition probability and POI popularity to solve the problem. But they did not incorporate spatial influence here.

In [56], the authors propose a Factorized Personalized Markov Chain (FPMC) model for next-item recommendation. In [9], the authors propose *FPMC-LR* model by extending *FPMC* model with localized region constraint to solve successive POI recommendation task. They divide the geographical space into a grid. Locations of the grid cell the user is currently visiting and its surrounding 8 grid cells are used

as candidate locations. This condition is called *Localized Region Constraint*. In [57], the authors propose a personalized metric embedding method (*PRME*) to model personalized check-in sequences for next new POI recommendation.

5.3 Preliminaries

5.3.1 Data Structure

In this chapter, we use one real-world LBSN dataset from *Foursquare*. This dataset has three key data structure: 1) User, 2) POI location and 3) Check-in.

1) Each user u is represented by a unique id. Let $U = \{u_1, u_2, u_3, ...u_n\}$ be the set of users.

2) Each POI location has a unique POI id and geographical position (latitude and longitude). Let $L = \{l_1, l_2, l_3, ..., l_m\}$ be the set of POI locations. Each location l is also associated with category information, which represents its functionality. In Foursquare, there are 8 primary categories ("Food", "Arts and Crafts" etc.). Each primary category includes other sub-categories. In this chapter, we only consider the sub-category information of a location for simplicity. The word category and sub-category will be used interchangeably throughout the chapter.

3) "*Check-in*" is a process by which a user u announces his physical arrival or presence at a venue in location based social network. Let $Ch_{ij} = \{u_i, l_j, t\}$ be a *check-in* tuple, which represents that user u_i checked in POI l_j at time t.

5.3.2 Data Analysis

In this section, we present some data analysis results to see how different factors (Spatial, Temporal, Preference) influence a user to choose a location to visit.

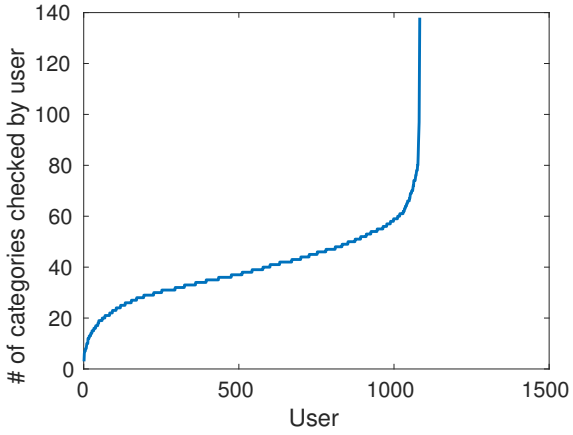

Figure 5.2: Number of unique categories checked-in by users

5.3.2.1 Categorical Preference Constraints

Personal preference plays an important rule for a user to choose a POI. They prefer to visit a location only if the category of that location matches their interests. To have a better idea, we count the number of unique categories visited by users. We sort the users based on the count and plot the result (see Figure 5.2). We have a total of 252 categories. We see that the number of categories visited by most of the users is less than 60. Users generally do not visit locations of all categories, they visit a location only if they like the category. So a good POI recommendation system must recommend a location to a user that matches with his preferences.

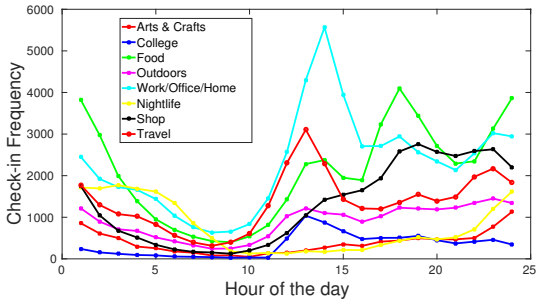

Figure 5.3: Check-in frequency at different hour of the day

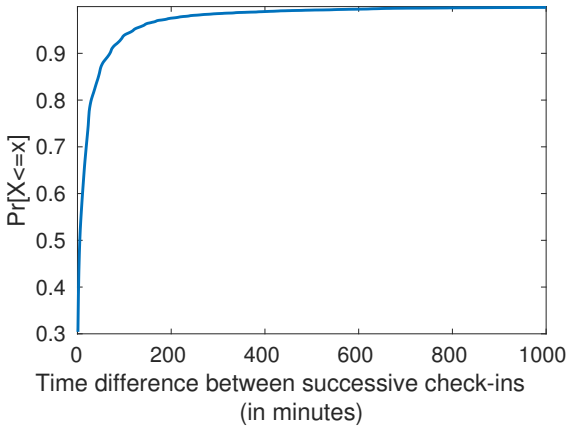

Figure 5.4: Time difference between two successive check-ins (in minutes)

5.3.2.2 Temporal Influence

User activities are significantly influenced by time [6]. We count the check-in frequency of 8 primary categories at different hours of the day (see Figure 5.3). Result shows that category "Shop" is more active from 3 pm to until 12 am. On the other hand, category "Nightlife" starts after 10 pm and continues until 5 am.

We have done analysis to see how frequently people visit locations. We plot the *Cumulative Distribution Function (CDF)* of the time differences between successive check-in data (see Figure 5.4). Result shows that 90% of successive check-ins have a time difference less than 200 minutes.

5.3.2.3 Spatial Influence

Geographical position of a POI location plays an important role. Figure 5.5 shows the *Cumulative Distribution Function (CDF)* of geographical distance between two successive check-ins. Result shows that about 90% of successive check-ins have a geographical distance less than 20 km.

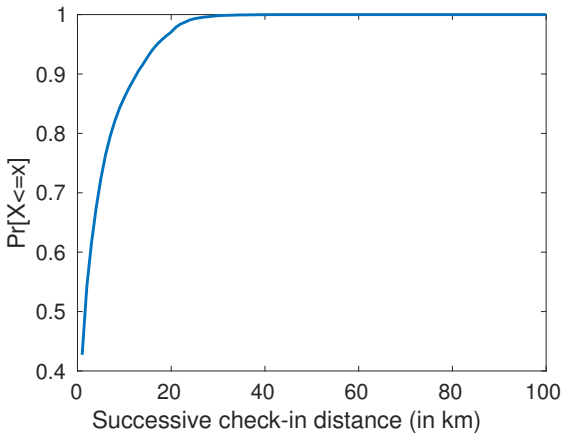

Figure 5.5: Geographical distance between two successive check-ins

5.3.3 Problem Formulation

Let U be the set of users and P be the set of locations. L_u denotes the check-in histories of user u. Given a user $u(u \in U)$, his check-in histories L_u, his currently visiting POI $l_{now}(l_{now} \in L)$ and corresponding visiting timestamp t_{now}, the task is to recommend a new POI $l_{next}(l_{next} \notin L_u)$ to u to visit within time range t_{now} to t_{next}. Here, (t_{next} - t_{now}) $\leq T_{max}$. Here, T_{max} is a user defined time interval parameter.

5.3.4 User-based Collaborative Filtering

User-based CF first finds similar users based on their interests/ratings on items using a similarity measure. Then the recommendation score for an item is computed by the weighted combination of historical ratings on the item from similar users [43].

Given a user $u \in U$, the recommendation score that u will check-in a POI l that she has not visited yet is computed with the following equation,

$$R_u(l) = \frac{\sum\limits_{v \in U} w_{uv}}{|v|} \tag{5.1}$$

Here $v \in U$ are list of users who have visited the same location l and w_{uv} is the similarity score between u and v.

5.4 PLTSRS Framework

Our proposed framework is comprised of two major steps. 1*) Offline Modeling* and 2*) Online Recommendation*.

The *Offline Modeling* step has 3 components. 1) *Learning User's Categorical Transition Probabilities*, 2) *Time-specific Personal Preference Discovery* and 3) *Calculating Time-Specific Popularity of Locations*.

In the first component, we learn each user's categorical transition probability denoted by $T_u(c_i, c_j)$. $T_u(c_i, c_j)$ is calculated using first order markov chain that indicates the probability of user u to move from a location of category c_i to location with category c_j. In the second component, we learn each user's personal categorical preference on category c denoted as $P_u(c)$. As preference depends on time, we learn time-specific categorical preference on category c at time segment t_s denoted as $P_u^{(t_s)}(c)$. In the third component, we calculate the time-specific popularity of each POI location l denoted as $\rho^{(t_s)}(l)$.

The *Online Recommendation* has two components. 1) *Spatial-Aware Candidate Selection* and 2) *Successive Location Recommendation*. The first component selects a set of candidate locations based on u's current location l_{now}. This component improves the efficiency of the approach significantly as the number of candidate locations is much smaller than the total number of locations. Given a user u, his current location l_{now} and current time t_{now}, the second component calculates the location ratings of all candidate locations based on the factors mentioned above. The *top-K* locations are recommended to user u.

5.4.1 Offiline Modeling

5.4.1.1 Categorical Transition Probability

In this step, for each user u, we first extract the successive location pairs from his check-in sequences. A location pair (l_i, l_j) is a successive pair if the time difference between u's visit at location l_i and location l_j is less than the time interval threshold T_{max}. Then, we map the locations of the successive location pairs with corresponding category information to mine the successive category pairs (c_x, c_y). Here, c_x is the category of l_i and c_y is the category of l_j. We build a *Category-To-Category* transition probability matrix of user u denoted as T_u. The transition matrix $T_u \in [0,1]^{|\mathrm{C}|\times|\mathrm{C}|}$, $T_u(i,j)$ specifies the probability for a user u to move from location with category c_i to a location with category c_j. Transition probability $T_u(i,j)$ is calculated as:

$$T_u(i,j) = \frac{|\{(l_1^{(u)}, l_2^{(u)}) : l_1^{(u)} \in C_i \cap l_2^{(u)} \in C_j\}|}{|\{(l_1^{(u)}, l_2^{(u)} : l_1^{(u)} \in C_i\}|} \tag{5.2}$$

Figure 5.6 shows the transition matrices of individual users. Entries with "?" refers to missing values as there is no data to estimate probabilities. A single user generally does not visit all categories, so there may be a lot of missing values in his transition matrix. To solve this problem, we use low-rank non-negative matrix factorization [58] algorithm to factorize each transition probability matrix T_u into two low rank matrices $W_u \in \mathbb{R}^{\mathrm{k}\times|\mathrm{C}|}$ and $H_u \in \mathbb{R}^{|\mathrm{C}|\times\mathrm{k}}$, with $k << |C|$ being the number of latent factors. After obtaining W_u and H_u, the probability matrix T_u is approximated as $\bar{T}_u$, $\bar{T}_u(i,j)$ being the approximated probability of transition from category c_i to category c_j (see Figure 5.7).

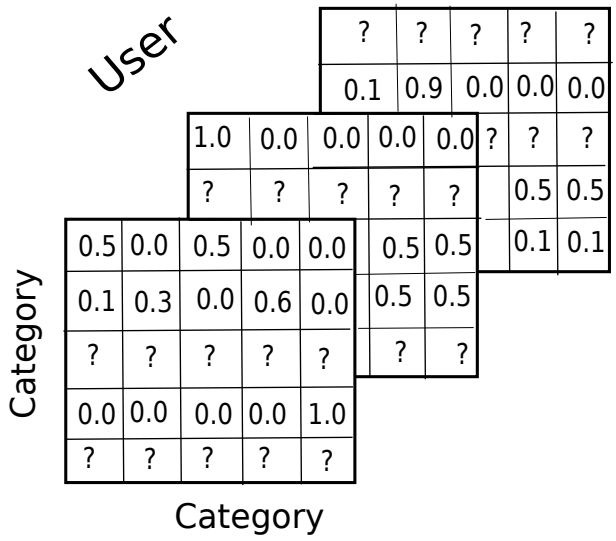

Figure 5.6: Personalized Transition Matrices of users

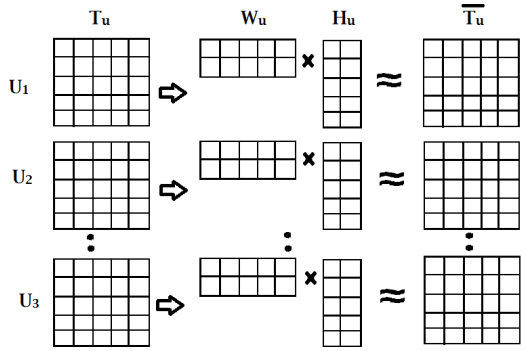

Figure 5.7: Factorized individual transition probability matrix

5.4.1.2 Personal Preference Discovery

In this step, we model each individual user's categorical preferences from his/her check-in history. Categorical preference of a user u denoted as $P_u(c)$ represents u's affinity to visit a location with category c. $P_u(c)$ is generated using following equation [54].

$$P_u(c) = CF(c, L_u) \times ILF(c, L) \tag{5.3}$$

Here $CF(c, L_u)$ is the measure of how many times user u has visited the locations with a category c. Intuitively, a user would visit more locations belonging to a category if he likes it. Here L_u is the location set visited by u. ILF handles the *Rare-Item* problem [45]. Some locations are not visited by a user very often. For example, the number of visits to a restaurant is generally more than that of a museum. If a user visits location of a category that is rarely visited by other users, it means that the user could like this category more prominently [7].

CF is calculated using eq. (5.4) and ILF is calculated using eq. (5.5).

$$CF(c, L_u) = \frac{|\{u.l_i : l_i \in c\}|}{|L_u|} \tag{5.4}$$

$$ILF(c, L) = \log \frac{|U|}{|\{u_j.l \in L : l_j.c \in u_j.C\}|} \tag{5.5}$$

Here, $|\{u.l_i : l_i \in c\}|$ is user u's number of visits in category c, $|L_u|$ is the total number of user's visit in all locations. $|U|$ is the number of total users in the system. $|\{u_j.l \in L : l_j.c \in u_j.C\}|$ is the number of users who visit category c among all users in U. User similarity between two users is calculated based on their categorical preferences. We use *Cosine Similarity* [46] to find the similarity between two users u and v denoted as w_{uv}.

5.4.1.3 Temporal Categorical Preference

As categorical preference may vary over time, we intend to find time-specific categorical preferences of each user. We divide the whole day into equal length of time segment (t_s). In this chapter, we use time slot length = 1 hour. So the whole day is divided into 24 time segments. Given a user u, time segment t_s, category c,

temporal preference of user u on category c, denoted as $P_u^{(t_s)}(c)$ is calculated using following equation [54].

$$P_u^{(t_s)}(c) = CF^{(t_s)}(c, L_u^{(t_s)}) \times ILF(c, L^{(t_s)}) \tag{5.6}$$

Here $CF^{(t_s)}(c, L_u^{(t_s)})$ is the *Category Frequency* of user u and category c at time segment t_s. $L_u^{(t_s)}$ is the location set visited by u at t_s. $ILF(c, L^{(t_s)})$ is the *Inverse Location Frequency* for category c. $L^{(t_s)}$ is the list of all locations that has been visited at t_s. $CF^{(t_s)}(c, L_u^{(t_s)})$ and $ILF(c, L^{(t_s)})$ are calculated using following equations.

$$CF^{(t_s)}(c, L_u^{(t_s)}) = \frac{|\{u.l_i^{(t_s)} : l_i^{(t_s)} \in c\}|}{|L_u^{(t_s)}|} \tag{5.7}$$

$$ILF(c, L^{(t_s)})) = \log \frac{|U^{(t_s)}|}{|\{u_j.l \in L : l_j.c \in u_j.C\}|} \tag{5.8}$$

Here, $|\{u.l_i^{(t_s)} : l_i^{(t_s)} \in c\}|$ is the number of visits by user u at category c at time segment t_s. $|L_u^{(t_s)}|$ is total visits by user u at time t_s. $|U^{(t_s)}|$ is the total number of unique users in the system that has checked-in at time t_s. $|\{u_j.l \in L : l_j.c \in u_j.C\}|$ is the total number of unique users that visit at category c at time t_s.

5.4.1.4 Temporal Popularity Of a Location

Popularity of a location plays a significant role to attract user. People tend to visit a more popular POI for better satisfaction. However, popularity also varies over time. For example, a bar is more popular at night, whereas people tend to visit a museum during morning or afternoon. For better recommendation, we intend to calculate popularity score of each POI on each time segment. Popularity of a POI l at time t_s is calculated using following equation [54].

$$\rho^{(ts)}(l) = \frac{1}{2} * \left\{ \frac{|U^{(t_s)}(l)|}{|U(l)|} + \frac{|Chk^{(t_s)}(l)|}{Chk(l)} \right\} \tag{5.9}$$

Here $|U^{(t_s)}(l)|$ is the number of users that visited l at time t_s, $|U(l)|$ is the total number of users visited l. $|Chk^{(t_s)}(l)|$ is the number of check-ins at l at time t_s and $Chk(l)$ is total number of check-ins at location l.

5.4.2 Online Recommendation

5.4.2.1 Spatial-Aware Candidate Selection

Geographical position of a POI plays a significant role to attract users [40, 6]. People tend to visit nearby places. The propensity of a user to choose a POI decreases as the distance between the user and the POI increases [4]. Consider the example in Figure 5.8. Black points represent all the POI locations of NY City. Red points are the check-in distribution of a single user. It is obvious that, this person does not move all over the city, rather his movement data is limited to some geographical regions. We have borrowed this example from [54]. Also Figure 5.5 indicates that 90% of successive check-ins have distance less than 20 km.

To incorporate spatial influence, we divide the whole problem space into square grids whose side length is d km. Locations of the grid cell the user is currently visiting and it's surrounding 8 adjacent grid cells are used as the candidate grid cells. The locations of the candidate grid cells are used as the candidate locations for recommendation (see Figure 5.9). Distance between two points are calculated using *Haversine Formula* [59].

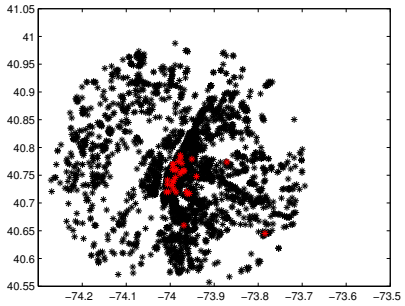

Figure 5.8: All POI locations in NY city

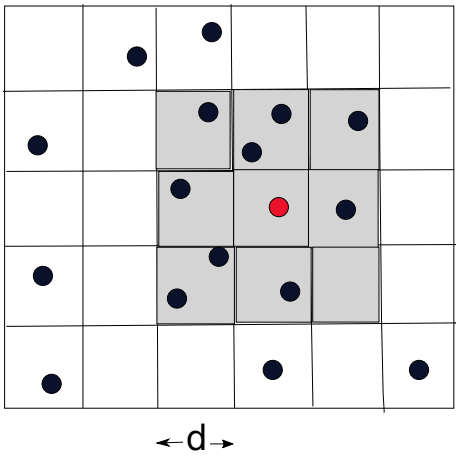

Figure 5.9: Spatial-Aware candidate selection

5.4.2.2 Successive Location Recommendation

Given a user u, his current location l_{now} with category c_{now}, we first generate spatial-aware candidate location list $S^{(u)}(L)$. Let the current time be t_{now}. In this section, we present the method to rank the candidate locations. T_{max} is a user-defined time interval parameter. Top-K locations are recommended to user u that he may want to visit within time range t_{now} to t_{next}, where $(t_{next} - t_{now}) \leq T_{max}$.

Let, the recommended location be l_{next}. Category of l_{next} is c_{next}. In Offline method, we have calculated time-specific categorical preference of user u at category

c denoted as $P_u^{(t_s)}(c)$. Given the time interval T_{max}, we find the preference of u for c_{next} from time range t_{now} to t_{next} defined as $P_u^{(t_{now},t_{next})}(c_{next})$.

$$P_u^{(t_{now},t_{next})}(c_{next}) = max\{P_u^{(t_s)}(c_{next})\} \tag{5.10}$$

Where $t_s \geq t_{now}$ and $t_s \leq t_{next}$. For example, let t_{now} = 10 am, T_{max} = 6 hours. so, t_{next} = 4 pm. So u's preference for c_{next} from time range 10 am to 4 pm is calculated as $max\{P_u^{(t_{10})}(c_{next}), P_u^{(t_{11})}(c_{next}), ...P_u^{(t_{16})}(c_{next})\}$

We find the popularity of location l_{next} at time range t_{now} to t_{next} denoted as $\rho^{(t_{now},t_{next})}(l_{next})$

$$\rho^{(t_{now},t_{next})}(l_{next}) = max\{\rho^{(t_s)}(l_{next})\} \tag{5.11}$$

Where $t_s \geq t_{now}$ and $t_s \leq t_{next}$.

The rating of location l for user u, denoted as $R_u^{t_{next}}(l_{next})$ is calculated as:

$$R_u^{t_{next}}(l_{next}) = \frac{\sum_{v \in U} w_{uv}}{|v|} * P_u^{(t_{now},t_{next})}(c_{next}) * \bar{T}_u(c_{now}, c_{next}) * \rho^{(t_{now},t_{next})}(l_{next}) \tag{5.12}$$

Here, $v \in U$ are the list of users who also visited the same location l_{next} at specified time range t_{now} to t_{next}.

5.5 Experiments

5.5.1 Dataset

We use the real-world check-in dataset from Foursquare. The dataset includes 227,428 check-in data from New York City, USA. The dataset has data from 12 April 2012 to 16 February 2013 (10 months). We obtain this dataset from [60]. Each check-in Ch_{ij} contains user (u_i), location id (l_j) and time (t). Each location id l_j is

associated with geographical position (latitude, longitude) and category c. It contains check-in data of $1,083$ users and $38,383$ locations. To get more effective results, we removed POIs that have lower than 10 check-ins. After preprocessing, the dataset contains $4,597$ locations and $164,307$ check-ins.

For experiment, we use the data of first 8 months as training set. The training data is used to learn the users' temporal categorical preferences and categorical transition probability and popularity of POIs. The rest of the data is used as a test set.

5.5.1.1 Evaluation Method

To evaluate our proposed method, we use two well-established metrics: *precision* and *recall* [49]. *Precision* and *recall* are calculated using the following equations.

$$pre@N = \frac{number\ of\ relevant\ recommendations}{N} \tag{5.13}$$

$$re@N = \frac{number\ of\ relevant\ recommendations}{total\ number\ of\ ground\ truths} \tag{5.14}$$

Here, N is the number of recommendation results. Ground truth refers to the set of locations where user has truly visited within the specified time range. So, *Pre@N* measures how many POIs in the top-N recommended POIs correspond to the ground truth POIs. *re@N* measures how many POIs in the ground truths has returned as top-N recommendation.

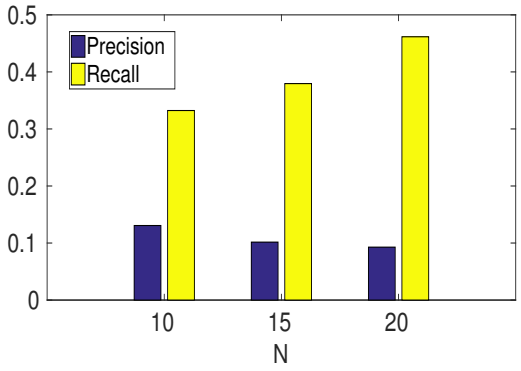

Figure 5.10: Precision and Recall

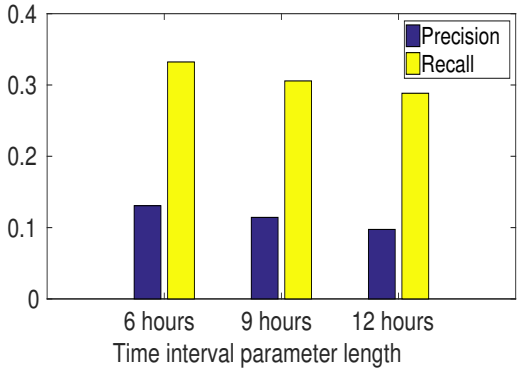

Figure 5.11: Effects of different T_{max}

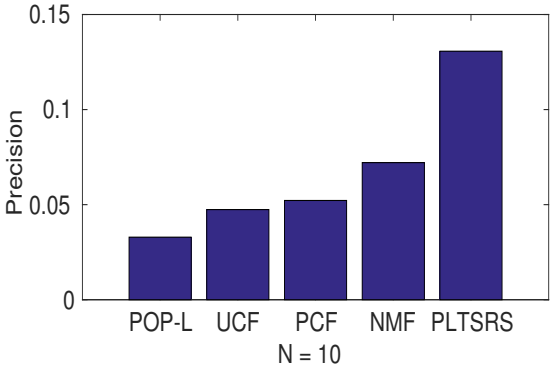

Figure 5.12: Pre@10

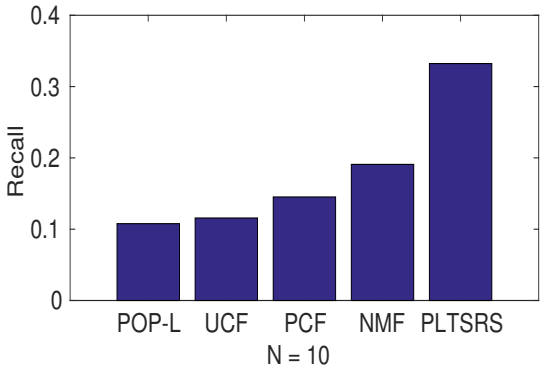

Figure 5.13: re@10

5.5.2 Experimental Results

We have used $d = 10$ km for grid cell size in all our experiments. Figure 5.10 shows the precision and recall value of our proposed method. We show the results for $N = 10$, 15 and 20. In this result we use $T_{max} = 6$ hours.

We compare our method with the four following baseline approaches,

1) **Popularity-Based Recommendation Method (Pop-L)**: This is a spatial-aware popularity based recommendation method. Based on the current location, it first generates spatial aware candidate locations. Candidate locations are ranked based on their popularity.

2) **Location-Based Collaborative Filtering (UCF)**: This method applies Collaborative Filtering method directly over locations. This baseline utilizes the users' location histories with a user-location matrix. User similarity is calculated using the location vector of users. Finally the locations are ranked using *CF* method. We consider the current location as a query location and generates spatial-aware candidate locations first to adapt this model for successive recommendation.

3) **Preference-Based Collaborative Filtering (PCF)**: This method is the baseline Preference-Aware approach. This method first generates users' categorical

preferences from their location histories. Then it generates user-preference matrix. Similarity between two users is calculated using their preference vector. Finally *CF* method is used to rank the candidate locations.

4) **Non-negative Matrix Factorization (NMF)**: This is the base-line low rank non-negative matrix facorization based recommendation method. This method first generates user-location matrix using their location histories. User-location matrix is factorized into two low rank matrices W and H.

Note that all methods use the current location as query location. We find spatial aware candidate locations first to adapt them for successive location recommendation. Figure 5.12 and Figure 5.13 show the precision and recall values respectively. UCF works better than Pop-L. PCF approach works better than UCF as PCF can handle the data sparsity problem. NMF approach works better than PCF, but our proposed method PLTSRS outperforms all other baseline approaches.

We change the value of T_{max} to see how the value of T_{max} affects the results. Figure 5.11 shows the precision and recall of our algorithm for $T_{max} = 6$ hours, 9 hours and 12 hours. $T_{max} = 6$ hours gives us the best result.

5.6 Conclusion

In this chapter, we present a novel approach for successive POI recommendation task. This approach recommends to a user a set of locations where he might be interested to visit next based on his current location and time. This method considers a combination of users' time-specific categorical preferences, categorical transition patterns, spatial influences and popularity of POIs. To the best of our knowledge, this is the first work that combines all the factors (temporal, user-preferences, categorical transition patterns, spatial and popularity) for successive POI recommendation task. Experimental results show that our method outperforms other baseline approaches.

In future work, we plan to incorporate social relationships to strengthen our recommendation model.

CHAPTER 6

Preference Aware Travel Route Recommendation with Temporal Influence

6.1 Introduction

There have been vast advances and rapid growth in Location based social networking (LBSN) services in recent years. Foursquare[1], Yelp[2] and Facebook places[3] are a few examples of LBSN services. LBSNs allow users to share their life experiences via mobile devices. Users post their presence or arrival to a physical location, also known as *Point of Interests* (POIs). The process is known as "Check-in". Users can also share their experiences by leaving comments or tips on that location. A *Point of Interest* (POI) location can be a "Restaurant", "Travel spot", "Park" and so on.

Users movement data with location and time information provide us better knowledge about their activities and interests. The availability of such information opens up an array of new research problems and various real world applications. POI recommendation systems and Travel Route recommendation systems are examples of such real world applications.

In this work, we focus on the travel route recommendation problem. Traveling is one of the most important entertainment activities in modern society. Traveling in an unfamiliar city requires knowledge about the interesting and popular attractions of that city. Also the movement sequence from one location to another location should

[1]www.foursquare.com

[2]www.yelp.com

[3]https://www.facebook.com/places

be planned effectively so that it maximizes user satisfaction under a specific time constraint.

A travel route recommendation problem has two major steps: *a*) How to find interesting attractions for a user in an unfamiliar city, *b*) How to plan the set of attractions as a travel route based on specific time constraints.

Finding interesting locations has the following challenges:

1) **Personal Preferences:** Personal preferences of different users are different. For example, a food lover is more likely to be interested in exploring better quality restaurants, while others may have interest in exploring art museums or historical places. A user visiting an unfamiliar city may not know which interesting POIs are available there to visit. The recommendation system should recommend POIs that match the user's personal interests.

2) **Temporal Influences:** User activities are significantly influenced by time [6]. For example, users are more likely to visit a restaurant rather than a bar at noon. Some people often want to visit a beach at noon, some may prefer to visit during late afternoon. So recommended POI locations should match with user's time specific personal interests.

3) **Popularity:** Choosing a POI can be influenced by the popularity or rank of a POI. People may visit a far away place if the place is very popular.

The second step of travel route recommendation is to plan the set of locations with corresponding time information such that it maximizes user satisfaction. This step has the following challenges:

1) **Sequences of locations:** A set of locations are ordered to build a travel route. The order of the locations is important. One may go to a coffee shop to grab a cup of coffee first, then head to visit museum or other POIs. On a weekend people often go shopping, then go to a restaurant for dinner or lunch.

2) **Transition time from one location to another location:** A travel route is a sequences of locations with the corresponding check-in time information. In our model, the transition time between two locations indicates the traveling time between the first location to second location plus the staying time at the first location. But Check-in data does not have this detailed information. It contains only the information of arriving time (check-in) of a user at a particular location. It does not tell us when the user has left from that location. So it is necessary to predict the transition time of a location pair, as the user has a certain time constraint.

3) **Large Search Space:** It is very important to plan a travel route efficiently, because the dataset has a large number of POI locations. Generating a k-length route with sequences of locations using *brute-force* search is not an efficient solution. For example, with 100 POIs in total, the number of trips that consists of 5 POIs can reach billions. Most of these candidate trips will not follow the time constraint budget. Also the locations of these trips may not match with user preferences. So, an efficient strategy to generate travel routes based on user preferences and time constraint budget is essential for such applications.

In this chapter, we propose a novel algorithm to generate preference-aware and time-aware travel route recommendation system. Our goal is to recommend top-K travel routes with the combination of interesting locations that will match with users' time specific interests. Each location of a route will be associated with the corresponding approximated time information. By this recommendation, the user will know not only where to go, but also when to go. The contribution of the chapter can be summarized as follows:

1) We incorporate time dimension to model time-specific user preferences, so our recommendation model aims to recommend the trips with the POI locations that match the time-specific preferences of individual user.

2) We estimate the similarity between two users based on the user-preference vector rather than the user-location vector. There are 3 main reasons behind this. **First**, it handles the data sparsity problem of user-location matrix. **Second**, two users who do not visit the exact same venue may still share common interest if their preferences are the same. **Third**, as users may visit an unfamiliar city, the dataset may not have any check-in information of the locations of that city for the user. In that case, user similarity based on user-location matrix will completely fail.

3) Our recommendation model uses the popularity factor of individual locations by calculating time-specific popularity.

4) We build a model to estimate the uncertain transition time between two locations.

5) We propose a novel framework *"PTTR-Reco"* (**P**reference-Aware, **T**ime-Aware **T**ravel **R**oute **Reco**mmendation) to recommend top-K travel routes to users.

6) We evaluate our proposed framework with one large scale LBSN dataset from *Foursquare*.

The rest of the chapter is organized as follows. We review the related work in Section 6.2. In Section 6.3, we present the definitions and formulate our problem statement. In Section 6.4, we present our proposed framework in detail. Section 6.5 describes the experimental results. Finally, Section 6.6 concludes the chapter.

6.2 Related Work

Recommendation system in Location Based Social Network can be categorized into two categories. 1) POI recommendation system, 2) Trip recommendation system.

6.2.1 POI recommendation

The first step of travel planning is to find the popular or interesting POI locations for users. There have been a number of studies that focus on POI recommendation.

In [40], the *User-based CF* approach considers a combination of social influence and spatial influence. In [6], *User-based CF* method is used. They exploit spatial influence as well as temporal influence for building recommendation model. In [7], authors explore user preferences, social influence and geographical influence for POI recommendation. They model user preferences using predefined categorical information of location data.

In [4], authors propose an geographical probabilistic factor analysis framework for recommendation that takes various factors into consideration, viz. user-item preferences, POI popularity and geographical preferences of individual user. In [41], authors proposed friendship based collaborative filtering (FCF) approach for POI recommendation based on common visited check-ins of friends. In [54], the authors propose a *User Based Collaborative Filtering* method based framework which combines 4 factors: categorical preferences, temporal influence, spatial influence and popularity of a location.

Lately a few successive POI recommendation works have been conducted. In [61], authors extended the work from [54] to recommend the next POI location based on the current location and current time. They incorporate personal preference, temporal and spatial influence and categorical transition probability to solve the problem. In [52], the authors propose a probabilistic model to integrate category transition probability and POI popularity to solve the problem. In [56], the authors propose a Factorized Personalized Markov Chain (FPMC) model for next-item recommenda-

tion. In [9], the authors propose *FPMC-LR* model by extending *FPMC* [62] model with localized region constraint to solve successive POI recommendation task.

6.2.2 Trip Recommendation

The purpose of trip recommendation is to plan the popular locations accordingly with a time sequence.

In [63] authors proposed an *Hypertext Induced Topic Search-based (HITS)* inference model [64] to build a travel recommendation framework. In this work, authors used GPS trajectory data. The *HITS* model is based on the assumption that, the interesting places might be visited by the more experienced travel experts, and the experienced tourists might visit more interesting places. However, GPS trajectory data is comparatively difficult to obtain. Also mining interesting locations from GPS trajectory data is not straightforward.

In [65] authors proposed an ontological travel recommendation system for Tainan city. In this work, authors first find the top three POI locations and top five restaurants. Then they plan the optimal trip based on those eight locations by ant colony optimization algorithm [66]. The main drawback of this method is that, the number of location is restricted to 8, where 5 of them are restaurants.

In [67], authors evaluated the quality of a trip using 4 metrices: *Elapsed Time Ratio (ETR)*, *Stay Time Ratio (STR)*, *Interest Density Ratio (IDR)* and *Classical Travel Sequence Ratio (CTSR)*. Candidate travel routes are ranked using Euclidean distance and *CSTR*.

In [68], authors used geo-tagged photos to extract features and topics of the attractions. First they find the similarity between location feature and user profile. Then three approximate methods are proposed to generate the travel route. The approximate methods they proposed are: *d-LOA*, *v-LOA*, *GOA*.

In [69], authors also used Panoramio geo-tagged photos to plan the trip. In this paper, authors used a dynamic programming method to generate the optimal trip. However, they did not consider efficiency of the algorithm.

In [70], authors consider an additional constraint POI category visiting order (e.g., restaurant → park → shopping → restaurant) to evaluate travel route. In [71], authors consider two constraints, 1) *POI availability* where a POI may be available only a certain time window and 2) *Uncertain traveling time* where the traveling time between two locations is uncertain. Then they proposed an efficient solution to generate an optimal trip which is based on two pruning strategies.

In [72], authors propose to utilize users' check-in patterns to recommend popular, time sensitive trips. They proposed that a good route should be evaluated based on 4 features, 1) popularity of locations, 2) the proper time to visit a place 3) the amount of time required to transit from one place to another and 4) visiting order of places. Then they proposed a greedy solution to generate the optimal k-length trip. The drawback of this method is that, they did not consider user preferences. Also the length of route is a user defined parameter.

In [73], authors present a *Personalized Trip Recommendation (PTR)* framework which recommended personalized arrangement of visit to venues, given a predefined budget(e.g. time, money).

6.3 Problem Statement

In this section, we first define some terms used in this work. Then we present the problem statement of travel route recommendation problem.

6.3.1 Definition 1: Users

Each user u is represented by a unique id. Let $U = \{u_1, u_2, u_3, ...u_n\}$ be a set of users.

6.3.2 Definition 2: POI locations

Let $L = \{l_1, l_2, l_3, ...l_m\}$ be the set of m locations. Let $C = \{c_1, c_2, c_3, ...\}$ is a set of categories. Each location $l \in L$ is associated with geographical position (latitude, longitude) and category information. Let $c(l)$ be the category of location l.

6.3.3 Definition 3: Check-ins

"Check-in" is a process by which a user u announces his physical arrival or presence at a venue in location based social network. Let $Ch_{ij} = \{u_i, l_j, t\}$ be a check-in tuple, which represent that user u_i checked in POI l_j at time t.

6.3.4 Definition 4: Travel Route

A travel route is a sequence of locations ordered by timestamps. A travel route $r = l_1 \rightarrow l_2 \rightarrow ... \rightarrow l_n$ is a n-length route. Here n indicates the number of POI locations.

6.3.5 Definition 5: Time-Aware Travel Route

A time-aware travel route is a sequence of locations with the corresponding visiting time information. Let $TR = \{tr_1, tr_2, ...\}$ be a set of routes. A time-aware travel route $tr \in TR$ is defined as $tr = \{(x, t_0) \rightarrow (l_1, t_1) \rightarrow .... \rightarrow (l_k, t_k) \rightarrow (y, t_e)\}$. Here x is the source and y is the destination. l_1, l_2,...l_k are the k visiting locations in tr. The departure time of trip is t_0 and the end time is t_e. Here, length of the route is $n = k + 2$.

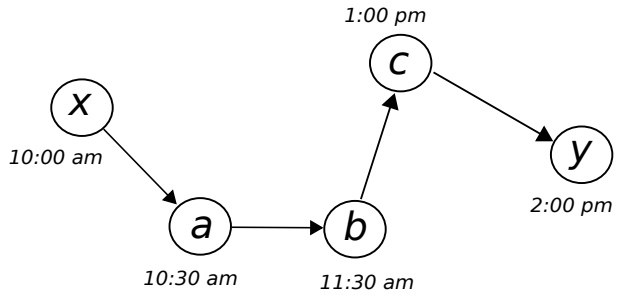

Figure 6.1: A time-aware travel route

Consider the example in Figure 6.1. Here a user starts his travel from x on 10:00 am. He visited 3 locations a, b and c. He reaches at a on 10:30 am. He might stay at a for some time and then arrives at b on 11:30 am. After staying some time at b, he arrives at c on 1:00 pm. Finally he reaches his destination on 2:00 pm.

6.3.6 Definition 6: Travel Time

Travel Time between two locations l_1 and l_2, denoted as $\tau(l_1, l_2)$ is the time it takes to travel from l_1 and l_2. Here, we only consider traveling by car to simplify the model.

6.3.7 Definition 7: Stay Time

Stay Time of location l_1, denoted as $\sigma(l_1)$ is the amount of time a person stays at location l_1.

6.3.8 Definition 8: Transition Time

Transition Time between two locations l_1 and l_2, denoted as $\Gamma(l_1, l_2)$ represents the amount of time the user stays at location l_1 plus the time it takes to travel from l_1 to l_2. $\Gamma(l_1, l_2)$ is defined as equation(1).

$$\Gamma(l_1, l_2) = \sigma(l_1) + \tau(l_1, l_2) \tag{6.1}$$

6.3.9 Definition 9: Trip Time

Trip Time of a trip represents the total time required to complete a trip. In a trip, starting from source location, user travels from one visiting location to another visiting location as well as staying at the visiting locations for a specific time period. Finally they returned to the destination. Trip time of a trip $tr = \{(x, t_0) \rightarrow (l_1, t_1) \rightarrow (l_2, t_2) \rightarrow, ... \rightarrow (l_k, t_k) \rightarrow (y, t_e)\}$, denoted as

$$T(tr) = \tau(x, l_1) + (\sum_{i=1}^{k-1} \Gamma(l_i, l_{i+1})) + \Gamma(l_k, y) \tag{6.2}$$

6.3.10 Definition 10: Time Constraints

Time constraint TC defines the total time the user u has to complete a trip.

6.3.11 Definition 11: Valid Route

A travel route tr is called valid route if Trip Time of a route Time $T(tr)$ is less than or equal to Time Constraint TC.

6.3.12 Problem Statement

Given a user u, source location x, destination location y, departure time t_0, time constraint TC, our goal is to find top-K travel routes under the following constraints: 1) Route starts at location x, 2) Route ends at location y and 3) Travel time of route $\leq$ than TC. Here TC is a user-defined parameter.

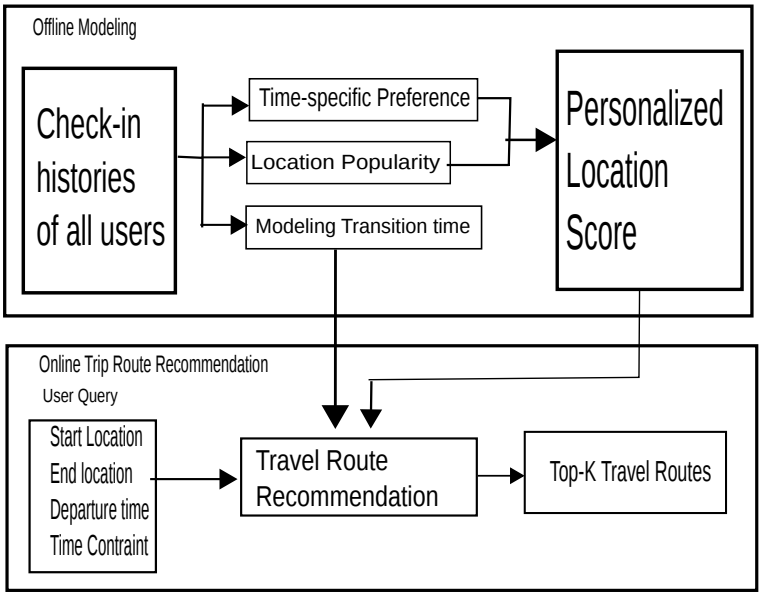

Figure 6.2: System Framework

6.4 Proposed Framework

In this section, we first describe our proposed system framework of *Preferece-Aware, Time-Aware Travel Route Recommendation (PTTR-Reco)* system. Our proposed framework has two major steps: 1*) Offline Modeling*, 2) *Online Travel Route Recommendation.*

The offline modeling has 3 components. 1) *Time-specific Personal Preference Discovery*, 2) *Calculating Popularity of Locations* and 3) *Modeling Transition Time between the Location Pairs.*

In the first component, we learn each user's personal categorical preference on category c denoted as $P_u(c)$. As preference depends on time, we learn time-specific categorical preference on category c at time segment t_s denoted as ${P_u}^{(ts)}(c)$.

In the second component, we calculate the popularity of each POI location l denoted as $\rho(l)$. POI locations for each user will be evaluated based on the two components: user's time specific categorical preference and location popularity.

In the third component, we model the uncertain transition time between two locations. Transition time depends on the time user is staying at a location and the traveling time between two locations. Check-in information only contains the time user has arrived at a location. It does not have the information how much time the user stays at a location. In this component, we model and pre-compute the estimated transition time between the location pairs. Note that, the offline components are calculated only once and will be used in online trip recommendation algorithm.

The second step is *Online Travel Route Recommendation*. In this step, user will input a source location (x), destination (y), departure time (t_0) and time constraint (TC). Based on the user input, our system will generate a set of travel routes that follows the specific time constraint. Each route tr is a sequence of POI locations, where each POI location is associated with the corresponding visiting timestamp. Each route is evaluated based on the locations recommended to the user. Finally top-K travel routes are recommended to user u.

6.4.1 Offline Modeling

The offline modeling has 3 components. 1) *Time-specific Personal Preference Discovery*, 2) *Calculating Popularity of Locations* and 3) *Modeling Transition Time between the Location Pairs*.

6.4.1.1 Modeling User Preference

In this step, we model each individual user's categorical preferences from his/her check-in history. Categorical preference of a user u denoted as $P_u(c)$ represents u's affinity to visit a location with category c. $P_u(c)$ is generated using following equation [54].

$$P_u(c) = CF(c, L_u) \times ILF(c, L) \tag{6.3}$$

Here $CF(c, L_u)$ is the measure of how many times user u has visited the locations with a category c. Intuitively, a user would visit more locations belonging to a category if he likes it. Here L_u is the location set visited by u. ILF handles the *Rare-Item* problem [45]. Some locations are not visited by a user very often. For example, the number of visits to a restaurant is generally more than that of a museum. If a user visits location of a category that is rarely visited by other users, it means that the user could like this category more prominently [7].

CF is calculated using eq. (6.4) and ILF is calculated using eq. (6.5).

$$CF(c, L_u) = \frac{|\{u.l_i : l_i \in c\}|}{|L_u|} \tag{6.4}$$

$$ILF(c, L) = \log \frac{|U|}{|\{u_j.l \in L : l_j.c \in u_j.C\}|} \tag{6.5}$$

Here, $|\{u.l_i : l_i \in c\}|$ is user u's number of visits in category c, $|L_u|$ is the total number of user's visit in all locations. $|U|$ is the number of total users in the system. $|\{u_j.l \in L : l_j.c \in u_j.C\}|$ is the number of users who visit category c among all users in U.

6.4.1.2 Modeling Time-specific User Preference

As categorical preference may vary over time, we intend to find time-specific categorical preferences of each user. We divide the whole day into equal length of time segments (t_s). In this chapter, we use time segment length = 1 hour. So the whole day is divided into 24 time segments. Given a user u, time segment t_s, category

c, temporal preference of user u on category c, denoted as $P_u^{(t_s)}(c)$ is calculated using following equation [54].

$$P_u^{(t_s)}(c) = CF^{(t_s)}(c, L_u^{(t_s)}) \times ILF(c, L^{(t_s)}) \tag{6.6}$$

Here $CF^{(t_s)}(c, L_u^{(t_s)})$ is the *Category Frequency* of user u and category c at time segment t_s. $L_u^{(t_s)}$ is the location set visited by u at t_s. $ILF(c, L^{(t_s)})$ is the *Inverse Location Frequency* for category c. $L^{(t_s)}$ is the list of all locations that has been visited at t_s. $CF^{(t_s)}(c, L_u^{(t_s)})$ and $ILF(c, L^{(t_s)})$ are calculated using following equations.

$$CF^{(t_s)}(c, L_u^{(t_s)}) = \frac{|\{u.l_i^{(t_s)} : l_i^{(t_s)} \in c\}|}{|L_u^{(t_s)}|} \tag{6.7}$$

$$ILF(c, L^{(t_s)})) = \log \frac{|U^{(t_s)}|}{|\{u_j.l \in L : l_j.c \in u_j.C\}|} \tag{6.8}$$

Here, $|\{u.l_i^{(t_s)} : l_i^{(t_s)} \in c\}|$ is the number of visits by user u at category c at time segment t_s. $|L_u^{(t_s)}|$ is total visits by user u at time t_s. $|U^{(t_s)}|$ is the total number of unique users in the system that has checked-in at time t_s. $|\{u_j.l \in L : l_j.c \in u_j.C\}|$ is the total number of unique users that visit at category c at time t_s.

6.4.1.3 Modeling Uncertain Transition Time

Transition time between two locations $\Gamma(l_i, l_j)$ indicates the traveling time between l_i and l_j plus the stay time at l_i. Check-in data cannot explicitly give us the both two individual components: stay time and travel time. Hence, for simplicity, we treat the time difference between two successive check-in locations (l_i, l_j) as the sum of stay time at l_i and travel time between l_i and l_j.

To estimate the transition time between a location pair, we first find all the consecutive location pairs with their corresponding timestamp from the historical

data. Given a location pair $\{(l_i, t_i), (l_j, t_j)\}$, transition time is $TT_{ij} = t_j - t_i$. For each location pair (l_i, l_j), we get a set of different transition times as the timestamp for the same location pair can be different. Let $\delta = \{\delta_1, \delta_2.....\delta_p\}$ is a set of p transition time for (l_i, l_j) generated from historical data.

To avoid noisy data, we first removed the outliers from δ. We use the interquartile range [74] method to find the outliers from δ. The interquartile range (IQR) is a measure of variability, based on dividing a data set into quartiles. Quartiles divide a sorted data set into four equal parts. The values that divide each part are called the first quartile(Q_1), second quartile(Q_2), and third quartile(Q_3). Data values that fall below $Q_1 - 1.5(IQR)$ or above $Q_3 + 1.5(IQR)$ are considered as outliers.

Let $\xi = \{\xi_1, \xi_2, ...\xi_q\}$ is a set of q outliers generate from δ. Transition time $\Gamma(l_i, l_j)$ is estimated using following equation:

$$\begin{aligned} \Delta &= \delta - \xi \\ \Gamma(l_i, l_j) &= E[\Delta] \end{aligned} \tag{6.9}$$

6.4.1.4 Learning Popularity of Locations

Popularity of a location plays a significant role to attract users. People tend to visit a more popular POI for better satisfaction. However, popularity also varies over time. For example, a bar is more popular at night, whereas people tend to visit a museum during morning or afternoon. For better recommendation, we intend to calculate popularity score of each POI on each time segment. Popularity of a POI l at time t_s is calculated using following equation [54].

$$\rho^{(ts)}(l) = \frac{1}{2} * \left\{ \frac{|U^{(t_s)}(l)|}{|U(l)|} + \frac{|Chk^{(t_s)}(l)|}{Chk(l)} \right\} \tag{6.10}$$

Here $|U^{(t_s)}(l)|$ is the number of users that visited l at time t_s, $|U(l)|$ is the total number of users visited l. $|Chk^{(t_s)}(l)|$ is the number of check-ins at l at time t_s and $Chk(l)$ is total number of check-ins at location l.

6.4.2 Personalized Temporal based Location Scoring

For each POI location, the travel route recommendation algorithm needs to understand how interesting the POI location is for a specific user at the specific time to visit. This task is called the *Personalized Location Scoring*.

6.4.2.1 Definition 12: Personalized Location Score

Given a user u, time t, the personalized location score of l is denoted as $LS(u, t, l)$. LS is the measure of how interesting a location is for u at time t. We used Preference-Aware User-based Collaborative Filtering [54] method to measure $LS(u, t, l)$.

Given a user u, the location score that u will visited location l at time t is given below

$$LS(u, t_s, l) = \frac{\sum_{v \in U} w_{uv}}{|v|} \times P_u^{t_s}(c_l) \times \rho^{t_s}(l) \tag{6.11}$$

Here v is list of users who also visit l. c_l is the category of location l and $P_u^{t_s}(c_l)$ is the preference score of u and c_l at time t_s. $\rho^{t_s}(l)$ is the popularity score of location l at time segment t_s.

6.4.2.2 Definition 13: Travel Route Score

Travel Route Score, denoted as $TS(u, tr)$ is a measure of how interesting the route tr is for user u. Travel route score depends on the visiting locations of the routes.

Given a travel route $tr = \{(x, t_0) \rightarrow (l_1, t_1) \rightarrow (l_2, t_2) \rightarrow, ... \rightarrow (l_k, t_k) \rightarrow (y, t_e)\}$ and user u, travel route score $TS(u, tr)$ is defined as:

$$TS(u, tr) = \sum_{i=1}^{k} LS(u, l_i, t_i) \tag{6.12}$$

6.4.3 Online Travel Route Recommendation Algorithm

In this section, we present an *Apriori-based* [75] travel route generation algorithm. Given a user u, source location x, destination y, Time Constraint TC, this algorithm will generate top-K travel routes dynamically. The basic idea of our route generation algorithm is to extend a k-length route (tr_k) to $(k + 1)$-length (tr_{k+1}) gradually by inserting a new location into the route.

6.4.3.1 Definition 14: Baseline Route

Given source x, and destination y, we define the 2-length baseline route as

$$tr_2 = \{(x, t_0) \rightarrow (y, t_e)\} \tag{6.13}$$

Here t_0 is the departure time and t_e is the end time of tr_2. Trip Time of tr_2 is denoted as $T(tr_2) = t_e$ - t_0. Here, trip time is the same as the travel Time $\tau(x, y)$. tr_2 is valid, if the $T(tr_2) \leq TC$ (Definition 11).

We intend to generate a 3-length route by extending the baseline route. In order to do that, we insert a new location l_i just before the destination, and right after the source location. For example, a 3-length trip $tr_3 = \{(x, t_0) \rightarrow (l_i, t_i) \rightarrow (y, t_e)\}$. Here, l_i is the visiting location and t_i is the visiting time of location l_i. A similar procedure will continue to generate $(k + 1)$-length route using k-length route.

The simplest approach to implement this algorithm is the *Brute-Force* method. We first discuss the *Brute-Force* strategy to solve the route generation algorithm. Then we discuss computational complexity of the *Brute-Force* approach.

6.4.4 Brute-Force Approach

For the travel route generation problem, the simplest approach is to address the issue using *Brute-Force* strategy. With *Brute-Force* strategy, we will consider all the locations in the dataset and generate all the possible travel routes. For each route, we check whether the route is valid or not. Then we will calculate the score for all the valid routes. Finally top-K routes will be recommended to the user.

6.4.4.1 Complexity of *Brute-Force* approach

Let n be the number of POI locations. Given source x, destination y, the number of possible trips using the other (n-2) POI locations is formulated as

$$Complexity = 1 + \sum_{i=1}^{n-2} (C_i^{n-2} \times i!) \tag{6.14}$$

Proof Baseline route can be constructed in one way. Having the source and destination fixed, the other (n-2) visiting locations will be used to build the routes. For each route of length i, the number of combination is C_i^{n-2}. Again, for each combination of route of length i can be permuted in different $i!$ ways. So, number of possible routes of length i is the number of combinations multiplied by the number of permutations ($C_i^{n-2} \times i!$).

Consider an example that, we have source x, destination y and other 4 visiting POI locations a, b, c and d. Based on the Equation 14, total number of possible route

is 65 (including the baseline route), where the total number POI location is only 6 (including source, destination).

6.4.4.2 PTTR-Reco Algorithm

In this section, we present our proposed travel route recommendation framework, "*PTTR-Reco*"(***P****reference-Aware,* ***T****ime-Aware* ***T****ravel* ***R****oute* ***Reco****mmendation*) that finds the top-K travel routes in location based social network service. In order to improve efficiency over *Brute-Force* method, we propose two new strategy: 1) *Preference Aware Candidate Location Set Generation* and 2) *Time Constraint Based Travel Route Pruning.*

6.4.4.3 Preference-Aware Candidate Location Set Generation

The *Brute-Force* considers all locations in the dataset and generate all possible travel routes. But our goal is to find the travel routes with the locations that match with user's personal categorical preference. Consider the example in Figure 6.3. To build a 3-length trip using baseline 2-length routes, *Brute-Force* method will consider all visiting locations (a,b,c and d) as a next location of source x. So, this method will generate total 4 3-length routes. At the same way, these 4-length routes will generate total 12 5-length routes.

In this chapter, we focus on to recommending *Preference-Aware* and *Time-Aware* travel routes with the combination of location sequences that will match users' personal interests. Personal preference plays an important role for a user to choose POI. To have a better idea, we count the number of unique categories visited by users. We sort the users based on the count and plot the result (see Fig. 6.4). We see that the number of categories visited by most of the users is less than 60. Users generally

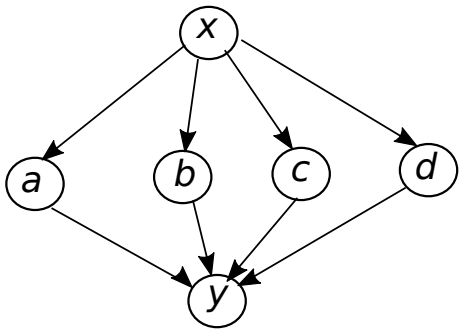

Figure 6.3: Given source, destination and 4 other visiting locations, building 3-length trips

do not visit the locations of all categories, they visit a location only if they like the category.

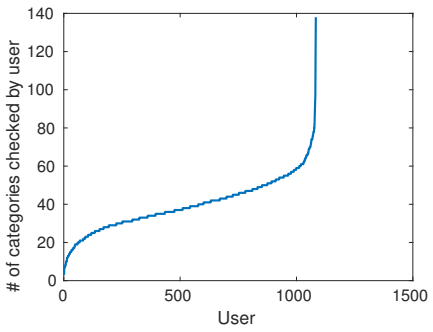

Figure 6.4: Number of unique categories checked-in by users, (Number of users = 1083)

Another criteria to choose a POI location is the sequences of locations. People often follow a periodic patterns. One may go to a coffee shop to grab a cup of coffee first, then head to work or university.

Motivated by the two properties, we propose a method to generate candidate locations as a next location. Given a current location l_i, to select the next location, we find all the continuous location pairs starting with l_i from historical check-in dataset. Let $L = (l_i, l_j),(l_i, l_k),(l_i, l_p),.....(l_i, l_s)$ be the location pairs of l_i. The location set

that follows l_i immediately in the dataset are considered as candidate locations as the next location of l_i.

Consider the k-length route be $tr_k = x \rightarrow .....l_i \rightarrow y$, where l_i be (k-1)th location. To build a $(k+1)$-length trip tr_{k+1}, we insert a location right before the destination location y and right after the location l_i. Instead of considering all the locations in the dataset, we consider only the candidate location set $C_{l_i} = \{l_j, l_k, ...l_s\}$.

We further reduced the Candidate Location set based on Preference Criteria. We only consider those locations if the user's categorical preference of that location is greater than a certain threshold (θ).

Preference Aware Candidate Location Set Generation method reduces the computation cost effectively. Still, the computation cost will depend on the number of candidate locations. We further use another pruning strategy to reduce the computation cost.

6.4.4.4 Time Constraint Based Travel Route Pruning

We recall the Definition (11), A travel route tr is called valid route if Travel time of a route Time $T(tr)$ is less than the Time Constraint value (TC). In other words, a travel route is invalid if $T(tr) > TC$.

In this section, we prove that a super route of an invalid route is invalid. In other words, given an invalid trip tr_{inv}, any super-trip that contains tr_{inv} must be an invalid trip. Hence all the super-trip of tr_{inv} can be pruned.

Proof Consider an invalid trip $tr_{inv} = x \rightarrow p \rightarrow y$. As tr_{inv} is invalid, $T(tr_{inv}) > TC$

now, $T(tr_{inv}) = \tau(x,p) + \Gamma(p,y)$ or, $T(tr_{inv}) = \tau(x,p) + \sigma(p) + \tau(p,y) > TC$

Consider a super trip of tr_{inv} is tr'_{inv} by adding a new POI q into it. $tr'_{inv} = x \rightarrow p \rightarrow q \rightarrow y$. We want to prove that $T(tr'_{inv}) > TC$. We have,

$T(tr'_{inv}) = \tau(x,p) + \Gamma(p,q) + \Gamma(q,y)$

$T(tr'_{inv}) = \tau(x,p) + \sigma(p) + \tau(p,q) + \sigma(q) + \tau(q,y)$

In order to proof $T(tr'_{inv}) > TC$, we have to prove that, $\tau(p,q)+\sigma(q)+\tau(q,y) \geq \tau(p,y)$.

Given any 3 locations a, b and c, and their corresponding distances be d_{ab}, d_{bc}, d_{ac}. They must follow the triangular inequality [76] concept. Hence, the sum of the distance between two routes must be greater than or equal to the third route.

$$d_{ab} + d_{bc} \geq d_{ac}$$

$$d_{ac} + d_{cb} \geq d_{ab}$$

$$d_{bc} + d_{ca} \geq d_{ba}$$

As travel time is directly proportionate to the distance (to simplify our model, we do not consider traffic issue), the same inequality holds for travel time between 3 locations.

$$\tau(a,b) + \tau(b,c) \geq \tau(a,c)$$

$$\tau(a,c) + \tau(c,b) \geq \tau(a,b)$$

$$\tau(b,c) + \tau(c,a) \geq \tau(b,a)$$

It proves that, $\tau(p,q) + \tau(q,y) \geq \tau(p,y)$.

so, $\tau(p,q) + \sigma(q) + \tau(q,y) > \tau(p,y)$, as $\sigma(q) \geq 0$.

Based on this pruning strategy, if we find a route that cannot be finished in a time budget, we prune that route and it is not necessary to extend the route further.

6.4.4.5 Algorithm

We present an *Apriori-based* algorithm *PTTR-Reco* in Algorithm 11. In the offline method, we calculated *Time-specific Category Preference (P_u)*, *Location Popularity (ρ)*, *Set of Location Pair with estimated Transition Time (LP)*. These results are calculated only once. The user input query to build trip is Q = (u, x, y, t_0, TC),

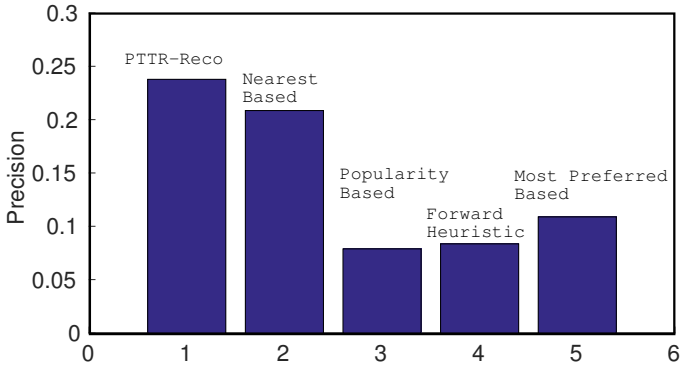

Figure 6.5: Tour Precision

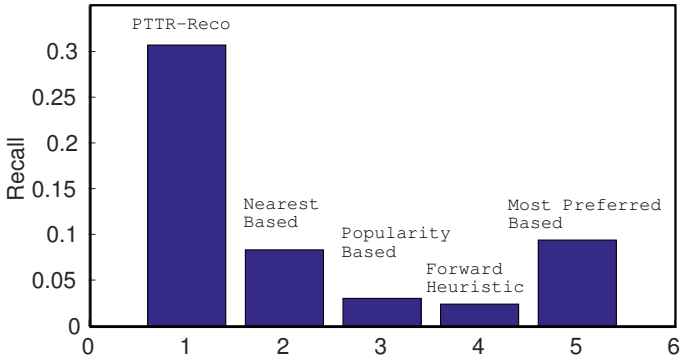

Figure 6.6: Tour Recall

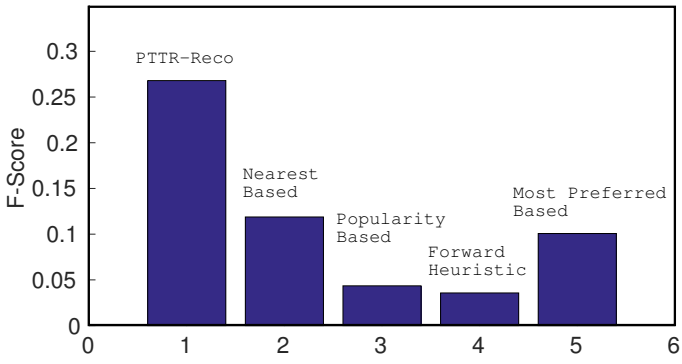

Figure 6.7: Tour F-Score

which are user id, source, destination, departure time and Time Constraint respectively.

6.5 Experiments

In this section, we describe the settings of our experiments that includes dataset and method of evaluation of our proposed method. Then, we present the experimental results of our method.

6.5.1 Experimental Settings

6.5.1.1 Dataset

We use one real-world check-in datasets from Foursquare. Dataset includes $227,428$ check-in data from New York City, USA. The dataset has data from 12 April 2012 to 16 February 2013 (10 months). Each check-in data Ch_{ij} contains a user (u_i), a location id (l_j) and the corresponding time (t). Each location id l_j is associated with geographical position (latitude,longitude), category (c_j). It contains check-in data of $1,083$ users and $38,383$ locations. To get more effective results, we removed POIs that have lower than 5 check-ins. After preprocessing, the dataset contains $4,597$ POI locations and $164,307$ check-ins. For each user, we randomly mark off 50% of his location histories as a training set to learn his temporal categorical preferences and location preferences. The other 50% is used as a test set.

6.5.1.2 Evaluation Methods

We use three measures to evaluate the results of our experiments: Tour recall, Tour Precision and Tour F-measure. They were also used in [77] for travel route

evaluation. Let P_r be the set of POIs recommended in a travel route tr and P_v be the set of POIs the user has visited in real life.

1. **Tour Recall (Trec)**: Tour Recall is the ratio of user's real life visited POIs that were also recommended in the travel routes. Tour recall is defined as $TRec(tr) = \frac{|P_r \cap P_v|}{|P_v|}$.

2. **Tour Precision (TPre)**: Tour precision is the ratio of POIs recommended in route tr that were also visited by user in real life. Tour Precision is defined as $TPre(tr) = \frac{|P_r \cap P_v|}{|P_r|}$.

3. **Tour F_1 Score (TF)**: Tour F_1 score is defined as $TF(tr) = \frac{2 \times Trec(tr) \times TPre(tr)}{Trec(tr) + TPre(tr)}$.

6.5.1.3 Baseline approaches

We use the following baseline approaches to evaluate the effectiveness of our framework. These approaches use fixed transition time model, where transition time between two location l_i and l_j is fixed.

1. **Nearest POI Approach** This method chooses the nearest POI to the current location as a next location. Haversine [78] formula is used to measure the distance between two locations. Each POI location is evaluated based on the distance from the current location. Trip score is measured using the equation $TS(tr) = (\prod_{i=1}^{n} \frac{1}{d(l_i, l_{i+1})})^{\frac{1}{n}}$.

2. **Popular POI Based Approach** This method chooses the most popular POI to visit as a next location. But as we have time constraint, it is not feasible to choose the popular POI if it is far away. So, this method first chooses the 10 nearest POI, then it chooses the most popular one among them. Trip score is measured using the equation $TS(tr) = \sum_{i=1}^{n} \rho^{(t_s)}(l)$.

3. **Most Preferred POI Based Approach** This method chooses his most preferred POI as a next location to visit. Route is evaluated based on the user's time-

specific categorical preference. Trip score is measured using the equation $TS(tr) = \sum_{i=1}^{n} P^{(t_s)}(c_l)$.

4. **Forward Heuristic Approach** This approach chooses a location l_i that has the highest transition probability with the previous location $P(l_i|l_{i-1})$. Trip score is measured using the equation: $TS(tr) = \sum_{i=1}^{n-1} P(l_i|l_{i-1})$.

We generate the travel route with the highest score using our framework (*PPTR-Reco*). All other 4 baseline approaches also generate one travel routes at a time. We use different current time and average the results. We use $TC = 5h$ in all approaches.

Figure 6.5, 6.6, 6.7 shows the tour precision, tour recall and tour fscore value of our proposed method and other 4 baseline methods. We see that, *PTTR-Reco* outperforms all other baseline approaches.

The nearest POI based method generates second best precision value. The reason is many users generally prefer to visit the nearest locations. The most preferred based method generates the third best results. The popularity based method performs worse among all.

6.5.1.4 Effects of different value of Time Constraint

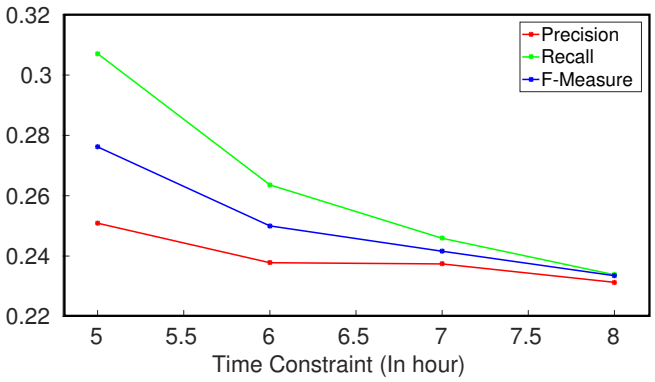

Figure 6.8: Effects of different Time Constraint length

This experiment analyzes the evaluation results of our proposed framework with different Time Constraint value. We use $TC = 5h$, $6h$, $7h$ and $8h$. Figure 6.8 shows the result. We see that smaller value of TC gives us better results.

6.5.1.5 Runtime

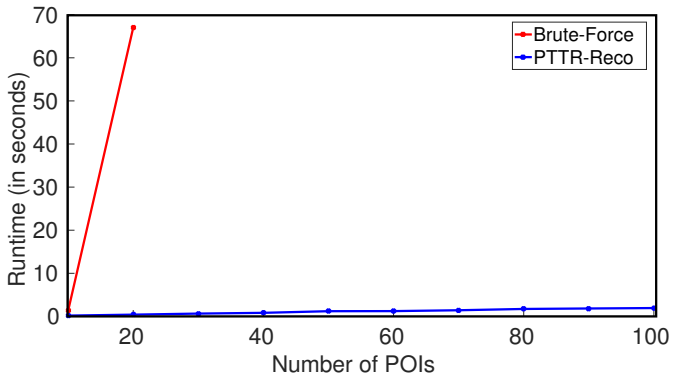

Figure 6.9: Comparison of execution time with Brute-Force method

This experiment analyzes the execution time our proposed framework. Figure 6.9 shows the comparison of executing time between *Brute-Force* approach and our proposed framework. We changed the number of location 10 to 100 to generate travel routes. Y-axis shows the average execution time. *Brute-Force* approach takes around 67 seconds to generate travel routes when the number of location is 20. With this approach run time increases exponentially with the number of locations. It takes more than 3000 seconds with 30 POI locations. It does not work when number of POI location is more than 30.

We analyze the execution time of our proposed framework with the framework without time constraint pruning. Figure 6.10 shows the result. Without the time con-

straint pruning, execution time increases rapidly as the number of location increases. But with the time constraint pruning, the method performs significantly better.

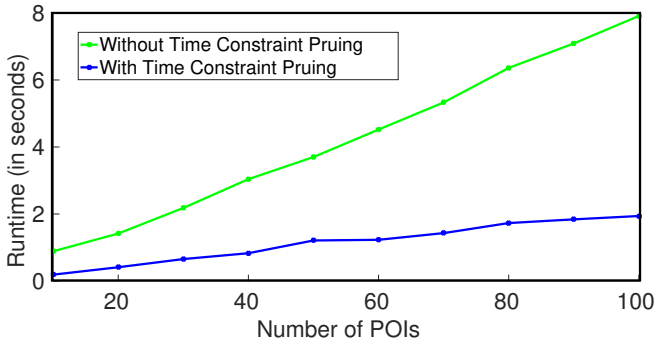

Figure 6.10: Comparison of execution time without time constraint pruning

6.6 Conclusion

In this chapter, we present a novel and efficient framework to recommend time-aware and preference-aware travel routes. This method considers user's personal categorical preference, temporal preference and popularity of a location to find interesting POIs. Then this method presents an efficient solution to generate top-K time aware travel routes. Each POI location of the recommended travel routes is associated with corresponding visiting time information, by which the recommended travel routes will tell us where to visit and when to visit. We experimented our method with other baseline approaches. Experimental results show that our method performs significantly better than other baseline methods.

Our future plan to extend the work is to add the financial budget constraint in addition to time constraint. Our another plan is to incorporate categorical diversity constraint to generate better travel route recommendation.

Algorithm 11 PTTR-Reco

Input: (i) u: User, (ii)x: Source Location, (iii)y: Destination Location, (iv)t_0: Departure Time, (v)TC: Time Constraint, (vi)Output: TR: a set of trip routes with trip score

1. $TR = \{\}$
2. $t_e = \tau(x, y)$
3. $TR_2 = \{(x, t_0) \rightarrow (y, t_e)\}$
4. $TR_3 = \{\}$
5. $Cl_2 =$ Generate Candidate Locations of x
6. $k = 3$
7. **for** each $l_c \in Cl_2$ **do**
8. $\quad t_c = \tau(x, l_c)$
9. $\quad t_e = \Gamma(l_c, y)$
10. $\quad tr = \{(x, t_0) \rightarrow (l_c, t_c) \rightarrow (y, t_e)\}$
11. $\quad$ **if** tr is a Valid Route **then**
12. $\quad\quad s_{tr} =$ Calculate Trip Score
13. $\quad\quad TR_3 = \{TR_3\} \cup (tr, s_{tr})$
14. $\quad$ **end if**
15. **end for**
16. $TR = \{TR\} \cup \{TR_3\}$
17. **while** No $(k+1)$-length trips are found **do**
18. $\quad$ **for** each $tr \in TR_k$ **do**
19. $\quad\quad l_{k-1} =$ (k-1)th location of tr
20. $\quad\quad Cl_k =$ Generate candidate location of l_{k-1}
21. $\quad\quad$ **for** each $l_c \in Cl_k$ **do**
22. $\quad\quad\quad t_k = \Gamma(l_{k-1}, l_c)$
23. $\quad\quad\quad t_e = \Gamma(l_c, y)$
24. $\quad\quad\quad tr_{k+1} = \{(x, t_0) \rightarrow ....(l_{k-1}, t_{k-1})$
25. $\quad\quad\quad \rightarrow (l_c, t_c) \rightarrow (y, t_e)\}$
26. $\quad\quad\quad$ **if** tr_{k+1} is Valid **then**
27. $\quad\quad\quad\quad s_{tr_{k+1}} =$ Trip score of tr_{k+1}
28. $\quad\quad\quad\quad TR_{k+1} = \{TR_{k+1}\} \cup (tr_{k+1}, s_{tr_{k+1}})$
29. $\quad\quad\quad$ **end if**
30. $\quad\quad$ **end for**
31. $\quad$ **end for**
32. $\quad TR = \{TR\} \cup \{TR_{k+1}\}$
33. **end while**
34. Return top-K travel routes

CHAPTER 7

CONCLUSIONS

7.1 Summary of Contributions

In this dissertation, we present novel frameworks and algorithms to analyze spatio-temporal data. We use two types of spatio-temporal data, trajectory data and check-in data. In the first part of dissertation, we use a clustering based method to mine useful information from trajectory data. In chapter 2, we have proposed a novel framework to cluster trajectories. We use a combination of spatial and non-spatial features. In chapter 3, we propose a density-based spatial clustering algorithm which can handle arbitrary shaped clusters and datasets with varying densities.

In the next three chapters, we use check-in data in the location-based social network service. Check-in data contains user movement data along with time information. Analyzing check-in data benefits us with many real-world applications. Recommendation system is one of them. In chapter 4, we present a time-aware, location-aware and preference-aware recommendation system, which provides a user time-specific location recommendation based on user's personal categorical preferences and spatial preferences. In chapter 5, we present a novel approach for successive POI recommendation task. This approach recommends to a user a set of locations where he might be interested to visit next based on his current location and time. This method considers a combination of users' time-specific categorical preferences, categorical transition patterns, spatial influences and popularity of POIs. In chapter 6, we present a novel and efficient framework to recommend time-aware and preference-aware travel routes. This method considers user's personal categorical

preference, temporal preference and popularity of a location to find interesting POIs. Then this method presents an efficient solution to generate top-K time aware travel routes.

REFERENCES

[1] M. Debnath, P. K. Tripathi, and R. Elmasri, "A novel approach to trajectory analysis using string matching and clustering," in *Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on.* IEEE, 2013, pp. 986–993.

[2] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, vol. 96, 1996, pp. 226–231.

[3] M. Debnath, P. K. Tripathi, and R. Elmasri, "Kdbscan: Identifying spatial clusters with differing density levels," in *Data Mining with Industrial Applications (DMIA), 2015 International Workshop on.* IEEE, 2015, pp. 51–60.

[4] B. Liu, Y. Fu, Z. Yao, and H. Xiong, "Learning geographical preferences for point-of-interest recommendation," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2013, pp. 1043–1051.

[5] T. Horozov, N. Narasimhan, and V. Vasudevan, "Using location for personalized poi recommendations in mobile environments," in *Applications and the Internet, 2006. SAINT 2006. International Symposium on.* IEEE, 2006, pp. 6–pp.

[6] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann, "Time-aware point-of-interest recommendation," in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval.* ACM, 2013, pp. 363–372.

[7] J. Bao, Y. Zheng, and M. F. Mokbel, "Location-based and preference-aware recommendation using sparse geo-social networking data," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, 2012, pp. 199–208.

[8] H. Gao, J. Tang, X. Hu, and H. Liu, "Content-aware point of interest recommendation on location-based social networks," in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.

[9] C. Cheng, H. Yang, M. R. Lyu, and I. King, "Where you like to go next: Successive point-of-interest recommendation." in *IJCAI*, 2013.

[10] J. Gudmundsson, A. Thom, and J. Vahrenhold, "Of motifs and goals: mining trajectory data," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, 2012, pp. 129–138.

[11] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 593–604.

[12] J. Hsieh, S.-L. Yu, and Y.-S. Chen, "Trajectory-based video retrieval by string matching," in *Image Processing, 2004. ICIP'04. 2004 International Conference on*, vol. 4. IEEE, 2004, pp. 2243–2246.

[13] Y. Yanagisawa, J.-i. Akahani, and T. Satoh, "Shape-based similarity query for trajectory of mobile objects," in *Mobile data management*. Springer, 2003, pp. 63–77.

[14] C.-S. Chen, C. F. Eick, and N. J. Rizk, "Mining spatial trajectories using non-parametric density functions," in *Machine Learning and Data Mining in Pattern Recognition*. Springer, 2011, pp. 496–510.

[15] http://weather.unisys.com/hurrricane/atlantic/index.html.

[16] S. Gaffney and P. Smyth, "Trajectory clustering with mixtures of regression models," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1999, pp. 63–72.

[17] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." Kdd, 1996.

[18] M. R. Vieira, P. Bakalov, and V. J. Tsotras, "On-line discovery of flock patterns in spatio-temporal data," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2009, pp. 286–295.

[19] Y. Huang, L. Zhang, and P. Zhang, "A framework for mining sequential patterns from spatio-temporal event data sets," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 20, no. 4, pp. 433–448, 2008.

[20] J.-G. Lee, J. Han, X. Li, and H. Gonzalez, "Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1081–1094, 2008.

[21] D. Gusfield, *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

[22] S. Shekhar and S. Chawla, *Spatial databases: a tour*. Prentice Hall Englewood Cliffs, 2003, vol. 2003.

[23] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[24] J. Han, M. Kamber, and A. K. H. Tung, "Spatial clustering methods in data mining: A survey," in *Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS*. Taylor and Francis, 2001. [Online]. Available: http://www-faculty.cs.uiuc.edu/~hanj/pdf/gkdbk01.pdf

[25] E. Kolatch, "Clustering algorithms for spatial databases: A survey," *PDF is available on the Web*, 2001.

[26] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 281-297. California, USA, 1967, p. 14.

[27] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[28] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *ACM SIGMOD Record*, vol. 28, no. 2. ACM, 1999, pp. 49–60.

[29] A. Hinneburg and D. A. Keim, "An efficient approach to clustering in large multimedia databases with noise," in *KDD*, vol. 98, 1998, pp. 58–65.

[30] A. P. Dempster, N. M. Laird, D. B. Rubin, *et al.*, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal statistical Society*, vol. 39, no. 1, pp. 1–38, 1977.

[31] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *ACM SIGMOD Record*, vol. 25, no. 2. ACM, 1996, pp. 103–114.

[32] S. Guha, R. Rastogi, and K. Shim, "Cure: an efficient clustering algorithm for large databases," in *ACM SIGMOD Record*, vol. 27, no. 2. ACM, 1998, pp. 73–84.

[33] L. Ertöz, M. Steinbach, and V. Kumar, "Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data." in *SDM*. SIAM, 2003, pp. 47–58.

[34] W. Wang, J. Yang, and R. Muntz, "Sting: A statistical information grid approach to spatial data mining," in *VLDB*, vol. 97, 1997, pp. 186–195.

[35] G. Sheikholeslami, S. Chatterjee, and A. Zhang, "Wavecluster: a wavelet-based clustering approach for spatial data in very large databases," *The VLDB Journal*, vol. 8, no. 3-4, pp. 289–304, 2000.

[36] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, *Automatic subspace clustering of high dimensional data for data mining applications*. ACM, 1998, vol. 27, no. 2.

[37] L. Duan, L. Xu, F. Guo, J. Lee, and B. Yan, "A local-density based spatial clustering algorithm with noise," *Information Systems*, vol. 32, no. 7, pp. 978–986, 2007.

[38] J. Han and M. Kamber, "Data mining: Concepts and techniques."

[39] Y. Zheng and X. Zhou, *Computing with spatial trajectories*. Springer Science & Business Media, 2011.

[40] M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee, "Exploiting geographical influence for collaborative point-of-interest recommendation," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 2011, pp. 325–334.

[41] M. Ye, P. Yin, and W.-C. Lee, "Location recommendation for location-based social networks," in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2010, pp. 458–461.

[42] https://developer.foursquare.com/categorytree.

[43] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, p. 4, 2009.

[44] S. Scellato, A. Noulas, R. Lambiotte, and C. Mascolo, "Socio-spatial properties of online location-based social networks." *ICWSM*, vol. 11, pp. 329–336, 2011.

[45] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[46] P.-N. Tan and M. Steinbach, "Vipin kumar, introduction to data mining," 2006.

[47] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 1082–1090.

[48] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[49] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," 2011.

[50] https://yelp.com.

[51] http://statspotting.com/foursquare-statistics-20-million-users-2-billion-check-ins/.

[52] J. Sang, T. Mei, J.-T. Sun, C. Xu, and S. Li, "Probabilistic sequential pois recommendation via check-in data," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, 2012, pp. 402–405.

[53] W. Zhang and J. Wang, "Location and time aware social collaborative retrieval for new successive point-of-interest recommendation," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 1221–1230.

[54] M. Debnath, P. K. Tripathi, and R. Elmasri, "Preference-aware poi recommendation with temporal and spatial influence," in *Florida Artificial Intelligence Research Society Conference*. AAAI, 2016, pp. 548–553.

[55] A. A. Markov, "An example of statistical investigation of the text eugene onegin concerning the connection of samples in chains," *Science in Context*, vol. 19, no. 04, pp. 591–600, 2006.

[56] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 811–820.

[57] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan, "Personalized ranking metric embedding for next new poi recommendation," in *Proc. IJCAI*, 2015.

[58] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization."

[59] H. Goodwin, "The haversine in nautical astronomy," in *US Naval Institute Proceedings*, vol. 36, 1910, pp. 735–746.

[60] D. Yang, D. Zhang, V. W. Zheng, and Z. Yu, "Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 1, pp. 129–142, 2015.

[61] M. Debnath, P. K. Tripathi, and R. Elmasri, "Preference-aware successive poi recommendation with spatial and temporal influence," in *International Conference on Social Informatics*. Springer, 2016, pp. 347–360.

[62] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 811–820.

[63] Y. Zheng and X. Xie, "Learning travel recommendations from user-generated gps traces," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 1, p. 2, 2011.

[64] H. Schütze, "Introduction to information retrieval," in *Proceedings of the international communication of association for computing machinery conference*, 2008.

[65] C.-S. Lee, Y.-C. Chang, and M.-H. Wang, "Ontological recommendation multi-agent for tainan city travel," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6740–6753, 2009.

[66] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[67] H. Yoon, Y. Zheng, X. Xie, and W. Woo, "Smart itinerary recommendation based on user-generated gps trajectories," in *International Conference on Ubiquitous Intelligence and Computing*. Springer, 2010, pp. 19–34.

[68] C. Zhou and X. Meng, "Sts: complex spatio-temporal sequence mining in flickr," in *International Conference on Database Systems for Advanced Applications*. Springer, 2011, pp. 208–223.

[69] X. Lu, C. Wang, J.-M. Yang, Y. Pang, and L. Zhang, "Photo2trip: generating travel routes from geo-tagged photos for trip planning," in *Proceedings of the 18th ACM international conference on Multimedia*. ACM, 2010, pp. 143–152.

[70] A. Gionis, T. Lappas, K. Pelechrinis, and E. Terzi, "Customized tour recommendations in urban areas," in *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM, 2014, pp. 313–322.

[71] C. Zhang, H. Liang, K. Wang, and J. Sun, "Personalized trip recommendation with poi availability and uncertain traveling time," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 911–920.

[72] H.-P. Hsieh, C.-T. Li, and S.-D. Lin, "Exploiting large-scale check-in data to recommend time-sensitive routes," in *Proceedings of the ACM SIGKDD International Workshop on Urban Computing*. ACM, 2012, pp. 55–62.

[73] E. H.-C. Lu, C.-Y. Chen, and V. S. Tseng, "Personalized trip recommendation with multiple constraints by mining user check-in behaviors," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, 2012, pp. 209–218.

[74] G. Upton and I. Cook, *Understanding statistics*. Oxford University Press, 1996.

[75] R. Agrawal, R. Srikant, *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[76] M. A. Khamsi and W. A. Kirk, *An introduction to metric spaces and fixed point theory*. John Wiley & Sons, 2011, vol. 53.

[77] K. H. Lim, J. Chan, C. Leckie, and S. Karunasekera, "Personalized tour recommendation based on user interests and points of interest visit durations," *Under Submission*, 2015.

[78] H. Goodwin, "The haversine in nautical astronomy," in *US Naval Institute Proceedings*, vol. 36, 1910, pp. 735–746.

BIOGRAPHICAL STATEMENT

Madhuri Debnath was born in Narsingdi, Bangladesh. She received her Bachelor degree in Computer Science and Engineering from University of Dhaka, Bangladesh, in 2007. In 2008, she joined as a software engineer in Cention AB, Dhaka, Bangladesh. She joined as a Ph.D. student in the University of Texas at Arlington in 2012 under the supervision of Professor Ramez Elmasri. Her major research interests include data mining, machine learning, spatio-temporal data analysis.