VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI - 590018

"Scalable Daily Human Behavioral Pattern Mining from Multivariate Temporal Data"

Thesis submitted in partial fulfillment of the curriculum prescribed for the award of the degree of Bachelor of Engineering in Computer Science & Engineering by

1CR14CS068	Kumar Ashwani
1CR14CS176	Rahul Kumar Raushan
1CR14CS025	Ashutosh Kumar

Under the Guidance of

Ms. Mariet S Furtado
Assistant professor
Department of CSE, CMRIT, Bengaluru

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
#132, AECS LAYOUT, IT PARK ROAD, BENGALURU - 560037

2017-18

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI - 590018

Certificate

This is to certify that the project entitled **"Scalable Daily Human Behavioral Pattern Mining from Multivariate Temporal Data "** is a bonafide work carried out by **Kumar Ashwani** bearing **USN:1CR14CS068**, **Rahul Kumar Raushan** bearing **USN:1CR14CS176** and **Ashutosh Kumar** bearing **USN:1CR14CS025** in partial fulfillment of the award of the degree of Bachelor of Engineering in Computer Science & Engineering of Visvesvaraya Technological University, Belgaum, during the year 2017-18. It is certified that all corrections / suggestions indicated during reviews have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the Bachelor of Engineering Degree.

Signature of Guide	Signature of HoD	Signature of Principal
Ms. Mariet S Furtado	**Dr. Jhansi Rani P**	**Dr. Sanjay Jain**
Assistant Professor	Professor & Head	Principal
Department of CSE	Department of CSE	CMRIT,
CMRIT, Bengaluru - 37	CMRIT, Bengaluru - 37	Bengaluru - 37

External Viva

	Name of the Examiners	Institution	Signature with Date
1.			
2.			

Acknowledgement

We take this opportunity to thank all of those who have generously helped us to give a proper shape to our work and complete our B.E. project successfully. A successful project is fruitful culmination efforts by many people, some directly involved and some others indirectly,by providing support and encouragement.

We would like to thank Dr. SANJAY JAIN , Principal , CMRIT ,for providing excellent academic environment in the college.

We would like to express our gratitude towards Dr. JHANSI RANI P , Professor and HOD , Dept of CSE , CMRIT , who provided guidance and gave valuable suggestions regarding the project.

We consider it a privilege and honour to express our sincere gratitude to our Internal Guide Ms. Mariet S Furtado , Assistant Professor , Department of Computer Science and Engineering , CMRIT , for her valuable guidance throughout the tenure of this project work.

Kumar Ashwani
Rahul K Raushan
Ashutosh Kumar

Table of Contents

List of Figures

Abstract

This work introduces a set of scalable algorithms to identify patterns of human daily behaviors. These patterns are extracted from multivariate temporal data that have been collected from smartphones. We have exploited sensors that are available on these devices, and have identified frequent behavioral patterns with a temporal granularity, which has been inspired by the way individuals segment time into events. These patterns are helpful to both end-users and third parties who provide services based on this information. We have demonstrated our approach on two real-world datasets and showed that our pattern identification algorithms are scalable. This scalability makes analysis on resource constrained and small devices such as smartwatches feasible. Traditional data analysis systems are usually operated in a remote system outside the device. This is largely due to the lack of scalability originating from software and hardware restrictions of mobile/wearable devices. By analyzing the data on the device, the user has the control over the data, i.e., privacy, and the network costs will also be removed.

Chapter 1

Preamble

1.1 Introduction

The proliferation of smartphones and, more recently, wearable devices such as fitness trackers and smart watches equipped with sensors, has led to a significant expansion of possibilities to study human behavior. Computing and networking capabilities of these sensor-embedded devices makes them appropriate tools for observing and collecting useful contextual information (mobile sensing). For instance, mobile health, which benefits from mobile sensing, offers the possibility of a shift from treatment to prevention in medical care systems. Unlike wearable devices, which are still quite new in the market, the smartphone platform has benefited from a significant amount of scientific work ranging from public transport navigation to well-being. Both wearable devices and smartphones are very capable of sensing and collecting the basic patterns of human behavior through contextual information.While simple human behaviors are predictable, at least in aggregate, traditional approaches for detecting human behavioral patterns (which are not digital) are often difficult. However, the advent of mobile devices enables researchers to identify human behavior to an extent that was not previously possible.On one hand, this information collection paradigm should be moved from simple data collection tools to intelligent systems with cognition capabilities . On the other hand, there is still a lack of wide acceptance of mobile sensing applications in real-world settings.

There are several reasons for this mismatch of capability and acceptance. First is the resource limitation and lack of accuracy in the collected contextual data, especially with regard to the battery life. The size of sensors that are dealing with radio frequency, i.e. bluetooth, WiFi and GPS, affects the quality of their data (smaller devices have less accurate data). The next reason, which has been noted but has not been widely explored, is the proximity of the smartphone to users. Smartwatches and wearables are body-mounted and thus the proximity problem has been resolved in

those devices, but they still suffer from a lack of accuracy .The third reason is operating system restrictions of mobile devices, which removes background services when the CPU is under a heavy load in order to preserve the battery life.As a result, there is no ideal data collection approach that can sense and record individuals' information 24/7 with no data loss. The uncertainty of these data objects is a major challenge that limits the applications that can benefit from them. Mobile Data Mining The goal of mobile data mining is to provide advanced techniques for the analysis and monitoring of critical data from mobile devices. Mobile data mining has to face with the typical issues of a distributed data mining environment , with in addition technological constraints such as low-bandwidth networks , reduced storage space, limited battery power, slower processors, and small screens to visualize the results. The mobile data mining field may include several application scenarios in which a mobile device can play the role of data producer ,data analyzer ,client of remote data miners ,or a combination of them. More specifically, we can envision three basic scenarios for mobile data mining: The mobile device is used as terminal for ubiquitous access to a remote server that provides some datamining services.In this scenario,the server analyzes data stored in a local or distributed database,andsends the results of the data mining task to the mobile device for its visualization.The system we describe in this chapter is based on this approach. Data generated in a mobile context are gathered through a mobile device and sentina stream to a remote server to be stored in to a local database. Data can be periodically analyzed by using specific datamining algorithms and the results used for making decisions about a given purpose.

1.1.1 Temporal Granularity

A temporal granularity can be intuitively described as a sequence of time granules, each one consisting of a set of time instants. A granule can be composed of a single instant, a set of contiguous instants (time-interval), or even a set of non-contiguous instants. A temporal granularity can be used to specify the temporal qualification of a set of data, similar to its use in the temporal qualification of statements in natural languages. For example, in a relational database, the timestamp associated with an attribute value or a tuple may be interpreted as associating that data with one or more granules of a given temporal granularity (e.g., one or more days). As opposed to using instants from a system-specific time domain, the use of user-defined granularities enables both more compact representations and temporal qualifications at different levels of abstraction. Temporal granularities include very common ones like hours, days, weeks, months and years as well as the evolution and specialization of these granularities for specific contexts or applications: trading days, banking days, academic semesters, etc.. Unlike digital systems, human understanding of time is not

precise. Our daily behaviors occur in time intervals. For instance, a person does not arrive at work every day at exactly the same time or eat lunch at exactly the same time every day. A time interval always exists for routine behaviors, even if it is only a small break, e.g., five minutes. This is also true for precise time scheduled tasks, such as a meeting. Therefore, it is essential to have flexibility in temporal analysis. This work is implemented on this dynamic of human behavior by introducing a simple but novel human-centric temporal granularity method. This algorithm use this temporal granularity instead of the original timestamp. Therefore, it should not be categorized as a time series approach.

1.1.2 Multivariate Data Analysis

Multivariate Data Analysis refers to any statistical technique used to analyze data that arises from more than one variable. This essentially models reality where each situation, product, or decision involves more than a single variable. The information age has resulted in masses of data in every field. Despite the quantum of data available, the ability to obtain a clear picture of what is going on and make intelligent decisions is a challenge.Multivariate statistics concerns understanding the different aims and background of each of the different forms of multivariate analysis, and how they relate to each other. The practical implementation of multivariate statistics to a particular problem may involve several types of univariate and multivariate analyses in order to understand the relationships between variables and their relevance to the actual problem being studied. When available information is stored in database tables containing rows and columns, Multivariate Analysis can be used to process the information in a meaningful fashion.

1.1.3 Scalability and Sensor Independence

A salient advantage of this approach is its scalability. It is light- weight and can be integrated into small devices with limited computing capabilities, e.g., wearables. Moreover, not with standing its temporal dependency, it does not consider the type of the underlying sensor data. In this model heterogeneous sensor data is converted into three tuples, which includes sensor name, data and discrete time. Such sensor independency makes the algorithm capable of running in settings that include temporal multivariate data, independent from any specific sensor. Furthermore, employing a combination of sensors rather than focusing on specific sensors, and using temporal granularity instead of exact timestamps, allows us to mitigate uncertainty by ignoring sensor data that is not available, and focusing instead on the available data.

The main goal of this work is to efficiently create a profile,For the given time stamped activities of the user, assuming they are occurring in a routine, which summarizes frequent behavioural patterns of a user. The final output of the project will be of the form of a triplet with percentage of confidence of a user doing a particular activity at a given timestamp. This will be achieved through implementation of the novel concept of temporal granularity for timestamp absolution. Finally this work will aim to reduce the execution time for creating user's frequent behavior profile by using a set of underlying scalable algorithms.

Chapter 2

Literature Survey

2.1 Mobile Data

Several experiments have been undertaken that have utilized large-scale smartphone data collection. One of the first studies that has benefited from the use of smartphones, and has resulted in the formation of a dataset, was Reality Mining [29]. This approach relied on a customized version of an early smartphone, the Nokia N6600. Next to the Reality Mining dataset, the same group introduced Social fMRI [14], which collected context sensing data and subjective input from users (e.g., Facebook activities) plus purchasing information, from 150 participants. Another well-known experiment is the Lausanne Data Collection Campaign [17], which uses another early version of a smartphone, the Nokia N95. It contains smartphone data of about 170 participants. As such, these efforts have (i) collected user data from the device (user-centric) as opposed from the network and (ii) provided some information about the method of this data collection.

Market deployment for smartphone data collection has recently gotten the attention of the community [2]. As a result, a new category of user experiment is emerging, which is based on market deployed data collection. An advantage of this approach is that these data collection experiments have been conducted in real-world settings and have benefited from a large number of users. For instance, Device Analyzer [2] has conducted the largest market-based mobile data collection, such as hardware settings, from approximately 23,000 Android devices. Since their focus is on large-scale data collection, they face challenges in scalability, consistency and privacy. Ferreira et al. [11] and Henze et al. [15] have both deployed their application into the market and describe their lessons learned for market deployment. For instance, they mention a lack of control over users, validity of their findings among lab settings and the users ranked impact on the installation of their application.

Similarly, our studies also rely on utilizing the phones of our users. As a result,

we need to take into account restrictions and challenges that each phone brand poses, such as differences in operating systems. Although we have not undertaken our experiment through market deployment, we do have the advantage of collecting participant feedback through interviews. Having the chance to receive and analyze such feedback has enabled us to identify prominent challenges, such as multivariate reflection and manual intervention, which have not been uncovered in mass market deployment or previous research efforts. This advantage makes our study different than market deployment studies, which do not have a chance of interviewing users.

The most similar approach to our work is provided by Blanke et al. [5]. They describe lessons learned while collecting location data in large scales. For instance, they suggest best practices on marketing a data collection smartphone application with incentivizing elements such as friend finder and unlocking badges. Nevertheless, our approach is more holistic and not focused only on location data. shows different sensor types that have been used for each device and study in our studies.

2.2 Human behaviour with temporal granularity

The behavior of humans (and other organisms or even mechanisms) falls within a range with some behavior being common, some unusual, some acceptable, and some beyond acceptable limits. Behavior in this general sense should not be mistaken with social behavior, which is a more advanced social action, specifically directed at other people. The acceptability of behavior depends heavily upon social norms and is regulated by various means of social control. Understanding human behaviour has attracted a significant number of researchers, and much work has been devoted to modeling human behaviour at different temporal resolutions, from pose estimation and activity recognition to long term behaviour pattern learning. In particular, spatial temporal based [16] or shape based [32] human action recognition algorithms have been developed for the understanding of human motion and activity. Multimodal sensors in ambient intelligence environments [18] or in mobile phones [25] have been integrated to discover patterns in diverse spatial and temporal scales, from a specific activity such as cooking to a persons daily life over a timespan of months. Other groups have tried to recognize affective and social signals in order to create anticipatory interfaces. A survey can be found in [31]. This work learns comprehensive behaviour patterns, and can incorporate the activity recognition or multi modal results to enrich the model. Contextual information has also been extensively used in the recognition of human poses [26] and pattern discovery [22]. Understanding the pattern of human activities is also beneficial for activities recognition. It has been shown that prior knowledge about the structure of human activities provides constraints in the learning process and better

As has been stated previously, this work tries to digitally map timestamps for human activities onto human temporal perception. The term temporal granularity has been introduced by [34], and is different from temporal abstraction. It is notable that temporal abstraction is the process of converting high-dimensional timestamp data to low-level qualitative descriptions of time [27] and has been introduced by [35]. Temporal granularity specifies the temporal qualification of a set of data, similar to its use in the temporal qualification of statements in natural languages. The temporal granularity models that are being used for human behavior analysis being reviewed. There is a limited number of works that consider how to apply temporal granularity to human behavioral data. One of the earlier works, [33], proposes a method to analyze human activities in the office via a probabilistic representation for inferring temporal granularity. The goal is not to infer temporal granularity, but we benefit from this concept to mine patterns of human behavior. The work most similar to ours is [24], which focuses on mining users' daily location patterns via trajectory mining and defines the temporal granularity as a day. As has been stated previously, [13] is another approach for identifying daily behavior and tries to match the daily location of users to application they use. They converted a day into two segments and model application usage in each segment.

2.3 Scalability

Existing works that support a mobile data mining [29] have offered very promising results. However, these studies employ specific hardware, which is known for data quality among users [17], or they analyze data offline outside the device [12]. There is lack of scalable data mining methods that can handle the uncertainty. In this work, we introduce scalable algorithm that utilize a variety of sensors, e.g., WiFi, location, etc. that are available on the device. By leveraging collected multivariate temporal data our algorithm can identify frequent human behavioral patterns (FBP) with a time estimation (temporal granularity), similar to the human perception of time. This algorithm has been tested with their scalability, on two real-world dataset, and two small devices, i.e., a smartphone and smartwatch.

Chapter 3

Requirements

3.1 Functional Requirements

1) Sensor data- The system has to make sure that data from the sensors are readily available the system should allow that if single device sensor is dominating other sensors, it should be removed the system should allow that if particular sensor is down due to any reason, all fields connected to it, should be removed.

2) Location Estimation- The system should appropriately locate user in 5 decimals of latitude/longitude position. The system should be able to classify from the sensor data for a user to be moving, rest or tilted.

3) Cleaning Data-The system must be able to make sure that no incorrect-data exist.

4) Front-end- The json file generated by the java code should be first approved .The system must be able to give quick.

5) Temporal Granularity(Time Precision)- The system should be able to work on various precision values for temporal granularity namely 5hp,hhp,hp,thp etc. The system should be able to define TGs based on common daily time scheduling Also system must be able to use rounding algorithm to convert times based on the given precision.

6) Group Creation-For a very large dataset the system must be able to create group of similar entities with changed threshold values, i.e. system should work for different kind of thresholds.

7) Profile Creation-The system should be able to clearly identify similar entities occuring for given data. Once such group is made, system must be able to filter all these groups to get profiles.

3.2 Non Functional Requirements

1. Scalability and portability - The system shall be able to scaled on to any amount of data with the particular sensor's information . For all the values of theta, lambda, window size etc. system should be able to produce accurate results .The system should be able to run on variety of hardware and softwares.

2. Maintainability -The system should be easy to maintain .There should be a clear separation between the interface and the backend code.

3. Performance- Since the system is very much free from computational complexities ,the results produced by it should be highly accurate and precise than other algos such as apriori,fp growth etc.

4. Reliability- Since the system will be checked for various threshold values for profile and group creation,it should be reliable to provide results to a high level of accuracy.

5. Exception handling- Exceptions should be effectively handled if they occur.

3.3 Hardware Requirements

- Intel Pentium 4 or AMD Athlon 64 processor or above
- 2GB of RAM (6GB recommended)
- 500 Gb of hard-disk space (according to the size of dataset)
- 2.8GHz speed

3.4 Software Requirements

- Eclipse neon or above

- Java jdk Java SE 5 Update 21
- Jre 1.8.0121
- Json Visualiser tool
- Ubiqlog Dataset obtained from uci machine learning repository.

Chapter 4

System Architecture

The system architecture fig 4.1 includes 3 main components:

- Data Preprocessing
- Temporal Granularity Calculation
- Group and Profile Calculation

4.1 Problem Statement

We live in a spatio-temporal world and all our behaviors occur in a specific location and time. Therefore, to digitally quantify human behavior the target system should sense both time and location. Human behavior is composed of many daily activities that are distinctive and recurring. Here, these types of activities have been called frequent behavioral patterns. The problem is defined as follows: Given timestamped activities of the user, assuming they are occurring in a routine, the goal is to efficiently create a profile, which summarizes frequent behavioral patterns of a user.

4.2 Data Processing

The dataset that is used is a human-centric lifelogging dataset UbiqLog.It has mobile sensor logs of each day for different users stored in separate .txt files. The data in the data set may contain data in different formats. It may also include certain unambiguous data, unrecognized symbols, etc. It is important that the data be in one standard format and free from all ambiguities, to extract desired results from it. The data collected from the mobile sensors is very large in volume. It may not be necessary to process the data from all the sensors depending on the application. A two-stage filtering is provided in this work to filter based on sensor and sensor parameters.

4.3 Temporal Granularity Calculation

TABLE I

Notations and Their Descriptions

Notation	Description
e	entity, is a tuple of $< A, D, T >$ that presents a fine-grained information unit
T	time interval of the entity based on temporal granularity, e.g., 16:00-15:00, 12:25-12:30
A	attribute name of an entity.
D	value of an entity. In this model per sensor only one data element will be used.
g	a set of similar entities (inside a window) that repeat in a consecutive days.
θ	minimum required number of entities between the same time intervals of two or more days.
$count(g)$	count the number of g appearances among all days (for each person).
λ	minimum number of repeats for a group to consider this group in the profile.
$profile$	a set of similar groups that have been repeated more than λ times.

Figure 4.1: Notations and their Descriptions

Humans do not perceive time in and of itself, but rather, perceive changes or events in time. To be able to model human behavior, a precise machine timestamp should be transferred to a format like the way humans perceive time. In a more technical sense, humans perceive events in relation to both location and time. All existing mobile and wearable devices can record information with a timestamp. To simulate the human perception of time, Temporal Granularity will be used. Setting the Temporal Granularity also depends on the target application. Here an attempt is made to make a Temporal Granularity for the daily human behaviors. For instance, every evening a user may make a phone call. However, it is unlikely that she/he will call every day exactly at 5:00 pm. The user could call one day at 5:15 PM and another day at 4:53 PM. Thus, we define Temporal Granularities based on common daily time scheduling, and a use rounding algorithm to convert times based on the given precision.

4.4 Group and Profile Creation

The definitions of entity, group and profiles help to understand the concept of group and profile creation. Entity e, is assumed to be a fine-grained unit of human behavior

and consists of a tuple of three e = $< A; D; T >$. Each entity contains a timestamp (time interval), T, attribute name, A, and attribute value (data)D.

Group g, is a collection of similar entities, for a specific time interval, in a set of consecutive days. Therefore, g = $e1; e2; ...; ek$, eg. In simple terms, if the number of entities in a specific time interval are greater or equal than , then they will be collected in a set and this set is being called group. Tc is a time interval that is constant among all entities of a group. In other words, groups are FBPs and the notation of a group is as follows: Group = $Ve : ei(t) = Tc$;

Profile, is characterized by a set of repeated similar groups g which have been identified more than or equal times,i.e., Profile =g1; g2; . . .; gk. We can formalize profile as: Profile = $ifcountofg$ Groups are created by matching similar entities. Here a concept of sliding window is used. A sliding window of ws days is set. The entities of the days in this window are matched. If an entity occurs on all days in the window, then it is set as a group. The data is divided into different sets based on the number of days in the sliding window and groups are found. All the groups that are found are saved. The user profile is created by matching similar groups in different windows. The groups in different windows are matched. If a group is found in more than 1 window its confidence is increased. The resultant user profile has various groups with their corresponding confidence.

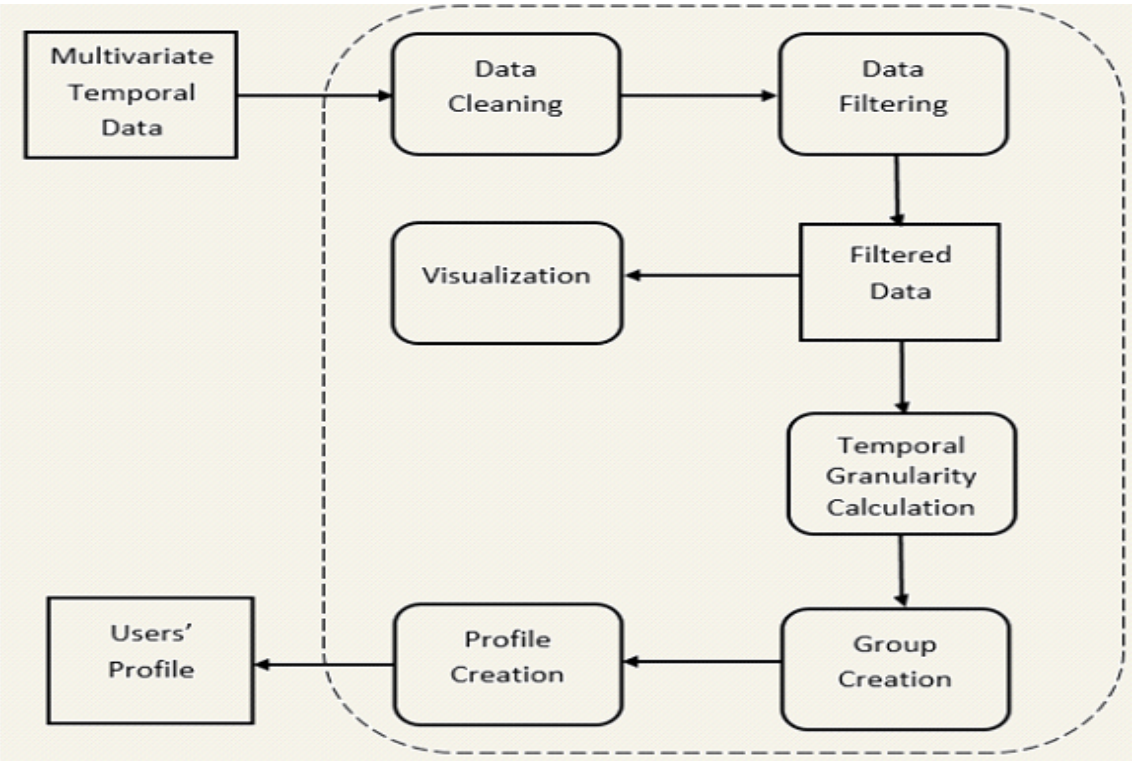

Figure 4.2: System Architecture

Chapter 5

Performance Analysis

The first step in the experimental evaluation is the creation of a ground truth dataset that can help in estimating the accuracy of algorithms. The accuracy of the FBP identification algorithms is evaluated based on (i) different segments of the day, (ii) different TGs. The following experiments describe the next phase of our evaluation and demonstrate the utility and efficiency of our algorithms: First, to demonstrate the scalability of the FBP detection algorithm by analyzing the impact of window size and grouping on the execution time. The FBP approach is lightweight enough to be used in wearable and mobile devices. Next, the execution time of our approach and compare it with other algorithms on both mobile phones and smartwatches is noted

5.1 Accuracy Analysis

5.1.1 Ground Truth Analysis

To evaluate the accuracy and quality of the identified FBPs, the authors had created a ground truth dataset, which is composed of more than 5,000 identified entities, from five users. It contains randomly identified FBP data that has been labeled by the users as either true or false. The number of identified entities in the profile objects are different among users, but each user has labeled about 1,000 entities that belong to him/her self. Users were asked to label if they agree (true) with each of their identifiedentities in their profile or do not agree (false).

5.1.2 Accuracy of Identified FBPs

The labels were carefully examined the accuracy of our algorithms using three temporal segments of the day: 0:00-07:59 (0-8), 08:00-15:59 (8-16) and 16:00-23:59 (1624) and different TGs. Figure 7.2 reports about the accuracy of the identified FBPs, based on labels, with different TGs and not using a TG at all (baseline). The results

of this analysis show that FBP identification accuracy is influenced by different values of TG and the segmentation of the day.Figure 7.2 shows that identifying FBPs using an hour as the TG improves the accuracy of the FBP identification, compared to other TGs. In other words, one hour TG has the highest accuracy among other TGs. Nevertheless, choosing 15, 30, 90 and 120 as TG performs almost the same or slightly lower than 60 but better than 5. The inaccuracy of five minutes is because this TG is too precise for an application to model human behavior. It can be concluded that most routine human behaviors that can be identified from a smartphone have a one hour approximation. Our other evaluation uses the same users to annotate the results delivered by similar algorithms, and compares the accuracy of our FBP algorithm with Apriori, FP-Growth, MTK and estDec+.Figure 7.3 shows the labeling results of users. With no TG (baseline) or low (5) and high (120) TG, other methods perform better than FBP. For the rest of the TGs, FBP outperforms other methods. This accuracy originates in the use of and . Other algorithms are very useful for the general problem domain. Unlike Apriori, both MTK and estDec+ apply another level of filtering such as considering top frequent itemsets by MTK or recent ones by estDec+. Therefore, baseline algorithms that do not filter have superior accuracy.

5.2 Scalability

The algorithms must be capable of being integrated into small devices, which have restricted computational resources compared to desktop computers. To demonstrate that the algorithms are lightweight, the execution time is measured, which is the representation of scalability. Moreover, the execution time of FBP algorithm among Apriori and FP-Growth as baseline algorithms and MTK and estDec+ as state-of-the-art algorithms is evaluated. Apriori is the well-known algorithm for frequent itemset mining. It is not the fastest or most resource efficient but we have chosen it as a baseline algorithm. FP-Growth is scalable and is known as a baseline of the fastest frequent itemset mining algorithms. Furthermore, MTK and estDec+ both alsohave been used as state-of-the-art algorithms that are scalable and fast.

5.2.1 Sliding Window Impact on the Execution Time

Execution time is directly correlated to scalability and scalability is a major contribution of this work. It has been achieved through (i) the adoption of a sliding window and (ii) the reduction of the number of comparisons via utilizing a group based comparison. To demonstrate scalability, first the execution time performance of the FBP algorithm with different window sizes for 60 days is analyzed. This time frame has

been utilized as it covers a significant period so that the capability of the FBP algorithm can be fully tested. Dealing with many days is an important requirement in Lifelogging systems, which use small devices. The baseline here is not using the window, and it compares each day with all the other days.

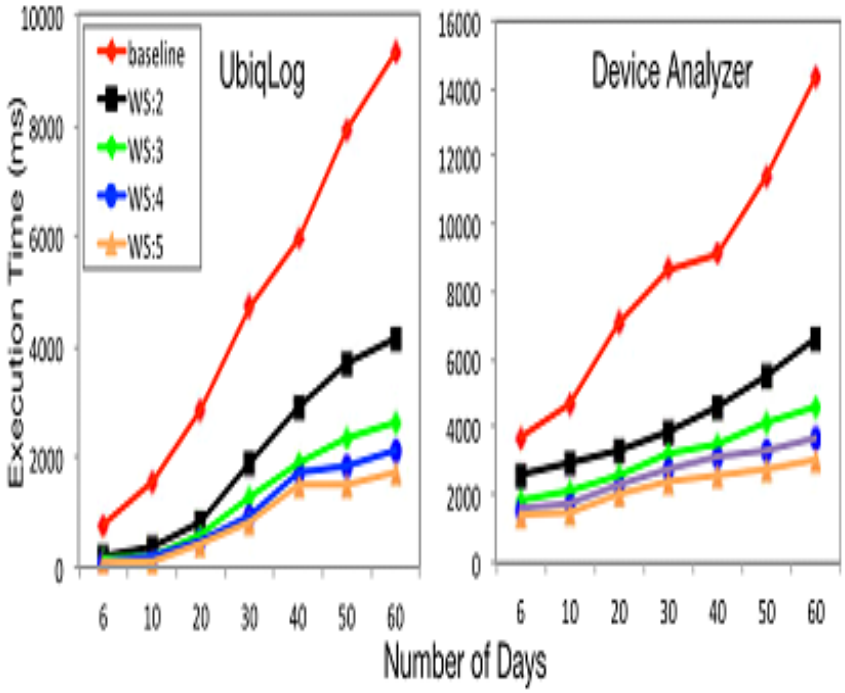

Figure 5.1: The effect of window size on the execution time performance

5.2.2 Comparison with Other Algorithms

Figure shows the execution time of running FBP in comparison to other algorithms on the smartphone, within the described settings. For more than six days of analysis, FBP execution time performance is faster than other algorithms. This has been highlighted especially as the number of days increases, the FBP execution timedoes not change significantly and stays at a near constant value. It has even outperformed state-of-the-art algorithms, which are known to be fast and scalable algorithms that operates on limited memory. However, for a smaller amount of data, FBP does not perform better than MTK or estDec+.

These evaluations demonstrate the scalability of our algorithms, while preserving accuracy. From a technical perspective, this superiority is because of: (i) using a sliding window that filters most of the irrelevant entities, and thus reduces comparisons significantly. (ii) theta and lambda that propose two layers of hierarchical filtering and result in both improving accuracy and reduction in the search space.

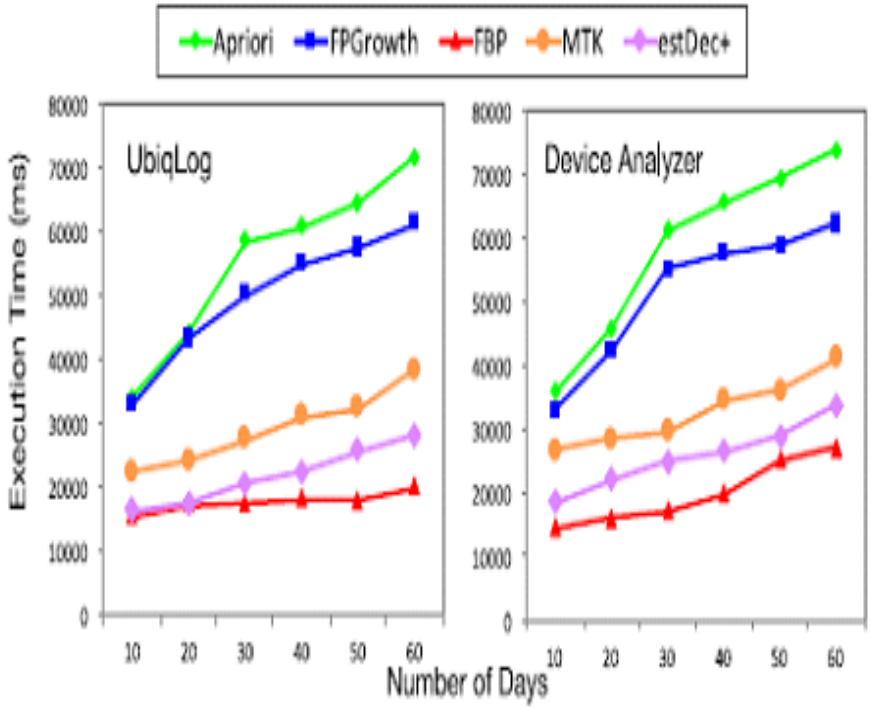

Figure 5.2: Execution time on Different Algorithms

Chapter 6

Implementation

Source CodeImplementation
Source code
Config:
Constants.java

```
package com.ubiqlog.analysis.group.config;
public class Constants {
/**
* This class contains all constants.
*/
public static String DIRECTORY_NAME =
"C:/Users/Kumar_Ashwani/Desktop/Project/New_folder";
        public static String RESTRUCTURED_DIREC
        TORY_NAME = DIRECTORY_NAME + "/Restructured";
        public static String ORDERED_DIRECTORY_
        NAME = DIRECTORY_NAME + "/Ordered";
        public static String FILTERED_DIRECTORY
        _NAME = DIRECTORY_NAME + "/Filtered" ;
        public static String CSV_DIRECTORY_NAME
         = DIRECTORY_NAME + "/CSVs" ;
        public static String SORTED_CSV_DIRECTOR
        Y_NAME = DIRECTORY_NAME + "/SortedCSVs" ;
        public static String FBP_REPORT= DIRECTO
        RY_NAME + "/FBP_REPORT" ;
        // if an app is repeated more than thresh
        old per day, it has been deleted from
        result set.
        public static int APP_REPEAT_THRESHOLD = 25;
```

```
public static String WIFI = "WiFi";
public static String APPLICATION =
"Application";
public static String ACTIVITY = "Activity";
public static String BLUETOOTH = "Bluetooth";
public static String CALL = "Call";
public static String SMS = "SMS";
public static String LOCATION =
"Location";
//Thursday, November 13, 2014 11:31:29
 AM Central European Standard Time ...
  used by WiFi and Bluetooth??
public static final String TIME_TEMPLA
TE_0 = "\\w+,␣\\w+␣\\d{1,2}"
//Monday, 25 November 2013 21:55:55 Ira
n Standard Time .... used by WiFi and
Bluetooth
public static final String TIME_TEMPLATE
_1 = "\\w+,␣\\d{1,2}␣\\w+␣\\d{4}␣\\d{1,2}"
//Tuesday, December 31, 2013 11:59:58 PM
Iran Standard Time      .... used by WiFi and
 Bluetooth
public static final String TIME_TEMPLATE_
2 = "\\w+,␣\\w+␣\\d{1,2},␣\\d{4}␣\\d{1,2}"
//Wednesday, November 13, 2013 9:05:35 PM
 GMT+00:00      .... used by WiFi and Bluetooth
public static final String TIME_TEMPLATE_3
= "\\w+,␣\\w+␣\\d{1,2},␣\\d{4}␣\\d{1,2}"

//12-31-2013 23:29:35 ... used by Call,
SMS, Application, Activity and Location
public static final String TIME_TEMPLATE_
4 = "\\d{1,2}-\\d{1,2}-\\d{4}␣\\d{1,2}"
//Fri 6 Dec 2013 15:24:58  ... used by Wifi
public static final String TIME_TEMPLATE_5
 = "\\w+␣\\d{1,2}␣\\w+␣\\d{4}␣\\d{1,2}"

// pattern matching to check if valid dura
tion values for call sensor
```

```
        public static final String DURATION_TEMPLA
        TE = "\\d+";
// pattern matching to check if valid latitude
 and longtitude values for location sensor
        public static final String LATITUDE_TEMPLA
        TE = "([-+]?\\d{1,2}([.]\\d+)?)";
        public static final String LONGITUDE_TEMPLATE
         = "([-+]?\\d{1,3}([.]\\d+)?)";

        //---------- For FilterJsonFiles.java-------
        -----
        //include the sensors in the string below,
         whose data you want in your filtered json
          strings for visualization
        public static String SENSORS = "WiFi|Applion";
        //include each sensor's attributes in the s
        tring below, whose data you want in your
        //filtered json strings for visualization
        public static String WIFI_PARAMETERS = "time"
        public static String APP_PARAMETERS = "Start"
        public static String ACTIVITY_PARAMETERS

        public static String BLUETOOTH_PARAMETERS
        public static String CALL_PARAMETERS = "Time";

        public static String LOCATION_PARAMETERS = "time|
           Latitude";
```

Clean:

CleanData.java

```
package com.ubiqlog.analysis.group.clean;
import com.ubiqlog.analysis.group.utils.GlobalCounters;
/**
 * The CleanData class implements an application that
 * cleans the logging data
 */
public class CleanData {
        /**
```

```
        * This is the main method for the the cleaning
          phase.
        * It calls the restructuringJsonFiles() and
        * reorderingJsonFiles() methods that restructure
          and sort
        * the logs.
        */
       public static void main(String[] args) throws
          Exception
       {
              RestructureJsonFiles rs = new
                 RestructureJsonFiles();
              ReorderJsonFiles ro = new ReorderJsonFiles();

              System.out.println("Begin cleaning the data
                 ....");
              //order important - first restructure and
                 then reorder
              System.out.println("Restructuring the data
                 now....");
              rs.restructuringJsonFiles();
              System.out.println("Reordering the data now
                 ....");
              ro.reorderingJsonFiles();
              System.out.println("Completed cleaning the
                 data");
       }
}
```

ReorderFiles.java

```
package com.ubiqlog.analysis.group.clean;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
```

```
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import org.json.JSONException;
import org.json.JSONObject;
import com.ubiqlog.analysis.group.config.Constants;
import com.ubiqlog.analysis.group.utils.GlobalCounters;
import com.ubiqlog.analysis.group.utils.JSONComparator;
/**
 * The ReorderJsonFiles class reorders and the
  sorts the data in each log file
 * in increasing order of time stamp.
*/
public class ReorderJsonFiles {
 /**
          * This method reorders the logs according to the
            timestamp.
          */
        public void reorderingJsonFiles() throws Exception
        {

            if(!restructuredDirectory.exists())
            {
                    System.out.println("Directory does not
                       exist.");
                    System.exit(0);
            }
            else
            {
                try{
                      delete(restructuredDirectory);
                    }catch(IOException e){
                        e.printStackTrace();
                        System.exit(0);
                    }
                }
        }
         /**
```

```
    * This method recursively iterates through the
     files and
    * sorts the logs in it.
    * @param files array of all files that need to
     be sorted
    */
public void showFiles(File[] files) throws Exception
{
String processName = null;
String currentLine = null;
BufferedReader br  = null;
List<String> objectList = null;
JSONObject jsonObject = null;
Map<String, Integer> processCountMap = new HashMa
p<String, Integer>();
for (File file : files)
{
        if (file.isDirectory())
        {

        }
else
        {
            if (file.getName().contains("log_"))
            {
        try
        {
                objectList = new ArrayList<>();
                        br = new BufferedReader(new
                           FileReader
                        (file.getAbsolutePath()));


                 while((currentLine = br.readLine())
                    != null)
                 {
                         try
                         {
```

```
                        jsonObject = new JSONObject(
                            currentLine);
                                } catch (JSONException e) {
                                        continue;
                                }


                                if (jsonObject != null)
                {
                        if (jsonObject.optJSONObject("
                            Application
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣") != null)
                        {
                        JSONObject appObject = jsonObject.
                            optJ
        SONObject("Application");
        if ((processName = appObject.getSt
        ring("ProcessName")) != null)
        {
        // all processes add to proces
        sCountMap to be used in
        // removeApplications method
        if (processCountMap.containsKey
 (processName)){
        int count = processCountMap
        .get(processName) ;
        processCountMap.put(process
        Name, ++count);
        }
        else
        {
        processCountMap.put(process
        Name, 1);
                                        }
                                }
                        }
                objectList.add(currentLine);
                }
```

```
            }

            for (String jObject : objectList)
                    writer.println(jObject);
            writer.close();
           } catch (IOException  e) {
                    throw new Exception("File:"+file.
                       getAbs
                    olutePath(), e);
           }finally {
                   try {
                           if (br != null) br.close();
                   } catch (Exception ex) {
                           ex.printStackTrace();
                   }
                }
               }
           }
       }
   }
   public static void delete(File file)throws
      IOException
   {
           if(file.isDirectory())
           {
                   //directory is empty, then delete it
                   if(file.list().length==0)
                   {
                      file.delete();
                   }
                   else
                   {
                      String files[] = file.list();

                      for (String temp : files)
                      {
                   //construct the file structure
   File fileDelete = new File(file, temp);
                         //recursive delete
```

```
                delete(fileDelete);
                    }
                    //check the directory again, if empty then
                        delete it
                    if(file.list().length==0)
                    {
                            file.delete();
                        }
                    }
            }
            else
            {
                    //if file, then delete it
                    file.delete();
            }
}

                                                }
                                        }
                                }
                    }
            } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            }
            return objectList;
}
}
PrecisionCalculatio.java
package com.ubiqlog.analysis.group.csv;
import java.util.Calendar;
import java.util.TimeZone;
import org.joda.time.DateTime;
import org.joda.time.DateTimeZone;
public class PrecisionCalculation {
/**
* The PrecisionCalculation class will calculate the closest
* HourPrecision(HP), HalfHourPrecision(HHP),
 QuarterHourPrecision(QHP) and5 Minute Precision(5MP).
```

```
   For example, for time 19:38:09
* HP = 20:00, HHP = 19:30, QHP = 14:45 & 5MP = 19:40
*/
//Precision to 0,2,4,6,8,10,12,....20,22 hours
public String getTwoHourPrecision(Calendar cal) {
String hourPrecision = null;
        TimeZone tz = cal.getTimeZone();
        DateTimeZone jodaTz = DateTimeZone.forID(tz.getID());
        DateTime dateTime = new DateTime(cal.getTimeInMillis
            (), jodaTz);
        int hourOfDay = dateTime.getHourOfDay();
        if (hourOfDay != 23)
        {
                        if (hourOfDay%2 == 0)
                                hourPrecision = hourOfDay + "
                                    :00";
                        else
                                hourPrecision = (hourOfDay) +
                                     1 + ":00";
        }
        else
        {
                        hourPrecision = "0:00";
        }

        return hourPrecision;
}
//Precision round-off to 0,1.5,3,4.5,6,7.5,9,....19.5,21,22.5
    hours
public String getOneAndHalfHourPrecision(Calendar cal) {
        String hourPrecision = null;
        TimeZone tz = cal.getTimeZone();
        DateTimeZone jodaTz = DateTimeZone.forID(tz.getID());
        DateTime dateTime = new DateTime(cal.getTimeInMillis
            (), jodaTz);
        int dayinMillis = dateTime.getMillisOfDay();
        for (int i = 0; i < 24; i = i + 3)
        {
                double millis = dayinMillis - (i * 3600000);
```

```
                if (millis > 0 && millis < 10800000)
                {
                        if (millis > 5400000)
                        {
                                if (millis > 8100000)


                                hourPrecision = (i+3)
                                    + ":00";

                                else
                                hourPrecision = (i+1)
                                    + ":30";
                        }
                        else
                        {
                                if (millis > 2700000)
                                hourPrecision = (i+1) + ":30"
                                    ;

                                else
                                hourPrecision = i + ":00";
                        }
                        break;
                }
        }
        return hourPrecision;
}
public String getHourPrecision(Calendar cal) {
        String hourPrecision = null;
        TimeZone tz = cal.getTimeZone();
        DateTimeZone jodaTz = DateTimeZone.forID(tz.getID());
        DateTime dateTime = new DateTime(cal.getTimeInMillis
            (), jodaTz);
        int hourOfDay = dateTime.getHourOfDay();
        if ((dateTime.getMillisOfDay() - (hourOfDay *
            3600000)) > (30*60000))
        {
                if ((hourOfDay + 1) == 24)
```

```
                hourPrecision = "0:00";
                else
                hourPrecision = (hourOfDay + 1) + ":00";
        }
        else
                hourPrecision = hourOfDay + ":00";
        return hourPrecision;
}
public  String getHalfHourPrecision(Calendar cal) {
        String halfHourPrecision = null;
        TimeZone tz = cal.getTimeZone();
        DateTimeZone jodaTz = DateTimeZone.forID(tz.getID());
        DateTime dateTime = new DateTime(cal.getTimeInMillis
           (), jodaTz);
        int dayinMillis = dateTime.getMillisOfDay();
        int hourOfDay = dateTime.getHourOfDay();
        int minsInMillis = dayinMillis - (hourOfDay *
           3600000);
        for (int i = 30 ; i >= 0 ; i = i - 30)
{
        if (minsInMillis > (i * 60000) && minsInMillis < ((i
           +30) * 60000)
                {
                if (minsInMillis > ((i + 15) * 60000))
                {
                if (i == 30)
                {
                                        if ((hourOfDay + 1)
                                           == 24)
                                                halfHourPrecision
                                                   = "0:00";


                                                halfHourPrecision
                                                   = (
                                                hourOfDay
                                                + 1) + "
                                                :00";
                                }
```

```
                                    else
                                    halfHourPrecision = hourOfDay
                                        + ":" + String.format("
                                       %02d", i + 30);
                                                        }
                            else
                                    halfHourPrecision = hourOfDay
                                        + ":" + String.format("
                                       %02d", i);
                            break;
                    }
            }
            return halfHourPrecision;
}
public String getQuarterHourPrecision(Calendar cal) {
            String quarterHourPrecision = null;
            TimeZone tz = cal.getTimeZone();
            DateTimeZone jodaTz = DateTimeZone.forID(tz.getID());
            DateTime dateTime = new DateTime(cal.getTimeInMillis
               (), jodaTz);
            int dayinMillis = dateTime.getMillisOfDay();
            int hourOfDay = dateTime.getHourOfDay();
            int minsInMillis = dayinMillis - (hourOfDay *
               3600000);
            for (int i = 45 ; i >= 0 ; i = i-15)
            {
                    if (minsInMillis > (i * 60000) &&
                       minsInMillis < ((i+15) * 60000))
                    {
                            if (minsInMillis > ((i + 7.5) *
                               60000))
                            {
                                    if (i == 45)
                                    {
                                            if ((hourOfDay + 1)
                                               == 24)
                                                    quarterHourPrecision
                                                       = "0:00";
                                            else
```

```
                                                    quarterHourPrecision
                                                        = (
                                                       hourOfDay
                                                       + 1) + "
                                                       :00";
                                        }
                                        else
                                                quarterHourPrecision
                                                   = hourOfDay + ":"
                                                   + String.format("
                                                   %02d", i + 15);
                                }
                                else
                                        quarterHourPrecision =
                                           hourOfDay + ":" + String.
                                           format("%02d", i);
                                break;
                        }
                }
                return quarterHourPrecision;
}
public String getFiveMinPrecision(Calendar cal)
{
                String fiveMinPrecision = null;
                TimeZone tz = cal.getTimeZone();
                DateTimeZone jodaTz = DateTimeZone.forID(tz.getID());
                DateTime dateTime = new DateTime(cal.getTimeInMillis
                   (), jodaTz);
int dayinMillis = dateTime.getMillisOfDay();
                int hourOfDay = dateTime.getHourOfDay();
                int minsInMillis = dayinMillis - (hourOfDay *
                   3600000);
                for (int i = 55 ; i >= 0 ; i = i-5)
                {
                        if (minsInMillis > (i * 60000) &&
                           minsInMillis < ((i + 5) * 60000))
                        {
                                if (minsInMillis > ((i + 2.5) *
                                   60000))
```

```
                        {
                                if (i == 55)
                                {
                                        if ((hourOfDay + 1)
                                           == 24)
                                                fiveMinPrecision
                                                   = "0:00";
                                        else
                                                fiveMinPrecision
                                                   = (
                                                   hourOfDay
                                                   + 1) + "
                                                   :00";
                                }
                                else
fiveMinPrecision = hourOfDay + ":" + String.format("%02d", i
   +5);
                        }
                        else
                                fiveMinPrecision = hourOfDay
                                   + ":" + String.format("%02
                                   d", i);
                        break;
                }
        }
        return fiveMinPrecision;
}
}
```

SortCSVs.java

```
package com.ubiqlog.analysis.group.csv;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import org.json.JSONException;
```

```
import org.json.JSONObject;
import au.com.bytecode.opencsv.CSVReader;
import au.com.bytecode.opencsv.CSVWriter;
import com.ubiqlog.analysis.group.config.Constants;
import com.ubiqlog.analysis.group.datamining.CSVComparator;
public class SortCSVs {
//separator for CSV file. Comma in our case.
public static final char DELIMITER = ',';
/**
* This is the main method for filtering the data according to
    the
* selected parameters above. It will create json files that
* are used for visualization.
*/
public static void main(String[] args) throws Exception
        {
                File[] files = new File(Constants.
                   CSV_DIRECTORY_NAME).listFiles();
                File filteredDirectory = new File(Constants.
                   SORTED_CSV_DIRECTORY_NAME);
                if (!filteredDirectory.exists())
                        filteredDirectory.mkdir();
                else
                {
                        filteredDirectory.delete();
                        filteredDirectory.mkdir();
                }
                showFiles(files);
        }
/**
* This method recursively iterates through the files and
* calls the createJSON method for each file.
* @param files array of all files that need to be sorted
*/
public static void showFiles(File[] files) throws Exception
        {
                BufferedReader br = null;
                CSVReader reader = null;
                for (File file : files)
```

```
                {
                if (file.isDirectory())
                {
                    System.out.println("Directory: " + file.
                        getName());
                    showFiles(file.listFiles()); // Calls
                        same method again.
                } else
                {
                   System.out.println("File: " + file.
                       getAbsolutePath());
                try {
                List<String[]> recordList = new ArrayList<>()
                    ;
                reader = new CSVReader(new FileReader(file.
                    getAbsolutePath()));
                            String[] currentString;
                int count = 0;
                String[] headerRecord = null;
                while ((currentString = reader.readNext()) !=
                     null)
                {
                if (count == 0)
                 headerRecord = currentString;
                if (count != 0)
                 recordList.add(currentString);
                count++;
                }
                        Collections.sort(recordList, new
                            CSVComparator());
                    recordList.add(0, headerRecord);
String fileName = file.getName();
                                FileWriter fileWriter = new
    FileWriter(Constants.SORTED_CSV_DIRECTORY_NAME + "/" +
    fileName);
CSVWriter csvWriter = new CSVWriter(fileWriter, DELIMITER,
    CSVWriter.DEFAULT_QUOTE_CHARACTER,
                    CSVWriter.NO_ESCAPE_CHARACTER, "\n");
                csvWriter.writeAll(recordList);
```

```
                csvWriter.close();

                } catch (IOException  e)
 {
                        throw new Exception("File:"+file.
                            getAbsolutePath(), e);
                }finally {
                        try {
                                if (br != null) br.close();
                        } catch (Exception ex) {
                        ex.printStackTrace();
                        }
                }
                }
}
}
}
```

Datamining:

GetSimilarEntities

```
package com.ubiqlog.analysis.group.datamining;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.TimeZone;
import au.com.bytecode.opencsv.CSVReader;
import au.com.bytecode.opencsv.CSVWriter;
import com.ubiqlog.analysis.group.config.Constants;
public class GetSimilarEntities {
```

```
//Map that stores unique group name with its contents
private static Map<String, String> groupNameNContentMap = new
    HashMap<>();
//Group Number
private static int groupIndex = 0;
//Start time of the program
private static long startTime = 0;
public static void main(String[] args) throws Exception
{
startTime = System.currentTimeMillis();
        File[] files = new File(Constants.
           SORTED_CSV_DIRECTORY_NAME).listFiles();
        File filteredDirectory = new File(Parameters.
           DATAMINING_CSV_DIRECTORY_NAME);
        if (!filteredDirectory.exists())
                filteredDirectory.mkdir();
        else
                {
                filteredDirectory.delete();
                filteredDirectory.mkdir();
                }
showFiles(files);
}
/**
* This method recursively iterates through the files and
* calls the findSimilarEntities(dayList, windowNumber) method
    for each window.
* @param files array of all files that need to be sorted
*/
public static void showFiles(File[] files) throws Exception
{
        CSVWriter csvWriter = null;
        FileWriter fileWriter = null;
        String fileName = null;
        CSVReader reader = null;
        List<List<String[]>> dayList = new ArrayList<>(
           Parameters.WINDOW_SIZE);
        List<String[]> oldList = new ArrayList<String[]>();
```

```
        Map<String, CSVWriter> writersMap = new HashMap<
            String, CSVWriter>();
for (File file : files)
        {
                if (file.isDirectory())
                {
                    System.out.println("Directory: " + file.
                        getName());
                    File directory = new File(Constants.
                        FILTERED_DIRECTORY_NAME +"/"+ file.
                        getName()+"_filtered");
                    directory.mkdir();
                    showFiles(file.listFiles()); // Calls
                        same method again.
                } else
                {
                    if (file.getName().contains("_" +
                        Parameters.TIME_PRECISION))
                    {
                //reset groupMap and index for new user
                groupNameNContentMap = new HashMap<>();
                groupIndex = 0;
                try {
                                        reader = new
                                            CSVReader(new
                                            FileReader(file.
                                            getAbsolutePath()
                                            ));



                 String[]  currentString;
                 int count = 0;
                 oldList = new ArrayList<String[]>();
                while ((currentString = reader.readNext()) !=
                    null)
                 {
                 if (count != 0)
                oldList.add(currentString);
```

```
                    count++;
                            }
                    int i=0,previousDay = 0, windowNumber = 1;
                    List<String[]> oneDay = null;
                    for (String[] record : oldList)
                    {
if (windowNumber <=(Parameters.NUMBER_OF_DAYS/Parameters.
   WINDOW_SIZE))
                    {
                    if(i <= Parameters.WINDOW_SIZE)
                    {
                    Calendar cal = Calendar.getInstance(TimeZone.
                       getTimeZone("Asia/Tehran"));
                                SimpleDateFormat formatter = new
                       SimpleDateFormat("MM-dd-yyyy HH:mm:ss");
               formatter.setTimeZone(TimeZone.getTimeZone("Asia/
                  Tehran"));
                    try {
cal.setTime(formatter.parse(record[0]));
                                        }
                    catch (ParseException e)
                    {
                            e.printStackTrace();
                            }
                    //ignore if day is Friday (Weekend)
                    int weekDay = cal.get(Calendar.DAY_OF_WEEK);
                    if (weekDay == 6)
                    {
                    continue;
                    }
                    int dayOfMonth = cal.get(Calendar.
                       DAY_OF_MONTH);
                    if (dayOfMonth == previousDay)
                    {
                          oneDay.add(record);
                    }
                    else
                    {
                          if (i != 0)
```

```
                    {
                    dayList.add(oneDay);


                                                }
                    previousDay = dayOfMonth;
                    oneDay = new ArrayList<String[]>();
                    oneDay.add(record);
                    i++;
                    }
                    }
                    else{
                    List<String> groupsList = findSimilarEntities
                       (dayList, windowNumber);
                     List<String[]> groupRecords = new ArrayList<
                        String[]>();
                     List<String[]> windowRecords = new ArrayList
                        <String[]>();
                     String[] groupNameNContent = null;
                     String groupsInWindow = "";
                     String[] arrayOfEntity = null;
                    int groupCount = 0;
for (String group : groupsList)
                                        {
                     groupNameNContent = group.split("@");
                     List<String> singleGroupRecord = new
                        ArrayList<String>();
singleGroupRecord.add(String.valueOf(groupNameNContent[0]));
singleGroupRecord.add(String.valueOf(groupNameNContent[1]));
String[] recordArray = new String[singleGroupRecord.size()];
singleGroupRecord.toArray(recordArray);
groupRecords.add(recordArray);
                     arrayOfEntity = groupNameNContent[1].split("
                        \\|");
                     if (arrayOfEntity.length >= Parameters.
                        ENTITY_THRESHOLD)
                    {
                     groupsInWindow = groupsInWindow +
                        groupNameNContent[0] + "|";
```

```
                                                    groupCount
                                                        ++;
                                                }
                    if (writersMap.containsKey(file.getName().
                        substring(0, 13)+"_group"))
                    csvWriter = writersMap.get(file.getName().
                        substring(0, 13)+"_group");
                                                }
                    else
                    {
fileName = file.getName().substring(0, 13);
fileWriter = new FileWriter(Parameters.
    DATAMINING_CSV_DIRECTORY_NAME + "/"
 + fileName + "_groups.csv");

   csvWriter = new CSVWriter(fileWriter, Parameters.DELIMITER,
        CSVWriter.DEFAULT_QUOTE_CHARACTER, CSVWriter.
       NO_ESCAPE_CHARACTER, "\n");
writersMap.put(file.getName().substring(0, 13)+"_group",
    csvWriter);
                                                }
                    csvWriter.writeAll(groupRecords);
                    csvWriter.flush();
                            }
                    List<String> singleWindowRecord = new
                        ArrayList<String>();
singleWindowRecord.add(String.valueOf("Window "+windowNumber)
    );                      singleWindowRecord.add(String.valueOf(
    groupsInWindow));
singleWindowRecord.add(String.valueOf(groupCount));
                    String[] recordArray = new String[
                        singleWindowRecord.size()];
                    singleWindowRecord.toArray(recordArray);
                    windowRecords.add(recordArray);
                    if (writersMap.containsKey(file.getName().
                        substring(0, 13)+"_window"))
                    {
                    csvWriter = writersMap.get(file.getName().
                        substring(0, 13)+"_window");
```

```
                                        }
                    else
                    {
fileName = file.getName().substring(0, 13);

fileWriter = new FileWriter(Parameters.
   DATAMINING_CSV_DIRECTORY_NAME + "/" +
 fileName + "_windows.csv");
csvWriter = new CSVWriter(fileWriter, Parameters.DELIMITER,
 CSVWriter.DEFAULT_QUOTE_CHARACTER, CSVWriter.
    NO_ESCAPE_CHARACTER, "\n");
writersMap.put(file.getName().substring(0, 13)+"_window",
   csvWriter);
                                }
                    csvWriter.writeAll(windowRecords);
                    csvWriter.flush();
                    i = 1; windowNumber++;
                    dayList = new ArrayList<>(Parameters.
                       WINDOW_SIZE);
                            }
                    }

                    system.out.println("Done");
                    } catch (IOException  e) {
                                throw new Exception("File:"+
                                   file.getAbsolutePath(), e)
                                   ;
                                }finally {
                                try {
                                        if (reader != null)
                                           reader.close();
                                } catch (Exception ex) {
                                        ex.printStackTrace();
                                }
                            }
                       csvWriter.close();
                       }
                    }
                    }
```

```
            for (Entry<String, CSVWriter> writerEntry :
                writersMap.entrySet())
            {
                    writerEntry.getValue().close();

            }
        ProfileCreator profileCreator = new
            ProfileCreator();
        profileCreator.createUserProfile();
        long stopTime = System.currentTimeMillis();
        long elapsedTime = stopTime - startTime;
        System.out.println("Number of Days = " +
            Parameters.NUMBER_OF_DAYS);
        System.out.println("Elapsed Time = " +(
            elapsedTime/1000) + " seconds");
    }
    /**
    * This method finds similar entities in each time
    * slot within a window
    * @param dayList list of list for each day in a
       window
    * @param windowNumber window number
    * @return list containing groups within a window
    */
    private static List<String> findSimilarEntities(List<
       List<String[]>>
     dayList, int windowNumber)
    {
            System.out.println("Window Number = " +
                windowNumber);
            String currentSlot = "00";
            List<String[]> sameSlotList = null;
            List<String> groupsList = new ArrayList<>();
            List<String[]> firstDayRecords = dayList.get
                (0);
            for (String[] record : firstDayRecords)
            {
                    if (!currentSlot.equals(record[2]))
                    {
```

```
                        if (sameSlotList != null)
                        {
                        Iterator<String[]> iter =
                            sameSlotList.iterator();
                        while (iter.hasNext())
                        {
                        String[] sameSlotRecord = iter.next()
                            ;
                        for (int i = 1; i < Parameters.
                            WINDOW_SIZE; i++)
                        {
boolean present = false;
                        Iterator<String[]> iter2 = dayList.
                            get(i).iterator();
                        while(iter2.hasNext())
                        {
                    String[] otherRecords = iter2.next();
if (otherRecords[2].equals(sameSlotRecord[2])&&
    sameSlotRecord[3]
.equals(otherRecords[3])&& sameSlotRecord[4].equals(
    otherRecords[4])){
                    present = true;
                    iter2.remove();
                                        }
                                }
                        if (present == false)
                        {
                        iter.remove();
                        break;
                                }
                        }
                    }
                    String group = ""; String index = null;
                    if (sameSlotList.size() > 0)
                    {
                    //Removing duplicate entities within a group
                    for (String [] entity : sameSlotList)
                    {
                            index=null;
```

```
                        String thisEntity =   entity[2] + "|"
                            + entity[3] + "|" + entity[4];
                        if (!group.contains(thisEntity)){
                                        group = thisEntity;
                                              }

                    for (Entry<String, String> entry :
                       groupNameNContentMap.entrySet())
                            {
                            if (entry.getValue().equals(group))
                            index = entry.getKey().substring(1);
                                        }
                            if (index == null)
                            {
                                    index = String.valueOf(
                                       groupIndex++);
                                    groupNameNContentMap.put("G"+
                                        index, group);
                                        }
                                    groupsList.add("G"+ index + "
                                       @" + group);

                                                  }
                                    }
                            }
                            sameSlotList = new ArrayList<>();
                            sameSlotList.add(record);
                            currentSlot = record[2];
                            }
                            else
                            sameSlotList.add(record);
                    }
             return groupsList;
            }
}
```

ProfileCreator.java

```
package com.ubiqlog.analysis.group.datamining;
import java.io.BufferedReader;
import java.io.File;
```

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import au.com.bytecode.opencsv.CSVReader;
import au.com.bytecode.opencsv.CSVWriter;
public class ProfileCreator {
/**
         * This class will create a user profile for each
            user with the
         * confidence of each activity.
 */
public void createUserProfile()
{
        File[] files = new File(Parameters.
           DATAMINING_CSV_DIRECTORY_NAME).listFiles();
    try {
                showFiles(files);
        } catch (Exception e) {
                e.printStackTrace();
        }
}
public  void showFiles(File[] files) throws Exception
{
        BufferedReader br  = null;
        CSVReader reader  = null;
for (File file : files)
        {
        if (file.isDirectory())
        {
        System.out.println("Directory: " + file.getName());
        showFiles(file.listFiles());
}
 else
                {
```

```
if (file.getName().contains("_window"))
                    {
                try {
                        Map<String, Integer> groupCountMap =
                            new HashMap<>();
reader = new CSVReader(new FileReader(file.getAbsolutePath())
    );
                                String[]  currentString;

                                int windowCount = 0;
                                while ((currentString =
                                    reader.readNext()) != null
                                    )
                                {
                                        windowCount++;
                                        String[] groups =
                                            currentString[1].
                                            split("\\|");
                                        for (String group:
                                            groups)
                                        {
                                        if (group.length() >
                                            0)
                                        {
                                        if (groupCountMap.
                                            containsKey(group)
                                            )
                                        {
                                                int count =
                                                    groupCountMap
                                                    .get(group
                                                    );
                                                groupCountMap
                                                    .put(group
                                                    , ++count)
                                                    ;
                                        }
                                        else
```

```
                    groupCountMap.put(group, 1);
                                    }
                                        }
                                    }
                    List<String[]> groupRecords = new ArrayList<
                        String[]>();
                    for (Entry<String, Integer> entry :
                        groupCountMap.entrySet())
                    {
                    if (entry.getValue() >= Parameters.
                      PROFILE_THRESHOLD)
                    {
                    List<String> singleGroupRecord = new
                        ArrayList<String>();
singleGroupRecord.add(String.valueOf(entry.getKey()));
                    Double confidence = (((double)entry.getValue
                        ())/windowCount) * 100;
                    String conf = setPrecision(confidence, 4);
                    singleGroupRecord.add(String.valueOf(conf + "
                      %"));
                    String[] recordArray = new String[
                        singleGroupRecord.size()];
                    singleGroupRecord.toArray(recordArray);
                    groupRecords.add(recordArray);
                                    }
                                }
String fileName = file.getName().substring(0, 13);

FileWriter fileWriter = new FileWriter(Parameters.
    DATAMINING_CSV_DIRECTORY_NAME + "/" + fileName + "
    _user_profile.csv");
CSVWriter csvWriter = new CSVWriter(fileWriter,Parameters.
    DELIMITER,CSVWriter.DEFAULT_QUOTE_CHARACTER, CSVWriter.
    NO_ESCAPE_CHARACTER, "\n");
                    csvWriter.writeAll(groupRecords);
                    csvWriter.close();
                    } catch (IOException e) {
                    throw new Exception("File:"+file.
                        getAbsolutePath(), e);
```

```
                }finally {
                try {
                if (br != null) br.close();
                } catch (Exception ex) {
                ex.printStackTrace();
                }
                }
                    }
                }
            }
        }
        public static String setPrecision(double nbr, int
           decimal)
        {
            int integer_Part = (int) nbr;
            double float_Part = nbr - integer_Part;
            int floating_point = (int) (Math.pow(10, decimal)
                * float_Part);
            String final_nbr = String.valueOf(integer_Part) +
                "." + String.valueOf(floating_point);
            return final_nbr;
        }
}
```

Parameters.java

```
package com.ubiqlog.analysis.group.datamining;
import com.ubiqlog.analysis.group.config.Constants;
public class Parameters {
/**
         * This class is used to set the input parameters
        */
//Select HP for hour precision, HHP for half hour precision,
//QHP for quarter hour precision, or 5MP for 5 min precision
   below
        public static final String TIME_PRECISION = "HP";
//Path of directory containing
public static String DATAMINING_CSV_DIRECTORY_NAME =
   Constants.DIRECTORY_NAME + "/DataMiningCSVs" ;
//separator for CSV file. Comma in our case.
```

```
        public static final char DELIMITER = ',';
        public static final int NUMBER_OF_DAYS = 40;
//Number of days to be considered as one window
public static final int WINDOW_SIZE = 3;
//theta
        public static final int ENTITY_THRESHOLD = 1;
//lambda
        public static final int PROFILE_THRESHOLD = 1;
}
```

Chapter 7

Experimental Analysis and Results

The first step in the experimental evaluation is the creation of a ground truth dataset that can help in estimating the accuracy of algorithms. The accuracy of the FBP identification algorithms is evaluated based on different segments of the day, different TGs. The following experiments describe the next phase of our evaluation and demonstrate the utility and efficiency of our algorithms First, to demonstrate the scalability of the FBP detection algorithm by analyzing the impact of window size and grouping on the execution time. The FBP approach is lightweight enough to be used in wearable and mobile devices. Next, the execution time of our approach and compare it with other algorithms on both mobile phones and smartwatches is noted.

7.1 Accuracy Analysis

7.1.1 Ground Truth Dataset

To evaluate the accuracy and quality of the identified FBPs, the authors had created a ground truth dataset, which is composed of more than 5,000 identified entities, from five users. It contains randomly identified FBP data that has been labeled by the users as either true or false. The number of identified entities in the profile objects are different among users, but each user has labeled about 1,000 entities that belong to him/her self. Users were asked to label if they agree (true) with each of their identified entities in their profile or do not agree (false).

7.1.2 Accuracy of Identified FBPs

The labels were carefully examined the accuracy of our algorithms using three temporal segments of the day: 0:00-07:59 (0-8), 08:00-15:59 (8-16) and 16:00-23:59 (1624) and different TGs. Table 4 reports about the accuracy of the identified FBPs, based on labels, with different TGs and not using a TG at all (baseline). The results of

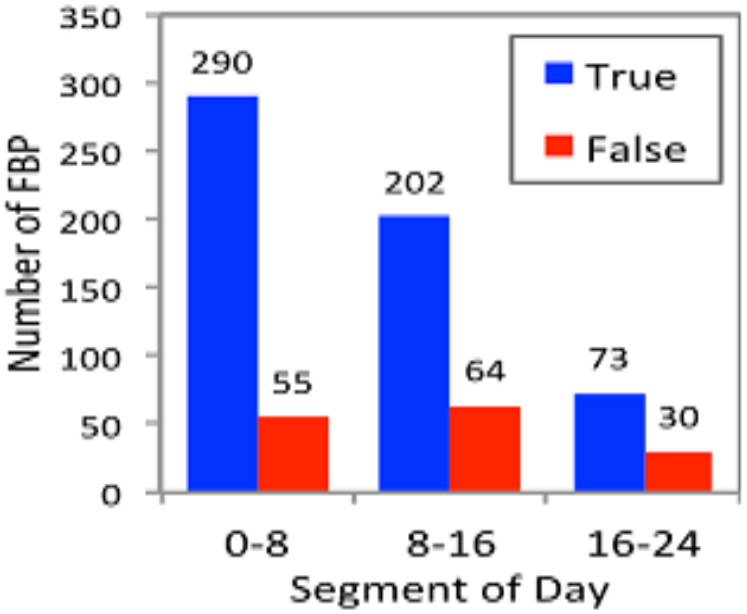

Figure 7.1: Correct and incorrectly labeled FBPs based on time segment for TG = 1Hr.

this analysis show that FBP identification accuracy is influenced by different values of TG and the segmentation of the day. Table 4 shows that identifying FBPs using an hour as the TG improves the accuracy of the FBP identification, compared to other TGs. In other words, one hour TG has the highest accuracy among other TGs. Nevertheless, choosing 15, 30, 90 and 120 as TG performs almost the same or slightly lower than 60 but better than 5. The inaccuracy of five minutes is because this TG is too precise for an application to model human behavior. It can be concluded that most routine human behaviors that can be identified from a smartphone have a one hour approximation.

Our other evaluation uses the same users to annotate the results delivered by similar algorithms, and compares the accuracy of our FBP algorithm with Apriori, FP-Growth, MTK and estDec+. Table 5 shows the labeling results of users. With no TG (baseline) or low (5) and high (120) TG, other methods perform better than FBP. For the rest of the TGs, FBP outperforms other methods. This accuracy originates in the use of and . Other algorithms are very useful for the general problem domain. Unlike Apriori, both MTK and estDec+ apply another level of filtering such as considering top frequent itemsets by MTK or recent ones by estDec+. Therefore, baseline algorithms that do not filter have superior accuracy.

Table 2 :FBP Identification Accuracy with Different Temporal Segments and TG's(0′ is the Baseline)

TG	Temporal Segment			
	0-8	8-16	16-24	All
0′	0.52	0.56	0.46	0.48
5′	0.66	0.62	0.64	0.62
15′	0.79	0.71	0.72	0.65
30′	0.84	0.74	0.70	0.69
60′	**0.84**	**0.76**	**0.71**	**0.77**
90′	0.80	0.75	0.71	0.75
120′	0.81	0.73	0.70	0.75

Figure 7.2: FBP identification accuracy

7.2 Scalability

The algorithms must be capable of being integrated into small devices, which have restricted computational resources compared to desktop computers. To demonstrate that the algorithms are lightweight, the execution time is measured, which is the representation of scalability. Moreover, the execution time of FBP algorithm among Apriori and FP-Growth as baseline algorithms and MTK and estDec+ as state-of-the-art algorithms is evaluated. Apriori is the well-known algorithm for frequent itemset mining. It is not the fastest or most resource efficient but we have chosen it as a baseline algorithm. FP-Growth is scalable and is known as a baseline of the fastest frequent itemset mining algorithms. Furthermore, MTK and estDec+ both alsohave been used as state-of-the-art algorithms that are scalable and fast.

7.2.1 Sliding Window Impact on the Execution Time

Execution time is directly correlated to scalability and scalability is a major contribution of this work. It has been achieved through (i) the adoption of a sliding window

Table 3: Accuracy of Our FBP Algorithm in Comparison to Apriori, FPGrowth, MTK and estDec+ Algorithms, with Different TGs

TG	Apri	FP-Growth	MTK	estDec+	FBP
0′	0.55	0.54	0.55	0.51	0.48
5′	0.66	0.62	0.62	0.67	0.63
15′	0.58	0.60	0.63	0.62	0.65
30′	0.65	0.63	0.66	0.67	0.69
60′	0.69	0.71	0.68	0.73	0.77
90′	0.75	0.71	0.70	0.71	0.75
120′	0.78	0.78	0.77	0.78	0.75

Figure 7.3: Table accuracy of our FBP

and (ii) the reduction of the number of comparisons via utilizing a group based comparison. To demonstrate scalability, first the execution time performance of the FBP algorithm with different window sizes for 60 days is analyzed. This time frame has been utilized as it covers a significant period so that the capability of the FBP algorithm can be fully tested. Dealing with many days is an important requirement in Lifelogging systems, which use small devices. The baseline here is not using the window, and it compares each day with all the other days.

7.2.2 Comparison with Other Algorithms

The execution time of running FBP in comparison to other algorithms on the smartphone, within the described settings. For more than six days of analysis, FBP execution time performance is faster than other algorithms. This has been highlighted especially as the number of days increases, the FBP execution timedoes not change significantly and stays at a near constant value. It has even outperformed state-of-the-art algorithms, which are known to be fast and scalable algorithms that operates on limited memory. However, for a smaller amount of data, FBP does not perform better than MTK or estDec+.

7.3 Snapshot

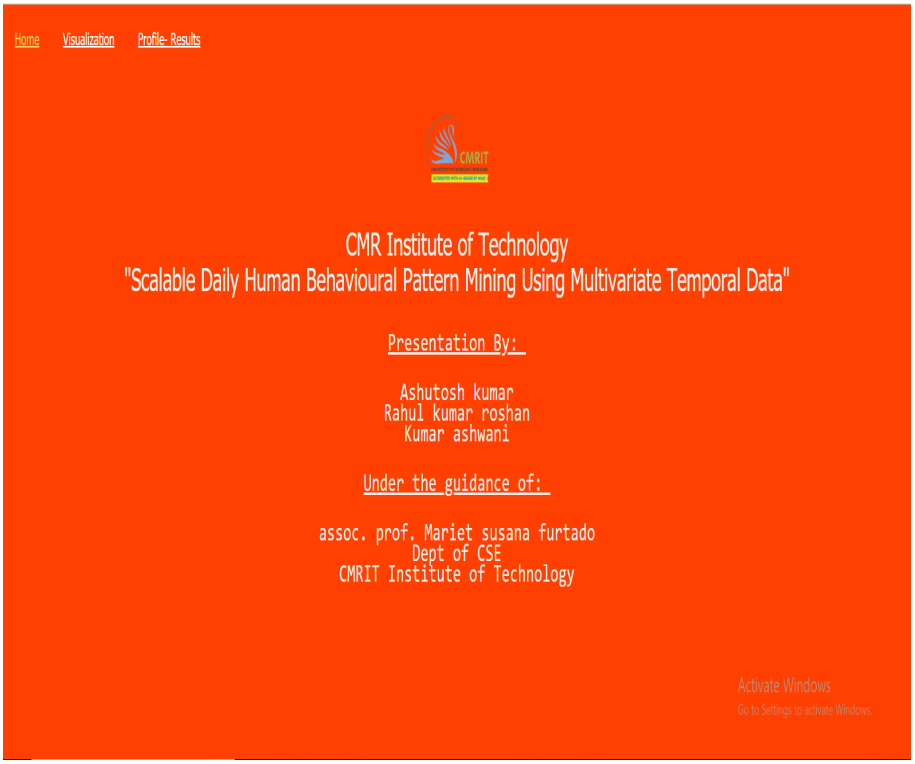

Figure 7.4: Front End

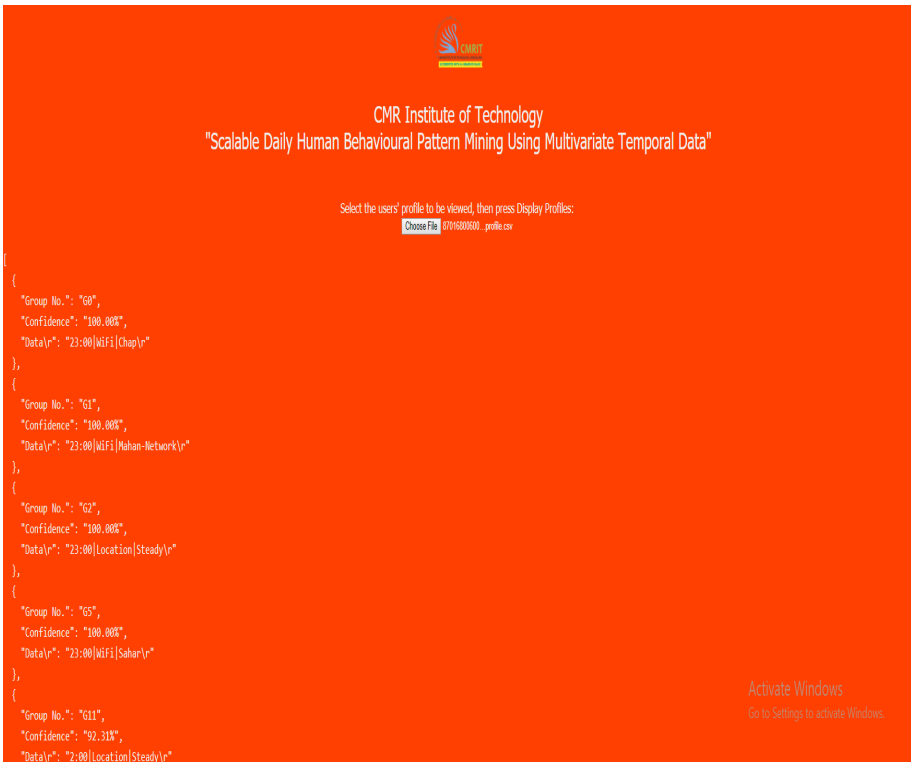

Figure 7.5: User profile as set of traids

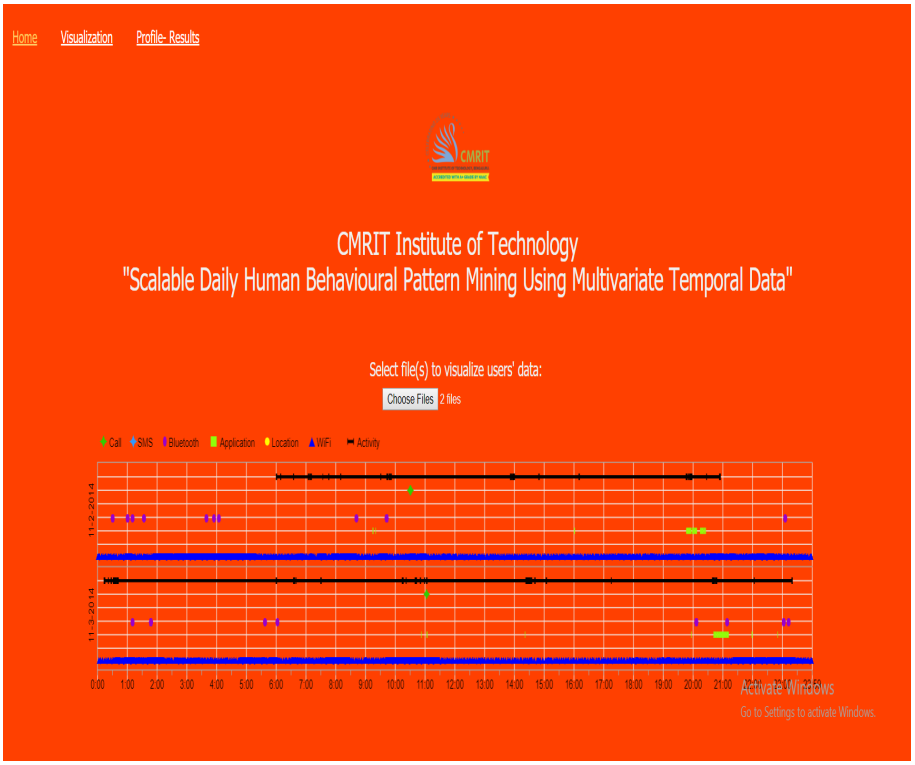

Figure 7.6: User Data represented on visualization tool

Chapter 8

Conclusion And Future Scope

8.1 Conclusion

A major contribution of this research is a generic mobile data mining system. It is generic because of its multisensory support and application independence. The secondary contribution is frequent item set mining algorithms and their sub components such as analyzing the temporal aspect of human behavior. Therefore, two categories of related works have been studied: mobile data mining efforts that focus on device data collection, frequent item set mining algorithms. In this paper, a scalable approach for daily behavioral pattern mining from multiple sensor information is proposed. This work has been benefited from two real-world datasets and users who use different smartphone brands. A novel temporal granularity transformation algorithm that makes changes on timestamps to mirror the human perception of time is used. The frequent behavioral pattern detection approach is generic and not dependent on a single source of information; therefore, we have reduced the risk of uncertainty by relying on a combination of information sources to identify frequent behavioral patterns. Furthermore, the approach is lightweight enough that it can be run on small devices, such as smartwatches, and thus reduces the network and privacy cost of sending data to the cloud. Results of the experimental evaluation shows the algorithm outperforms the baseline and two state-of-the-art algorithms in both execution time and accuracy. Moreover, converting raw timestamps to temporal granularities increase the accuracy of the FBP identification, which is influenced by different values of temporal granularity, the segment of the day and the sensor type. These findings assist the system in identifying the appropriate run time and sensor impact of the behavioral pattern identification.

8.2 Future Scope

- Health Care Rapid growth of the aged population has caused an immense increase in the demand for healthcare services. Generally, the elderly is more prone to health problems compared to other age groups. With effective monitoring and alarm systems, the adverse effects of unpredictable events such as sudden illnesses, falls, and so on can be ameliorated to some extent. Recently, advances in wearable and sensor technologies have improved the prospects of these service systems for assisting elderly people. These technologies are categorized into three types: indoor positioning, activity recognition and real time vital sign monitoring. Positioning is the process of accurate localization and is particularly important for elderly people so that they can be found in a timely manner. Activity recognition not only helps ensure that sudden events (e.g., falls) will raise alarms but also functions as a feasible way to guide peoples activities so that they avoid dangerous behaviors. Since most elderly people suffer from age-related problems, some vital signs that can be monitored comfortably and continuously via existing techniques are also summarized.

- Transportation Another use-case can be transportation optimization. The use of intelligent transport systems has a positive impact on the environment. As vehicle move in an optimized manner and travelers are advised to use the most suitable transport means to reach their intended destination, the travel times and congestions decrease, the fuel consumption also decrease and the emissions are reduced. The main goal is to improve travel times, fuel consumption and hence emissions. Another objective of such system is to try to shift transport behavior from private vehicles to public transport, further improving the overall efficiency of urban transport system, by identifying the routine behaviors of the users using their cellphone data. For example, to arrive at the train station on time, a system can learn the routine commuter patterns of a user, and notify them about the appropriate time for leaving toward the station. On the other hand, the scalability enables on-device and online analysis, and therefore removes both the network cost.

- Personal Assistant Currently, personal assistant systems, run on smartphones and use natural language interfaces. However, these systems rely mostly on the web for finding information. Mobile and wearable devices can collect an enormous amount of contextual personal data such as sleep and physical activities. These information objects and their applications are known as quantified-self, mobile health or personal informatics, and they can be used to provide a deeper insight into our behavior. Existing personal assistant systems do not support all

types of quantified-self queries. A user study was undertaken, to analyze a set of textual questions/queries that users have used to search their quantified-self or mobile health data. Through analyzing these questions, we have constructed a light-weight natural language based query interface - including a text parser algorithm and a user interface - to process the users queries that have been used for searching quantified-self information collected from users smartphones. This query interface has been designed to operate on small devices, i.e. smart-watches, as well as augmenting the personal assistant systems by allowing them to process end users natural language queries about their quantified-self data

References

[1] Reza Rawassizadeh, Elaheh Momeni, Chelsea Dobbins, Joobin Gharibshah, and Michael Pazzani, Scalable Daily Human Behavioral Pattern Mining from Multivariate Temporal Data, IEEE Transaction on knowledge and data Engineering, VOL. 28, NO. 11, NOVEMBER 2016

[2] Wagner DT, Rice A, Beresford AR, Device analyzer: Largescale Mobile Data Collection. ACM SIGMETRICS Perform. Eval. Rev, 41, 5356. 2014.

[3] Mankodiya K, de la Torre F, Zhang A, Ryan N,Ton TG, Ganhdi R,Jain S. SPARK: Personalized Parkinson Disease Interventions through Synergy between a Smartphone and a Smartwatch. In Design, User Experience, and Usability. User Experience Design for Everyday Life Applications and Services; Springer International Publishing: Crete, pp. 103114. Greece, 2014

[4] Begum N,Keogh E, Rare Time Series Motif Discovery from Unbounded Streams. Proc. VLDB Endow., 8, 149160, 2014

[5] Blanke U, Troster G, Franke T, Lukowicz P, Capturing crowd dynamics at large scale events using participatory GPS-localization. In Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing, pp. 17. Singapore, 2124 April 2014;

[6] Mcnaney R., Vines J, Roggen D, Balaam M, Zhang P, Poliakov I, Olivier P, Exploring the Acceptability of Google Glass as an Everyday Assistive Device for People with Parkinson, in Proceedings of CHI2014

[7] Hedegaard S, and Simonsen J, Extracting usability and user experience information from online user reviews. (pp. 20892098). In Proceedings of CHI2013

[8] Iacob C, Veerappa V, Harrison R. , What are you complaining about?: a study of online reviews of mobile applications. In Proceedings of the 27th International BCS Human Computer Interaction Conference (p. 6),2013.

[9] Berlin E, Laerhoven KV, Detecting Leisure Activities with Dense Motif Discovery. In Proceedings of the 2012 ACM Conference on Ubiquitous Computing, Pittsburgh, PA, USA; pp. 250259, 58 September 2012

[10] Dong Y, Hoover A, Scisco J, Muth E, A new method for measuring meal intake in humans via automated wrist motion tracking. Appl. Psychophysiol. Biofeedback , 37, 205215, 2012

[11] Ferreira D, Kostakos V, Dey AK Lessons learned from large-scale user studies: Using android market as a source of data. Int. J. Mob. Hum. Comput. , 4, 2843. Interact. 2012

[12] S. Nath, ACE: Exploiting correlation for energy-efcient and continuous context sensing, in Proc. 10th Int. Conf. Mobile Syst. Appl. Serv pp. 2942, 2012.

[13] H. Ma, H. Cao, Q. Yang, E. Chen, and J. Tian, A habit mining approach for discovering similar mobile users, in Proc. 21st Int. Conf. World Wide Web pp. 231240., 2012,

[14] Aharony N, Pan W, Ip C.; Khayal I, Pentland A, Social fMRI: Investigating and Shaping Social Mechanisms in the Real World. Pervasive Mob. Comput, 7, 643659. 2011

[15] Henze N, Poppinga B, Pielot M, Schinke T, Boll S, My app is an experiment: Experience from user studies in mobile app stores. Int. J. Mob. Hum. Comput. Interact. 2011

[16] A. Oikonomopoulos, I. Patras, and M. Pantic. Spatiotemporal localization and categorization of human actions in unsegmented image sequences. IEEE transactions on Image Processing, 20(4):112640, Apr. 2011.

[17] Kiukkonen N, Blom J, Dousse O, Gatica-Perez D, Laurila J, Towards Rich Mobile Phone Datasets: Lausanne Data Collection Campaign. In Proceedings of the International Conference on Pervasive Services, Berlin, Germany, 1315 July 2010.

[18] A. Aztiria, A. Izaguirre, and J. C. Augusto. Learning patterns in ambient intelligence environments: a survey. Artificial Intelligence Review, 34(1):3551, May 2010.

[19] C. Wu, A. Khalili, and H. Aghajan. Multiview activity recognition in smart homes with spatio-temporal features. In ACM/IEEE ICDSC, 2010.

[20] N. Kiukkonen, J. Blom, O. Dousse, G.-P. Daniel, and J. Laurila, Towards rich mobile phone Datasets: Lausanne data collection campaign, Proc. ACM Int. Conf. Pervasive Services, 2010.

[21] Mancini C, Thomas K, Rogers Y, Price B A, Jedrzejczyk L, Bandara AK., Joinson AN, and Nuseibeh B. From Spaces to Places: Emerging Contexts in Mobile Privacy, 110. In: UbiComp2009

[22] A. Aztiria, A. Izaguirre, R. Basagoiti, and J. Augusto. Learning about preferences and common behaviours of the user in an intelligent environment. In Behaviour Monitoring and Interpretation-BMI, pages 289315. IOS Press, 2009.

[23] F. De La Torre, J. Hodgins, J. Montano, and S. Valcarcel. Detailed Human Data Acquisition of Kitchen Activities : the CMU-Multimodal Activity Database (CMU-MMAC). In Workshop on Developing Shared Home Behavior Datasets to Advance HCI and Ubiquitous Computing Research, 2009.

[24] Y. Ye, Y. Zheng, Y. Chen, J. Feng, and X. Xie. Mining Individual Life Pattern Based on Location History. In MDM '2009

[25] K. Farrahi and D. Gatica-Perez. What Did You Do Today? Discovering Daily Routines from Large-Scale Mobile Data. In ACM Multimedia, 2008.

[26] L.-P. Morency, C. Sidner, C. Lee, and T. Darrell. Head gestures for perceptual interfaces: The role of context in improving recognition. AI, 171(8-9):568585, June 2007.

[27] M. Stacey and C. McGregor. Temporal Abstraction in Intelligent Clinical Data Analysis: A Survey. Artificial Intelligence in Medicine '2007,.

[28] Poh MZ, Swenson NC, Picard RW, A Wearable Sensor for Unobtrusive, Long-term Assessment of Electrodermal Activity. IEEE Trans. Biomed. Eng. , 57, 12431252,2007.

[29] Eagle N, Pentland A, Reality mining: Sensing Complex Social Systems. Pers. Ubiquitous Comput. , 10, 255268,2006.

[30] Maurer U, Rowe A, Smailagic A, Siewiorek DP, eWatch: A wearable sensor and notification platform. In Proceedings of the IEEE Workshop on Wearable and Implantable Body Sensor Networks, Cambridge, MA, USA, 35 April 2006.

[31] M. Pantic, A. Pentland, A. Nijholt, and T. Huang. Human computing and machine understanding of human behavior: A survey. In ICMI, 2006.

[32] A. Veeraraghavan, A. K. Roy-Chowdhury, and R. Chellappa. Matching shape sequences in video with applications in human movement analysis. IEEE PAMI, 27(12):1896909, Dec. 2005.

[33] N. Oliver, E. Horvitz, and A. Garg. Layered Representations for Human Activity Recognition. In ICMI '2002.

[34] C. Bettini, S. Jajodia, and S. Wang. Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer, 2000.

[35] Y. Shahar. A framework for knowledge-based temporal abstraction. Artificial intelligence '1997.