METHODS FOR LOCATION PREDICTION OF MOBILE PHONE USERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İLKCAN KELEŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JULY 2014

Approval of the thesis:

METHODS FOR LOCATION PREDICTION OF MOBILE PHONE USERS

submitted by **İLKCAN KELEŞ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _______________

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _______________

Prof. Dr. İsmail Hakkı Toroslu
Supervisor, **Computer Engineering Department, METU** _______________

Examining Committee Members:

Assoc. Prof. Dr. Tolga Can
Computer Engineering Department, METU _______________

Prof. Dr. İsmail Hakkı Toroslu
Computer Engineering Department, METU _______________

Assoc. Prof. Dr. Pınar Karagöz
Computer Engineering Department, METU _______________

Assist. Prof. Dr. İsmail Sengör Altıngövde
Computer Engineering Department, METU _______________

Dr. Güven Fidan
AGMLAB _______________

Date: _______________

ABSTRACT

METHODS FOR LOCATION PREDICTION OF MOBILE PHONE USERS

Keleş, İlkcan
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. İsmail Hakkı Toroslu

July 2014, 69 pages

Due to the increasing use of mobile phones and their increasing capabilities, huge amount of usage and location data can be collected. Location prediction is an important task for mobile phone operators and smart city administrations to provide better services and recommendations. In this work, we have investigated several approaches for location prediction problem including clustering, classification and sequential pattern mining. We propose a sequence mining based approach for location prediction of mobile phone users as an appropriate solution. More specifically, we present a modified Apriori-based sequence mining algorithm for next location prediction, which involves use of multiple support thresholds for different levels of pattern generation. The proposed algorithm involves a new support definition as well. We have analyzed the behaviour of the algorithm under the change of threshold through experimental evaluation and the experiments indicate improvement in comparison to conventional Apriori-based algorithm.

Keywords: Location Prediction, Mobile Phone Users, Sequential Pattern Mining, AprioriAll Algorithm

ÖZ

CEP TELEFONU KULLANICILARININ KONUM TAHMİNİ İÇİN YÖNTEMLER

Keleş, İlkcan
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. İsmail Hakkı Toroslu

Temmuz 2014 , 69 sayfa

Cep telefonu kullanımının ve telefonların yeteneklerinin artması, cep telefonu kullanımıyla ve yer bilgisi ile ilgili büyük miktarda verinin toplanmasına sebep olmaktadır. Bu sebeple cep telefonu kullanıcılarının konum tahminini yapabilmek operatörler ve yetkililer açısından önemli bir noktaya gelmektedir. Konum tahmininin doğru yapılması, kullanıcılara daha iyi servis sağlamalarına ve daha doğru öneriler yapabilmelerine imkan sağlayabilir.Bu çalışmada, cep telefonu kullanıcılarının konum tahminine yönelik bir çok yaklaşım (kümeleme, sınıflandırma, sıralı örüntü madenciliği) denenmektedir. Ayrıca, kullanıcıların bir sonraki konumunu tahmin etmeye yönelik Apriori tabanlı bir algoritma uygun çözüm olarak önerilmektedir. Önerilen algoritmada destek tanımı değiştirilmektedir ve her seviye için ayrı destek sınırı belirlenmektedir. Bilinen AprioriAll algoritmasından farklı bir diğer yanı da yeni bir destek tanımı içermesidir. Bu çalışmada önerdiğimiz algoritmanın parametrelere bağlı olarak değişimi de incelenmektedir. Yapılan deneyler, algoritmamızın bilinen AprioriAll algoritmasına göre ilerleme kaydettiğini göstermektedir.

Anahtar Kelimeler: Yer Tahmini, Cep Telefonu Kullanıcıları, Sıralı Örüntü Madenciliği, AprioriAll Algoritması

To my family and friends

ACKNOWLEDGMENTS

I would like to express my sincere appreciations to my supervisor Professor İsmail Hakkı Toroslu for his constant guidance, support and friendship during my master studies. His vision and encouragement are the factors which make me finish my studies. I would like to thank to Associate Professor Pınar Karagöz for her support during these three years. Working with them improved my vision and my thoughts about the academic world.

This thesis would not have been possible without the help, support and patience of my colleague Mert Özer. I am really happy to work with him in this project. This research project was supported by Ministry of Science, Industry and Technology of Turkey with project number 01256.STZ.2012-1 and title "Predicting Mobile Phone Users' Movement Profiles".

I am also grateful for my friends in A210 and in the department: Özgür Kaya, Hüsnü Yıldız, Merve Aydınlılar, Gökdeniz Karadağ, Selma Süloğlu, Itır Önal. Without their friendship, help and encouragement, it would be impossible to finish this work.

I am very grateful for my friends Uğur Dönmez, Taylan Işıkdemir, Osman Kaya, Aybike Avşaroğlu, Alper Güngör, Fırat Akyıldız, Volkan Çetin and Önder Kalacı. They provided invaluable support for me. Even though some of them are away from Ankara, they were always with me.

I would also like to express my thankfulness to Kemal Sunal. Without his wonderful films, I would not feel motivated to finish my studies.

I would like to express my deepest gratitude to my fiancee Emel Şentürk. She was with me throughout my university life and I felt her support, encouragement, patience and love all the time.

Lastly, sincerest thanks to my family for supporting and believing in me all the way through my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASMAMS	Apriori-based Sequence Mining Algorithm with Multiple Support Thresholds
GPS	Global Positioning System
CDR	Call Detail Record
GPRS	General Packet Radio Service
HCMM	Home-cell Community-Based Mobility Model
TM	Mobility Prediction based on Transition Matrix
PMM	Periodic Mobility Model
ICTM	Inhomogeneous Continuous Time Markov Model
HPY	Hierarchical Pitman-Yor Language Model
SSE	Error Sum of Squares

CHAPTER 1

INTRODUCTION

Today, mobile phones are considered as an essential tool of daily life and almost everyone has a mobile phone on average. Intensive amounts of basic usage data including base station information, call records and GPS records are stored by large-scale mobile phone operators. This data gives companies ability to build their users' daily movement models and helps them predict the current location of their users. Using these models and prediction schemes, companies can arrange more effective advertisement strategies and the city administrators can determine mass people movement patterns around the city.

We have explored clustering and classification algorithms for the location prediction problem. Although at first these algorithms seem like proper solutions to the problem, they did not give successful results. This is due to the fact that our data have a same percentage value of 76%.

One of the common approaches for location prediction systems usually makes use of sequential pattern mining methods. These methods usually follow two steps; extract frequent sequence patterns and predict accordingly. They mostly use Apriori-based algorithms for the phase of extracting sequence patterns. Our approach embraces the idea to use historical movement patterns for current location prediction of a person. However, due to the huge size of the Call Detail Record (CDR) data, it is impossible to use all historical data to model daily behaviour of people. Rather than using whole patterns contained in the CDR data implicitly, we need to devise a control mechanism over the elimination of sequence patterns. However, the flexibility of control that conventional Apriori-based sequence mining algorithms provide is not fully adequate due to the necessity of the balance between the accuracy and space cost in our work. It is a well known fact that when minimum support gets lower, the number of patterns extracted increases, thereby size of prediction sets for next location of a person gets larger and accuracy of predictions eventually increases. However, the larger number of patterns causes larger space cost. Conventional technique to prevent space cost explosion is to increase minimum support value. Yet this time, it decreases the number of frequent patterns and the size of the prediction sets dramatically, and this causes to miss some of the interesting patterns in data. To prevent possible space explosion and not to miss valuable information in data, we propose a modified version of Apriori-based sequence mining algorithm, that works with level-based multiple minimum support values instead of a global one. To the best of our knowledge, this is the first work which uses different minimum support values at different levels of pruning phases of the conventional algorithm.

Normally the number of levels for Apriori-based sequence mining algorithms is not pre-configured. However, in our case, we consider a predefined number of previous steps to predict the next one. Therefore, we can set the number of levels in Apriori search tree. Moreover, we slightly change the definition of minimum support in our context, which will be explained in more detail in the following chapters. We have experimentally compared the performance of the proposed method involving multiple support thresholds in comparison to that of conventional Apriori-based algorithm that uses only a single minimum support value. The experiments indicate that the proposed approach is more effective to decrease the prediction count and memory requirement.

The outline of this thesis are organised as follows:

- In Chapter 2, a survey of sequential pattern mining algorithms and location prediction algorithms is presented.
- In Chapter 3, clustering, classification and sequential pattern mining algorithms used in this work are presented in a detailed way.
- In Chapter 4, the data is explained and the problem definition is given.
- In Chapter 5, the information about how basic clustering, classification and

sequential pattern mining methods are applied to the CDR data is presented.

- Chapter 6 contains the details of the proposed method. This chapter presents the phases of the proposed algorithm and includes an example run too.
- In Chapter 7, the results of the experiments are presented and discussed. In addition, a comparison with AprioriAll is included in this chapter.
- Chapter 8 concludes the thesis and provides future work which can enhance the accuracy of the proposed algorithm.

CHAPTER 2

RELATED WORK

In this chapter, we provide information about the current sequential pattern mining and location prediction algorithms. We also summarize various aspects of each technique.

2.1 Sequential Pattern Mining

Sequential pattern mining problem is first introduced by Agrawal and Srikant in [1] and they also give an algorithm to solve this problem which is named as AprioriAll. This algorithm finds the frequent sequences in a transaction database whose support is over the given minimum support. They defined support as the ratio of the number of customers having this sequence to the number of all customer sequences.

In recent years, a variety of modifications to the minimum support concept in Apriori-based algorithms have been proposed in [10, 17, 2, 25, 28, 12, 26] for both association rule mining and sequential pattern mining. In [10], Han and Fu propose a new approach over the conventional Apriori Algorithm that works with association rules at multiple concept levels rather than single concept level. They first create hierarchical model for items, and iterate levels in a top-down manner by defining a unique minimum support value for each level. In [17], Liu et al., propose a novel technique to the rare item problem which is named as MSapriori algorithm. They define a modified concept of minimum support which is a minimum item support having different threshold values for different items. In [2], Baralis et al. used association rule mining to classify the instances. In order to handle classification and to extract similar number of rules for all classes including the classes with a small number of instances, they propose a class based support definition. They also propose that decreasing the minimum support by some fixed constant at each pruning level makes more sense rather than using fixed global minimum item support. Their experiments show that Apriori-based classification gives better accuracy than the other classification methods in some cases.

Since then, several variations of Liu et al.'s work have been proposed [12, 26]. In [12], Hu et al. propose a new tree based approach in order to save information about sequences and to allow users enhance the model without scanning the database again. In [25], Toroslu and Kantarcioglu introduce a new support parameter named as repetition support to discover cyclically repeated patterns. The new parameter helps them to discover more useful patterns by reducing the number of patterns searched. In [28], Ying et al. propose a location prediction system using both conventional support concept and a score value that is related with semantic trajectory pattern in the candidate elimination phase. The score depends on the geographic interpretation of the movement.

Most of the multiple minimum support concepts are based on the rare item set problem. To the best of our knowledge, this is the first work which uses different minimum support values at the different levels of pruning phases of conventional algorithm.

2.2 Location Prediction

In recent years, a variety of location prediction schemes which use different approaches have been proposed. In [21], Rajagopal et al. propose a location prediction algorithm which first calculates cell-to-cell transition probabilities of a mobile user using the previous movements of the user, then saves this information to a matrix. Based on this matrix, next location is predicted using k, which is a user defined parameter, most probable cells which are the neighbors of the current cell. This algorithm is called Mobility Prediction based on Transition Matrix (TM).

In [27], Yavas et al. presented an AprioriAll based sequential pattern mining algorithm to find the frequent sequences and to predict the next location of the user. They use a single support value as in the conventional AprioriAll algorithm. They added a new parameter named maximum number of predictions which is used to limit the size of the prediction set. They compared their algorithm's results with TM. Although it is an applicable algorithm in location prediction domain, since it only uses a single support value, it can not predict a user's next location if the sequence of the user is not frequent in the data. This is a problem which is similar to the rare itemset problem.

In [24], Thanh et al. make use of Gaussian distribution and expectation maximization algorithm to learn the model parameters. Then, mobility patterns, where each is characterized by a common trajectory and a cell residence time model, are used for making predictions. The results of the experiments show that their algorithm gives better results than TM since they use the time component when building the model. If the data does not contain cell residence times, this algorithm can not be used.

In [4], Boldrini et al. propose a method which uses social based mobility model. In other words, they predict the next location of a user with respect to his/her social network information. They define the mobility in three aspects. First of all, users will visit more locations if his network i.e friends, family etc., is distributed all over the network. Secondly, users have a tendency towards visiting just a few locations where they spend most of their time. Lastly, users prefer shorter paths to the longer paths. Home-cell is defined as the cell within which the users' social circle mainly moves. They first started with a purely social prediction scheme, then according to the properties of the mobility, they incrementally improved the model which at last becomes Home-cell Community-Based Mobility Model (HCMM). The algorithm relies on the other data sources than mobile phone traces.

In [28], Ying et al. enhance current location prediction schemes with a different approach which uses semantic tags. They introduce the semantic trajectory notion which includes a sequence of locations as in the other methods. The difference is that semantic trajectory includes semantic labels along with the location information. In their work, they propose SemanPredict algorithm which works with the notion they introduced. They also define a new similarity method called as semantic trajectory similarity to calculate the similarity between semantic trajectories. In addition, they propose index structures in order to represent these trajectories efficiently. We could not use SemanPredict algorithm since our data does not include semantic label information about the base stations.

In [5], Cho et al. propose a method which uses social information and tries to model the mobility as in [4]. They build the periodic and social mobility model on the observation that mobile phone users show strong periodic movements throughout the day. They think that people tend to alternate between primary and secondary locations on weekdays and between home and locations which are results of social networks on weekends. They model user locations as a mixture of Gaussians centered at primary and secondary locations. The model is named as Periodic Mobility Model (PMM). As HCMM, this algorithm relies on the other data sources than mobile phone traces.

In [29], Zhang et al. further improve the prediction accuracy of [4] and [5] by amplifying the effect of social network information in location prediction. They propose a novel scheme to improve the prediction accuracy by changing the weight of social interplay revealed in the data. Their proposed scheme checks whether the social information has any effect on the prediction. If that is the case, the scheme uses PMM, otherwise it uses HCMM.

In [8], Gao et al. use both spatial and temporal data to predict users' locations. In order to use temporal information in prediction phase, they assume that a person's daily visit frequencies follow Gaussian distribution. They also break down the temporal information as hour and day. They managed to predict user locations with an accuracy rate of 50% with their proposed method which is named as HPY-Prior Hour Day Model.

In [9], Gidofalvi et al., propose a statistical method which uses both spatial and temporal GPS data and predicts the location and the time of user movements. To build this model, they firstly extract grid-based statistical information with respect to cell stay time information of users. Then using this information, they extract frequently visited regions for the users. Lastly, from the sequence of regions, they continuously estimate parameters for an inhomogeneous continuous time Markov Model (ICTM). In the prediction phase, ICTM predicts when and where information with respect to the given region sequence. According to the experiments, they can predict the departure time to be in an interval of 45 minutes centered at the actual departure time and the next region correctly in 67% of the cases.

In [6], Montjoye et al. propose a method using both voronoi diagrams involving base stations and spatial and temporal properties of users' movement data to find the minimum number of points adequate to uniquely identify individuals. They state that if the location is specified hourly with enough resolution, four spatio temporal points are enough to uniquely identify 95% of the individuals.

Our approach in this work is inspired from [27] and [6], and our aim is to use these ideas combined with a new support definition to predict location changes of mobile users.

CHAPTER 3

BACKGROUND

In this chapter, we provide information about the algorithms used in this work. We divide the chapter in three main parts namely clustering, classification and sequential pattern mining. In each part, we first define the concepts in a general manner, then we explain the algorithms in more detail.

3.1 Clustering

Clustering is the task of grouping instances in the data so that the instances in the same group are similar to each other and dissimilar to instances in the other groups [3]. There are many similarity metrics to measure similarity or dissimilarity. One of them can be chosen according to the properties of the data used in clustering. There are many algorithms proposed for clustering, but in this work we use partition relocation methods which improve the clusters gradually by relocating the instances to the clusters.

3.1.1 K-Means Algorithm

K-means algorithm is a clustering algorithm which utilizes the partition relocation approach [11]. "k" is provided by the user and represents the number of clusters. The algorithm firstly initializes means by choosing k random centers from the data. After this step, the algorithm goes over two steps until a convergence is found. At each iteration, it computes the Euclidean distance from each object to each cluster mean.

Then, it assigns each data point to the nearest cluster. Lastly, it computes the new means for all clusters. The algorithm stops when there are no more changes in the clusters.

Algorithm 1 K-Means Algorithm

Input: k,$data$, $means$, $clusters$

Output: $means$,$clusters$

```
1: function KMEANS($k$, $data$, $means$, $clusters$)
2:     initialize $k$ $means$ randomly from the $data$
3:     repeat
4:         for all $instance \in data$ do
5:             $nearestCluster \leftarrow 0$
6:             $minimumDistance \leftarrow$ maximum value
7:             for $i = 0$ to $means.length$ do
8:                 $distance \leftarrow ||instance - means[i]||$
9:                 if $distance < minimumDistance$ then
10:                    $minimumDistance \leftarrow distance$
11:                    $nearestCluster \leftarrow i$
12:                end if
13:            end for
14:            assign $instance$ to the $nearestCluster$
15:        end for
16:        update $means$ according to the new $clusters$
17:    until convergence is found
18: end function
```

Since it is a clustering algorithm, its objective is to minimize the error sum of squares (SSE) which is calculated by Equation 3.1. In the equation, C_i represents the ith cluster, μ_i represents the mean of the ith cluster and c represents the instance.

$$SSE = \sum_{i=1}^{k} \sum\nolimits_{c \in C_i} ||c - \mu_i|| \tag{3.1}$$

Although K-means algorithm is popular as a clustering algorithm, it has some drawbacks too [3]. First of all, the algorithm is sensitive to initial mean selection. Secondly, the algorithm has problems about the outliers and empty clusters. Thirdly, the algorithm does not work when data has non-numeric attributes. Finally, the algorithm can not cluster properly when data includes clusters with differing sizes, densities and non-globular shapes.

3.1.2 K-Medoids Algoritm

K-Medoids is a clustering algorithm which is very similar to K-means algorithm. In K-medoids instead of calculating the mean, cluster centers are determined among the instances of the data [14].

The algorithm starts with a random selection of cluster centers from the instances. After this step, the same procedure is applied as in the K-means algorithm. The difference is that instead of calculating means after relocation, K-medoids chooses the new medoid of the cluster. Medoid can be defined as the representative instance of the cluster, whose sum of distance to all instances in the cluster is the minimum.

Since this algorithm calculates medoids of each cluster in every iteration, it is more time consuming compared to K-means algorithm. This algorithm is useful when the data has non-numeric attributes and one can define a custom similarity metric for these attributes.

3.2 Classification

Classification is a supervised learning task which tries to predict the class of unlabelled instances given the training data [23]. Training data consists of labelled instances, which are sometimes referred as records and tuples, in the form of <**X**,**y**> where **X** is the attribute vector and **y** is the class value. The attributes in the attribute vector can be continuous or discrete. However, in classification task the class value is always discrete.

Classification is the task of learning the function **f** which maps an attribute set **X** to one of the class labels **y**. This function, which is sometimes referred as classification model, can be used for two different purposes. Firstly, it can give descriptive information about the data. Using this model, one can infer which attributes affect the class value and which attributes are important in this data. Secondly and more importantly, this model can be used to classify unlabelled instances which can be named as prediction.

3.2.1 NBTree Algorithm

NBTree algorithm is built upon two different classification algorithms: Naive Bayesian Classification and C4.5 algorithm [16]. For this reason, before going into the details of NBTree algorithm, we will first introduce these preliminary algorithms.

3.2.1.1 Naive Bayesian Classification

Naive Bayesian classification algorithm assumes two properties about the data. Specifically, it assumes that the attributes are conditionally independent and the classification process is not affected by any unseen attributes. With these assumptions, the algorithm uses Bayes' rule in order to compute the class probabilities with respect to the given attribute set [13].

To formalize the classification process of Naive Bayes classifiers, let C be the random variable denoting the class of an instance and let X be a vector of random variables denoting the attribute values. In addition, let x represents an observed attribute value vector and let c represents a specific class value. Given x to find the class value, the algorithm uses Bayes' rule to calculate the probability of each possible class value as shown in Equation 3.2.

$$p(C = c|X = x) = \frac{p(C = c)p(X = x|C = c)}{p(X = x)} \tag{3.2}$$

Because the algorithm assumes that the attributes are conditionally independent, the following equation can be obtained

$$p(X = x|C = c) = \prod_i p(X_i = x_i|C_i = c_i) \tag{3.3}$$

Equation 3.3 can be computed easily using the training data. Since the denominator part of Equation 3.2 is just a normalization factor, its value does not need to be calculated. Instead, the probabilities can be normalized so that the sum of $p(C = c|X = x)$ over all classes is one.

Naive Bayes classification algorithm deals with continuous and discrete attributes using a different manner. For discrete attributes, $P(X = x|C = c)$ is calculated as a single number between 0 and 1, corresponding to the probability that X takes the specific value x when the class value is c. For continuous attributes, this algorithm assumes that the values are normally distributed and it tries to estimate a Gaussian distribution.

3.2.1.2 C4.5 Algorithm

C4.5 algorithm is a decision tree based classification algorithm which is proposed by Quinlan [20].

The decision tree construction phase of C4.5 employs a recursive approach and consists of three cases about the samples. Let T be the set of instances in the construction phase of the algorithm at a specific recursion step. This phase has the following conditions about T:

- T can comprise of instances of a single class C_i. If this is the case, a leaf node which recognizes the class value C_i is created as the decision tree.
- T contains no instances. If this is the case, the algorithm needs data other than T to determine the class value to be recognized. In C4.5 algorithm, the decision tree for T is a leaf node which identifies the most common class value of the parent node.
- T consists of instances which belong to more than one class. In this case, the algorithm tries to split T so that the resulting sets are closer to the sets consisting of instances belonging to a single class.

  To split T, the algorithm uses tests based on a single attribute and having one or more mutually exclusive results (R_1,...,R_n). After the test is applied on T, T

is splitted into T_1,...,T_n where T_i contains all the samples having the result R_i in the selected test.

As a result of this process, the decision tree for T includes a decision node to compute the test on the single attribute and one branch for each possible result of the selected test. The construction method is called again for all splitted sets T_1,...,T_n.

To select the single attribute for the test in the third case, C4.5 calculates the information gain of the attribute based splits. After this computation, the algorithm selects the attribute with the highest gain value as the attribute for the test. To understand the gain definition used in C4.5 algorithm, the concept of entropy should be defined first. If S is any set of instances, let freq(C_i,S) be the number of instances in S that belong to the class C_i, let $|S|$ be the number of instances in S and let k be the number of classes. Then the entropy, which can be defined as the average information needed to determine the class of an instance in S, is computed by using Equation 3.4.

$$info(S) = -\sum_{i=1}^{k} \frac{freq(C_i, S)}{|S|} \times log_2(\frac{freq(C_i, S)}{|S|}) \tag{3.4}$$

They also employ a similar measurement after set S is splitted with respect to the n results of the test X. The expected information requirement can be found as the weighted sum over the subsets, as

$$info_X(S) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} \times info(S_i) \tag{3.5}$$

The measure gain determines the information gained by splitting S using the test X and it is calculated using Equation 3.6.

$$gain(X) = info(S) - info_x(S) \tag{3.6}$$

The algorithm also proposes three testing methods for the attributes. For discrete attributes, it proposes a standard test with possible values of the attribute. For continuous attributes, the proposed method determines a threshold value Z and employs a binary test with results $Y \leq Z$ and $Y > Z$. The final proposed test method on discrete attributes includes the allocation of possible values to a variable number of groups. In this method, the attribute is tested using the group membership and a branch is created for each group.

NBTree algorithm is a hybrid algorithm which constructs a decision tree using C4.5 algorithm with Naive Bayes classifiers at leaf nodes. NBTree employs a utility definition to determine the action taken for the sample set S for the third case of constructing phase of decision tree in C4.5 algorithm. If it decides that the utility of using Naive Bayes algorithm for the current node is greater than the utility of splitting S, it creates a Naive Bayes classifier for the current node and return. Otherwise, it splits the sample set as shown in C4.5 algorithm. The details of the algorithm is given in Algorithm 2.

Algorithm 2 NBTree Algorithm

```
Input: $S$
Output: $nbtree$
1: For each attribute $X_i$, compute the utility, $u(X_i)$ of a split on attribute $X_i$.
2: $j \leftarrow argmax_i(u_i)$ i.e, the attribute with the highest utility is selected.
3: If $u_j$ is not significantly better than the utility of the current node, create a Naive-
   Bayes classifier for the current node and return $nbtree$.
4: Partition $S$ with respect to the test on $X_j$ as explained in C4.5 algorithm.
5: For each possible outcome, call the algorithm on the subset corresponding to the
   outcome.
```

In this algorithm, utility is defined in order to compare the gain employing a Naive-Bayes classifier at the current node to splitting the set of samples as in C4.5 algorithm. The utility of the node is defined as the accuracy estimate of using Naive-Bayes classifier at this node by computing 5-fold cross validation. Utility of split is computed by weighted sum of utility of nodes which will be created after partitioning the set of samples. The algorithm also makes use of a significance parameter. If a split of the set of samples results in a relative reduction in the error which is greater than 5% and there are at least 30 instances in the resulting sets, the algorithm accepts the split as significant.

3.2.2 Decision Table

Decision table algorithm is a supervised learning algorithm which is proposed by Kohavi [15]. The algorithm uses the Decision Table Majority (DTM) model as the classifier. DTM consists of a schema which is the list of attributes that will be used in the classification process, and a body which contains labelled instances with respect to the attributes included in the schema.

In classification process, the algorithm looks for perfect match in the decision table by handling the attributes in the schema only in the following manner:

- If perfect matches are found in the table, the algorithm returns the most common class of perfect matches.
- Otherwise, the algorithm returns the majority class in DTM.

The algorithm uses an induction method to determine which instances will be stored in the body and which attributes will be stored in the schema. Let $A = X_1, ..., X_n$ be a set of attributes and let S be a set of m instances with the attributes in A. With a subset of attributes $A' \subseteq A$, $DTM(A', S)$ can be defined by the schema A' and the body which includes the instances of S projected on A'. The induction method tries to choose a schema A^* such that A^* has the minimum error with respect to a target function f as can be seen in Equation 3.7.

$$A^* = \underset{A' \subseteq A}{\arg\min}\, err(DTM(A', S), f) \tag{3.7}$$

In order to search attribute subsets space efficiently, they converted the problem to the problem of state space search. After this transformation, they used best first search to heuristically search the state space [18].

3.2.3 AdaBoost Algorithm

AdaBoost algorithm is a supervised learning algorithm which employs boosting technique in order to enhance the accuracy performance of any learning algorithm [7].

Boosting method works by running a given learning algorithm on the different distributions of the training data and then integrating the classifiers into a single classifier.

AdaBoost algorithm takes training data, a learning algorithm and the number of iterations as the input. At each iteration, the algorithm calls learning algorithm with different distribution of the training data. On iteration t, the booster executes the learning algorithm with a distribution D_t over the training data S. As the output, the learning algorithm gives a hypothesis $h_t : X \rightarrow Y$ in accordance with the input distribution. In other words, the underlying learning algorithm tries to minimize the classification error with respect to D_t. After T rounds, the booster merges all hypotheses $h_1, ..., h_T$ and creates a final hypothesis h_{fin}.

As shown in Algorithm 3, the distribution is updated with respect to the error related variable B_t and the normalization constant Z_t. With this update, the algorithm tries to increase the weight of hard examples which are not correctly classified by the previous hypotheses and also tries to decrease the weight of easy examples which are correctly classified by many of the previous hypothesis. The weighting scheme used by the algorithm increases the accuracy of the underlying supervised learning algorithm.

3.3 Sequential Pattern Mining

Sequential pattern mining is simply defined as determining the complete set of frequent sequences in a set of sequences [1].

The problem of sequential pattern mining can be defined as follows: Initially, a database D which includes customer transactions is given. In this database, each transaction has the following attributes: customer id, transaction time and set of items purchased. It is assumed that no two transactions have same customer id and transaction time. In other words, each transaction of the same customer has a different time stamp. The other point about the transaction database is that the quantities of items are not stored in the database. The item was either purchased or not purchased. An itemset is a nonempty set of items. A sequence s is a temporally ordered list of itemsets [22]. It is denoted by $< s_1, s_2, ..., s_n >$ where each s_j is an itemset. A sequence

Algorithm 3 AdaBoost Algorithm

Input: S: a set of instances i.e training data, LA: a supervised learning algorithm, T: the number of iterations

Output: h_{fin}: the final hypothesis

```
1: $D_1(i) \leftarrow 1/m$ for all $i$
2: for $t = 1$ to $T$ do
3:     Call $LA$, with the input distribution $D_t$
4:     Get back a hypothesis $h_t : X \to Y$
5:     Compute the error of $h_t$:
```

$$\varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i) \tag{3.8}$$

```
6:     Calculate the error related variable $\beta_t$:
```

$$\beta = \frac{\varepsilon_t}{1 - \varepsilon_t} \tag{3.9}$$

```
7:     Update distribution $D_t$:
```

$$D_{t+1} = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases} \tag{3.10}$$

In equation 3.10, Z_t is a normalization constant chosen so that D_t is a distribution.

```
8: end for
9: Merge all hypothesis and create the final hypothesis $h_{fin}$:
```

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t} \tag{3.11}$$

$s =< s_1, s_2, ..., s_n >$ is said to be contained in $s' =< s'_1, s'_2, ..., s'_m >$ if there exists integers $i_1 < i_2 < ... < i_n$ such that $s_1 \subseteq s'_{i_1}, s_2 \subseteq s'_{i_2} ... s_n \subseteq s'_{i_n}$. The support of a sequence s is the ratio of the number of customers whose sequences contain s to the number of all customers. A sequence is called a large sequence if its support is greater than a minimum support. In a set of sequences, a sequence s is maximal if only s is not contained in any other sequence. After these definitions, we can define sequential pattern mining problem formally. Given a database D including transactions and a minimum support threshold, the problem is to find the complete set of maximal large sequences among all sequences.

3.3.1 AprioriAll Algorithm

AprioriAll algorithm is a sequential pattern mining algorithm which consists of five phases [1]. As explained above, this algorithm finds maximal large sequences given a database D and a minimum support parameter ms. Before going into details of the algorithm, we will define terminology used in this algorithm. The length of the sequence is the number of itemsets in the sequence. A sequence is called a k-sequence, if its length is k.

The first phase of the algorithm is sorting phase. In this phase, the algorithm sorts the transaction records in the database with respect to its customer id and transaction time. In other words, this phase converts the transaction database into customer sequences.

The second phase which is called litemset phase finds large itemsets in the customer sequences. Actually, this phase finds the set of all large 1-sequences since it is just $< l > | l \in L$. For this phase, an algorithm which is proposed for finding large itemsets can be used. There is only a difference in the support definition: In algorithms proposed for this problem, the support is defined as the ratio of the number of transactions including this itemset. However, in sequential pattern mining problem, the support is defined as the ratio of customer sequences including this itemset. After finding the large itemsets, these itemsets are mapped to a set of contiguous integers. The intuition behind this mapping is that the time required to compare two itemsets for equality and to check whether one sequence is contained in another sequence is reduced.

The third phase of the algorithm is transformation phase. In this phase, using the litemset mapping found in the previous phase, each transaction is converted to the set of litemsets contained in this transaction.

The fourth and the main phase of the algorithm is sequence phase. In this phase, the algorithm finds all large sequences starting with the large 1-sequences set obtained in litemset phase. Let L_{k-1} be the set of large (k-1)-sequences. To find large k-sequences, the algorithm generates candidates from L_{k-1} by joining this set by itself. After the candidates are generated, the algorithm counts each candidate by passing over the database. As a final step, the algorithm forms the set L_k with the candidates which have the required minimum support. The algorithm iterates until the set L_k is empty. The details of the algorithm is shown in Algorithm 4.

Algorithm 4 AprioriAll Algorithm

```
Input: $td$: transformed database of customer sequences
$ms$: minimum support parameter
Output: $frequentPatternSet$: the set of large sequences
1: $L_1 \leftarrow$ large 1-sequences found in litemset phase
2: for $k = 2$ TO $L_{k-1} \neq \emptyset$ do
3:     $C_k \leftarrow$ New candidates generated from $L_{k-1}$
4:     for all customer-sequence $c \in td$ do
5:         Increment the count of all candidates in $C_k$ that are contained in $c$
6:     end for
7:     $L_k \leftarrow$ candidates in $C_k$ having the minimum support.
8:     $k \leftarrow k + 1$
9: end for
10: $frequentPatternSet \leftarrow$ maximal sequences in $\cup L_k$.
```

The last phase of the algorithm is called the maximal phase. In this phase, non-maximal sequences are pruned from the union of large sequence sets.

CHAPTER 4

DATA AND PROBLEM DEFINITION

In this chapter, we provide information about the data used in this work and we define the problem of location prediction.

4.1 Call Detail Record Data

In this work, we utilize the Call Detail Record (CDR) data of one of the largest mobile phone operators of Turkey. The data corresponds to an area of roughly 25000 square km with a population around 5 million. Almost 70% of this population is concentrated in a large urban area of approximately 1/3 of the whole region. The CDR data contains roughly 1 million users' log records for a period of 1 month. For each user, there are 30 records per day on average. The whole area contains more than 13000 base stations.

Each record in this dataset is a phone action which can be a phone call, short message or a GPRS connection. There are eleven attributes in every record, namely first cell id, first phone number, first city, second cell id, second phone number, second city, action date, action time, action type, url and duration. An example record from the CDR data can be found in Table 4.1.

4.1.1 Attributes

- First Cell Id: Base station id which first user is connected to.
- First Phone Number: The phone number of the first user. Due to privacy con-

Table 4.1: Example CDR Record

17083	7bcfc0259b9c8a4af95177a7e79bcd28	06
17083	28119ffa652d31607a3bb573bd3d594b	06
20120907	170251	mmo
	47	

straints, it is an anonymized value of a real phone number.

- First City: The city number of first user. Since all of the records in CDR data are in Ankara, this attribute is same for all of them.
- Second Cell Id, Second Phone Number, Second City: These attributes have same properties with the 3 attributes above.
- Action Date: Date of the action which is represented as string value.
- Action Time: Time of the action which is represented as string value.
- Action Type: A categorical attribute which shows the action type. It can be GPRS, incoming phone call, outgoing phone call, incoming short message, outgoing short message.
- Url: The url value for the GPRS connection. It is null for the other action types.
- Duration: The duration of the phone call. It is null for the other action types.

4.2 Problem Definition

In this section we will give the problem definition for location prediction. Let D be a database of CDR records explained in Section 4.1. Since the attributes first city, second cell id, second phone number, second city, action type, url and duration are not related with the location prediction problem, we can filter out these attributes. After this transformation phase, the database D' consists of data with attributes of cell id which represents a region covered by this cell tower, phone number, date and time. Region is not necessarily a single base station. It is possible to combine base stations to form larger regions.

A *sequence* is the ordered list of regions with respect to date and time information and it is expressed as $s < i..j >$, where i is the starting location and j is the last location in the sequence. A sequence of length k is called *k-sequence*.

We call the locations of the users ordered by date and time information as the *user sequence*.

Given a database D of user phone actions, the problem of location prediction is to build a model from the user sequences in order to predict the user's next location given the previous locations.

CHAPTER 5

BASIC LOCATION PREDICTION METHODS

In this chapter, we give information regarding how basic location prediction methods are applied to CDR data.

5.1 Location Prediction with Clustering Algorithms

In this section, we focus on clustering algorithms used in this work to predict the next location of the user.

5.1.1 Preprocessing

In order to apply clustering algorithms in location prediction domain, we need to apply preprocessing to the data.

5.1.1.1 Discretization

First of all, to be able to construct users' daily movements, we need to discretize the data. Otherwise, daily sequence records can be of different lengths. Our aim in this phase is to construct fixed length daily sequences. We discretized a day to 24 slots which is one slot per hour. Then, data is discretized according to their date and time values. We used a heuristic at this step: If a slot has more than one corresponding values in the user sequence, the most frequent base station id in this slot is selected. An example discretization is displayed in Table 5.1 and Table 5.2. For instance, in

Table 5.1: User Daily Sequence

Cell Id	Date	Time
6042	01/09/2012	00:01:26
8002	01/09/2012	10:22:16
8002	01/09/2012	10:22:17
6042	01/09/2012	10:27:25
6042	01/09/2012	10:27:28
8002	01/09/2012	10:28:07
8002	01/09/2012	12:51:26
22411	01/09/2012	13:23:31
22411	01/09/2012	13:55:49
22411	01/09/2012	13:57:09
8003	01/09/2012	18:45:36
8003	01/09/2012	18:55:41
8003	01/09/2012	18:57:02
8007	01/09/2012	23:55:41

this example, although there are records in which user is connected to the base station with id of 6042, the discretized 10:00-11:00 slot contains 8002. Due to the heuristic used in discretization, 8002 is selected as the representative base station id of this time slot.

5.1.1.2 Filling the Missing Values

After discretization, filling the missing values is needed, since in some slots, there are not any records which can be due to a data collection error or due to lack of phone action in that hour. We used a heuristic here: If a time slot has missing value, it is filled according to closest slot where it has value. For instance, if 7:00 – 8:00 slot has a missing value, and the closest slot where it has a value is 10:00 – 11:00, 7:00 – 8:00 slot is filled with the value of 10:00 – 11:00 slot. The filled data according to the above example is shown in Table 5.3.

After preprocessing step, daily movement data with 24 attributes is constructed. Each attribute is the representative base station id of this time slot. Since the problem is not related to specific users, the user information is not included.

Table 5.2: Discretized User Daily Sequence

Time Slot	Cell Id
00:00 - 01:00	6042
01:00 - 02:00	-
02:00 - 03:00	-
03:00 - 04:00	-
04:00 - 05:00	-
05:00 - 06:00	-
06:00 - 07:00	-
07:00 - 08:00	-
08:00 - 09:00	-
09:00 - 10:00	-
10:00 - 11:00	8002
11:00 - 12:00	-
12:00 - 13:00	8002
13:00 - 14:00	22411
14:00 - 15:00	-
15:00 - 16:00	-
16:00 - 17:00	-
17:00 - 18:00	-
18:00 - 19:00	8003
19:00 - 20:00	-
20:00 - 21:00	-
21:00 - 22:00	-
22:00 - 23:00	-
23:00 - 00:00	8007

Table 5.3: Filled Discretized User Daily Sequence

Time Slot	Cell Id
00:00 - 01:00	6042
01:00 - 02:00	6042
02:00 - 03:00	6042
03:00 - 04:00	6042
04:00 - 05:00	6042
05:00 - 06:00	6042
06:00 - 07:00	8002
07:00 - 08:00	8002
08:00 - 09:00	8002
09:00 - 10:00	8002
10:00 - 11:00	8002
11:00 - 12:00	8002
12:00 - 13:00	8002
13:00 - 14:00	22411
14:00 - 15:00	22411
15:00 - 16:00	22411
16:00 - 17:00	8003
17:00 - 18:00	8003
18:00 - 19:00	8003
19:00 - 20:00	8003
20:00 - 21:00	8003
21:00 - 22:00	8007
22:00 - 23:00	8007
23:00 - 00:00	8007

5.1.2 K-Medoids Algorithm

We applied K-Medoids algorithm to our data with a different distance measure since Euclidean distance is not applicable for categorical data. The distance is calculated as the number of non-matching time slots for two records. For instance, distance between $< 1, 2, 3, 4 >$ and $< 2, 3, 3, 4 >$ is 2, since there exists two non-matching slots between these records.

We build a clustering model for each time slot. For example, to predict the time slot 09:00-10:00, we clustered the sequences with time slots up to that point (00:00 - 01:00 to 09:00 - 10:00). By employing this approach, in the prediction phase, we used the clustering corresponding to the time slot requested in the prediction query. The cluster closest to the given query is found and the most frequent base station id of the corresponding time slot is given as the prediction.

5.1.3 K-Means Algorithm

In order to apply K-Means algorithm to our data defined in the previous chapter, we need to convert categorical attributes to numeric attributes because of the fact that the mean of the categorical attributes is not a meaningful value. For this reason, we used the coordinates of the base station ids (x and y values) as the time slot values. In other words, each time slot has two values corresponding to the coordinates of the representative base station id of this slot. After this conversion, we define the distance between the records as the sum of Euclidean distance of corresponding time slots.

As explained above, we build a clustering model for each time slot. The only difference is that, after finding the closest cluster for a given query, the algorithm calculates the average of the coordinates in the corresponding time slot in order to determine the closest base station id. This base station id is given as the prediction.

5.2 Location Prediction with Classification Algorithms

In this section, we focus on classification algorithms used in this work to predict the next location of the user.

5.2.1 Preprocessing

In addition to the preprocessing steps explained in Section 5.1.1, we applied one more preprocessing method before applying the classification algorithms.

In the parts of the cities which have a population over the average population, the base stations are closer to each other. In other words, the distribution of the base stations is dense throughout these locations. Since the number of base stations is high, considering each station as a center of the movement does not imply the semantic meaning of the movement. For this reason, before applying classification algorithms, regions are defined by grouping the base stations. To achieve this, we cluster base stations with respect to their location information (x and y coordinates) using k-means algorithm explained in Section 3.1.1. There are 13281 base stations in the original CDR data, and after exploring several other k values, we decided to group them into 100 clusters which we name as regions. Then, in the preprocessed data obtained by applying the steps introduced in Section 5.1.1, the base station ids are replaced with the corresponding region ids.

At the end of this process, the largest cluster contains 656 base stations and the smallest cluster contains only 6 base stations. Visualization of the regions in three different zoom levels can be seen in Figure 5.1, Figure 5.2 and Figure 5.3.

To be able to build classification models for all time slots, we constructed data for all time slots beginning from 8th time slot. To illustrate, for 8th slot, previous 7 region values are taken as attributes and the 8th region is taken as the class value.

After converting the problem of location prediction to the problem of classification and constructing data for all time slots, we created classification models for each time slot using the following algorithms whose details are given in Section 3.2:

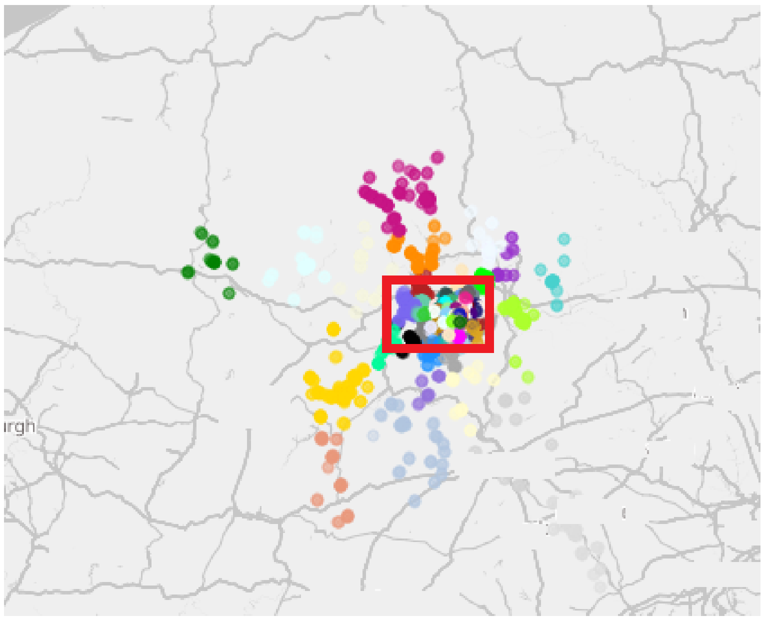

Figure 5.1: Regions in Zoom Level 1

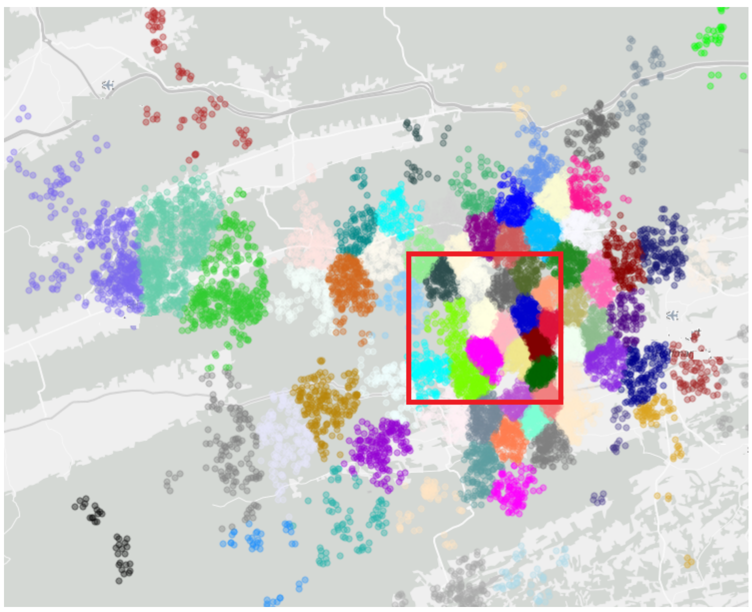

Figure 5.2: Regions in Zoom Level 2

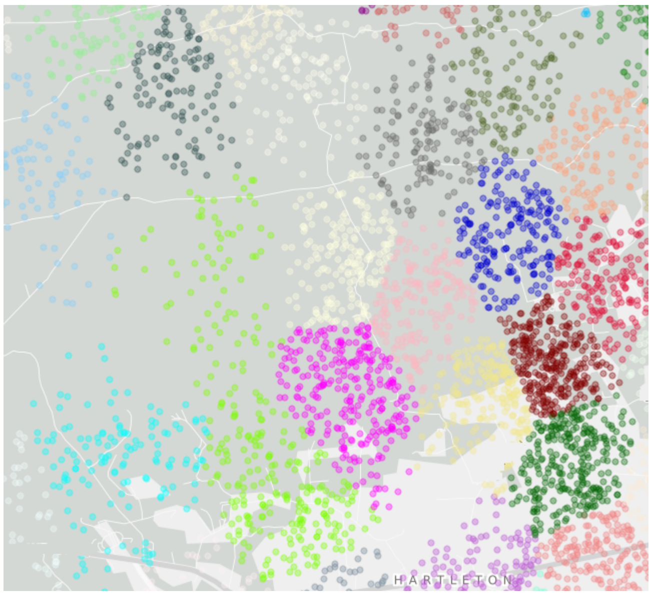

Figure 5.3: Regions in Zoom Level 3

- NB Tree Algorithm [16]
- Decision Table [15]
- AdaBoost Algorithm [7]

5.3 Location Prediction with Sequential Pattern Mining Algorithms

In this section, we give detailed information about how the sequential pattern mining algorithm named as AprioriAll [1] is used in our work.

5.3.1 Preprocessing

We created user daily sequences without discretization and filling the missing values. The records are sorted by temporal information and the result forms the user daily sequence. For the example user records given in Table 5.1, the user daily sequence is computed as:
$< 6042, 8002, 8002, 6042, 6042, 8002, 8002, 22411, 22411, 22411, 8003, 8003, 8003, 8007 >$.

After forming user daily sequences, fixed length sequences are created which are

Table 5.4: 5-Sequences of Daily User Sequence

$< 6042, 8002, 8002, 6042, 6042 >$
$< 8002, 8002, 6042, 6042, 8002 >$
$< 8002, 6042, 6042, 8002, 8002 >$
$< 6042, 6042, 8002, 8002, 22411 >$
$< 6042, 8002, 8002, 22411, 22411 >$
$< 8002, 8002, 22411, 22411, 22411 >$
$< 8002, 22411, 22411, 22411, 8003 >$
$< 22411, 22411, 22411, 8003, 8003 >$
$< 22411, 22411, 8003, 8003, 8003 >$
$< 22411, 8003, 8003, 8003, 8007 >$

used for building the prediction model. In this phase, an input named as $levelCount$ is introduced in order to make the user determine how many previous locations will be used for the prediction. For instance, if the user wants to build a prediction model according to 4 previous locations, the user should select 5 as the $levelCount$ and 5-sequences should be created in this phase. For the example sequence given above, 5-sequences formed is shown in Table 5.4.

At the end of this preprocessing phase, we have the sequences of fixed length to build our prediction model.

5.3.2 AprioriAll Algorithm

We applied AprioriAll algorithm in order to find the frequent sequences in the data obtained applying preprocessing phase.

Then, in the prediction phase, frequent sequences of length $levelCount$ is used. We used a perfect matching strategy in this phase. If a given sequence of length $(levelCount - 1)$ is totally equal to the prefix of a frequent sequence, then the last element of the sequence is given as the prediction. If there are more than one matches for a given query, all possible outputs are given as a set. If there are not any frequent sequences matching with the given query, no output is produced.

Table 5.5: Example Sequences

id	sequence	-	id	sequence
1	<1, 2, 3, 4, 5>		7	<4, 7, 11, 12, 13>
2	<1, 2, 3, 4, 6>		8	<4, 7, 11, 10, 9>
3	<1, 2, 3, 4, 5>		9	<5, 6, 11, 10, 9>
4	<1, 2, 3, 4, 6>		10	<5, 8, 9, 10, 11>
5	<4, 7, 11, 12, 13>		11	<4, 7, 11, 12, 13>
6	<4, 7, 11, 12, 13>		12	<1, 2, 3, 4, 5>

5.3.2.1 Example Run

In this section, we show an example run of AprioriAll for location prediction. For this example, we set $levelCount$ to 5 and support to $1/6$ and use the data given in Table 5.5.

After applying AprioriAll algorithm on this data, the frequent 5-sequences in this data are found as $< 1,2,3,4,5 >$, $< 1,2,3,4,6 >$ and $< 4,7,11,12,13 >$ since their support values are $1/4$, $1/6$ and $1/3$ respectively.

In the prediction phase, if the query given is $< 1,2,3,4 >$, the algorithm gives 5 and 6 as the outputs since $< 1,2,3,4 >$ matches with the prefix of the frequent sequences $< 1,2,3,4,5 >$ and $< 1,2,3,4,6 >$. If $< 4,7,11,12 >$ is given as the query, the algorithm produces 13 as the output. However, if $< 2,3,4,7 >$ is given as the query, the algorithm does not produce any outputs since it does not match with any of the frequent sequences found.

CHAPTER 6

APRIORI-BASED SEQUENCE MINING ALGORITHM WITH MULTIPLE SUPPORT THRESHOLDS

In this section, we introduce our proposed algorithm and define related concepts.

6.1 Preliminaries

In Apriori-based sequence mining, the search space can be represented as a hash tree in which each node contains a location information. A *path* in the tree is a sequence of nodes $< n_1, ..., n_m >$ such that n_1 is a children of the root of the tree and for all $i > 1$, n_{i-1} is the parent of n_i. p <a..b> expresses a path starting with node a and ending with node b.

We say that a path p is equal to a sequence s, denoted by $p = s$, if the length of path p and sequence of s are equal and there is one to one correspondence between the locations of s and the nodes of p.

We say that a sequence s is an element of the hash tree t, if there exists a path p in the tree such that $p = s$ and it is denoted by $s \in t$.

We say that a sequence $s < s_1, s_2, ..., s_n >$ is *contained in* another sequence $s' < s'_1, s'_2, ..., s'_m >$ if there exists integers $i_1 < i_2 < ... < i_n$ such that $s_1 = s'_{i_1}, s_2 = s'_{i_2} ... s_n = s'_{i_n}$.

A sequence s is a subsequence of s' if s is contained in s' and it is denoted by $s \subseteq s'$.

A *rule* is denoted with the notation of $a \rightarrow b$. The left-hand side of the rule contains a sequence consisting of locations and the right-hand side of the rule contains a single location value. $lhs(r)$ denotes the left-hand side of the rule and rhs(r) denotes the right-hand side of the rule. A rule whose left-hand side contains a (k-1)-sequence is said to have a length of k and called as *k-rule*.

6.2 The Algorithm

To build a model which aims to predict the next location of the user, we developed a recursive hash tree based algorithm namely Apriori-based Sequence Mining Algorithm with Multiple Support Thresholds (ASMAMS). This algorithm constructs level based models i.e. hash trees whose nodes contain corresponding base station id and frequency count of the sequence corresponding to the path up to this node.

The main novelty of the algorithm in comparison to the conventional algorithm is the level based support mechanism with a new support definition. In contrast to previous approaches that aim to extract all frequent sequences, we focus on predicting the next item in a sequence in this work. Therefore, we defined a level-based support in order to keep track of the relations between the levels. Conventionally, support of a given sequence pattern is defined as the ratio of the number of the sequences containing the pattern to the number of all sequences in the dataset. If the support threshold is lowered, accuracy increases, but space requirement increases as well. On the other hand, higher threshold values lead to lower space requirement, yet lower accuracy. In order to keep the balance between accuracy and space requirement, in ASMAMS, support of an n-sequence is defined as the ratio of the count of a given sequence s to the count of the parent sequence with length *(n-1)*.

$$support(s) = \frac{\#\ of\ occurrences\ of\ the\ sequence\ s\ with\ length\ n}{\#\ of\ occurrences\ of\ prefix\ of\ sequence\ s\ with\ length\ (n-1)} \tag{6.1}$$

Before going into the details of the algorithm, we need to define the following parameters that will be used in the algorithm.

- *levelCount:* It is the height of the hash tree to be constructed. It also states how many previous base station ids to be used in prediction i.e. $levelCount - 1$.

- *currentLevel:* It denotes the current level throughout the construction of the hash tree.
- *supportList:* It denotes a list of minimum support parameters for each level. Its length is equivalent to $levelCount$.
- *sequences:* A set of fixed-length location id sequences derived from CDR data after preprocessing.
- *tree:* A hash tree where each node stores the location id and the count of sequence represented by a path from root to this node and it is also connected to its children via a hash structure. The root of the tree represents an empty sequence and named as *root*. It is both input and output of this algorithm.
- *tolerance:* An integer value which corresponds to the length tolerance of rule extraction phase. If it is 0, the algorithm extracts rules of length $levelCount$, Otherwise it extracts rules whose lengths are between $levelCount$ and $levelCount - tolerance$.

ASMAMS algorithm has three phases which are model construction, rule extraction and prediction. As given in Algorithm 5, model construction phase is divided into two sub-phases: tree construction and pruning.

Algorithm 5 ASMAMS Model Construction Phase

```
Input: $sequences$,$levelCount$,$supportList$,$currentLevel \leftarrow 1$
Output: $tree$
1: function BUILDMODEL($sequences$, $levelCount$, $currentLevel$, $supportList$, $tree$)
2:     $constructTree(sequences, tree, currentLevel)$
3:     $pruneTree(tree, currentLevel, supportList[currentLevel])$
4:     if $currentLevel \neq levelCount$ then
5:         $buildModel(levelCount, currentLevel + 1, supportList, tree)$
6:     end if
7: end function
```

In the tree construction phase, the data is read sequentially, and new level nodes are added to the corresponding tree nodes. For instance, assume that we are constructing the fourth level of the tree and we have <1,2,3,4> as the sequence. If <1,2,3> corresponds to a path in the input tree, 4 is added as a leaf node as the child of this path with count 1. If we encounter the same sequence, the algorithm only increments the count of this node. If the current tree does not contain <1,2,3>, then 4 is not added to the tree. The construction algorithm is given in Algorithm 6.

Algorithm 6 ASMAMS Tree Construction Phase

```
Input: $sequences, tree, currentLevel$
Output: $tree$
1: function CONSTRUCTTREE($sequences, tree, currentLevel$)
2:     for all $s < l_1..l_{currentLevel} > \in sequences$ do
3:         if $\exists p < root..leaf > \in tree$ s.t $p = s$ then
4:             $leaf.count = leaf.count + 1$
5:         else
6:             if $\exists p < root..leaf > \in tree$ s.t $p = s < l_1..l_{currentLevel-1} >$ then
7:                 $insert(tree, leaf, l_{currentLevel})$ //add $l_{currentLevel}$ as a child of $leaf$
8:                 $l_{currentLevel}.count = 1$
9:             end if
10:        end if
11:    end for
12: end function
```

In the pruning phase, constructed model and the corresponding minimum support value are taken as parameters. In this phase, initially we calculate leaf nodes' support values. If it is below the minimum support value, it is removed from tree, otherwise no action is taken. The detailed algorithm of pruning phase can be found in Algorithm 7.

6.2.1 Rule Extraction

In the rule extraction phase, the algorithm extracts rules from the hash tree built in model construction phase with respect to a tolerance parameter. If tolerance param-

Algorithm 7 ASMAMS Pruning Phase

```
Input: $tree$, $currentLevel$ $minimumSupport$
Output: $tree$
1: function PRUNETREE($tree$, $currentLevel$, $minimumSupport$)
2:     for all $leaf \in tree$ s.t. $depth(leaf) = currentLevel$ do
3:         $support \leftarrow leaf.count/parent.count$
4:         if $support < minimumSupport$ then
5:             delete $leaf$
6:         end if
7:     end for
8: end function
```

eter is set to 0, the algorithm extract levelCount-rules, whose left-hand side contains (levelCount - 1)-sequence and right-hand side contains the output level location, from the levelCount-sequence s as follows:

$s_1, s_2, ..., s_{levelCount-1} \rightarrow s_{levelCount}$

If tolerance is greater than 0, the algorithm extracts rules until the rules have the length of (levelCount - tolerance) as shown in Algorithm 8.

Algorithm 8 ASMAMS Rule Extraction Phase

```
Input: $tree$, $levelCount$ $tolerance$
Output: $ruleSet$
1: function RULEEXTRACTION($tree$, $levelCount$, $tolerance$)
2:     for all $s < s_1, s_2, ..., s_{levelCount} > \in tree$ s.t. $length(s) = depth(tree)$ do
3:         for $t = 0$ to $tolerance$ do
4:             $subSequencesSet \leftarrow$
5:                 $t$-deleted subsequences of $s < s_1, ..., s_{levelCount-1} >$
6:             for all subsequence $s' \in subSequencesSet$ do
7:                 $ruleSet \leftarrow ruleSet \cup \{s' \rightarrow s_{levelCount}\}$ //Add new rule
8:             end for
9:         end for
10:    end for
11: end function
```

6.2.2 Prediction

In the prediction phase, we use set of rules constructed by rule extraction phase to predict user's next location. The prediction algorithm takes a sequence as input and returns a list of predicted locations.

The algorithm firstly checks whether rules with length of $levelCount$ is contained in the given sequence. In that case, the right-hand side of the rules constitute the prediction set. If the rules of length $levelCount$ are not contained in the given sequence, then it checks whether the rules of length $levelCount - 1$ are contained in the given sequence. This continues until the rules are contained in the sequence or until the tolerance parameter is reached but no output is produced. The detailed algorithm of prediction phase is shown in Algorithm 9.

Algorithm 9 ASMAMS Prediction Phase

```
Input: $sequence, ruleSet, levelCount, tolerance$
Output: $predictionSet$
function PREDICT($sequence, ruleSet, levelCount, tolerance$)
    for $t = 0$ to $tolerance$ do
        for all $rule \in rules$ of length $levelCount - t$ do
            if $lhs(rule) \subseteq sequence$ then
                $predictionSet \leftarrow predictionSet \cup \{rhs(rule)\}$
            end if
        end for
        if $predictionSet \neq \emptyset$ then
            break
        end if
    end for
    return $predictionSet$
end function
```

Table 6.1: Example Sequences

id	sequence	-	id	sequence
1	<1, 2, 3, 4, 5>		7	<4, 7, 11, 12, 15>
2	<1, 2, 3, 4, 6>		8	<4, 7, 11, 10, 9>
3	<1, 2, 3, 4, 5>		9	<5, 6, 11, 10, 9>
4	<2, 3, 4, 7, 8>		10	<5, 8, 9, 10, 11>
5	<3, 4, 7, 9, 10>		11	<5, 11, 10, 9, 4>
6	<4, 7, 11, 12, 13>		12	<1, 2, 3, 4, 5>

6.2.3 Running Example

To illustrate our proposed algorithm ASMAMS, we display an example run. In this example, we set level count to 5 and minimum support list to $[1/6, 1/2, 1/2, 2/3, 0]$ and we used the sample sequences shown in the Table 6.1.

In the first level, the data is traversed sequentially and the first location ids in the sequences are added to the hash tree together with their counts. Then in the pruning phase, their support values are calculated and nodes 2 and 3 are pruned since their support fall below the given minimum support 1/6. In the second level, 2-sequences are added to the hash tree with their counts. After support values are found, the nodes <5,6>, <5,8> and <5,11> are pruned since their support values are 1/3 and falls below the given minimum support 1/2. The resulting hash trees can be seen in Figure 6.1.

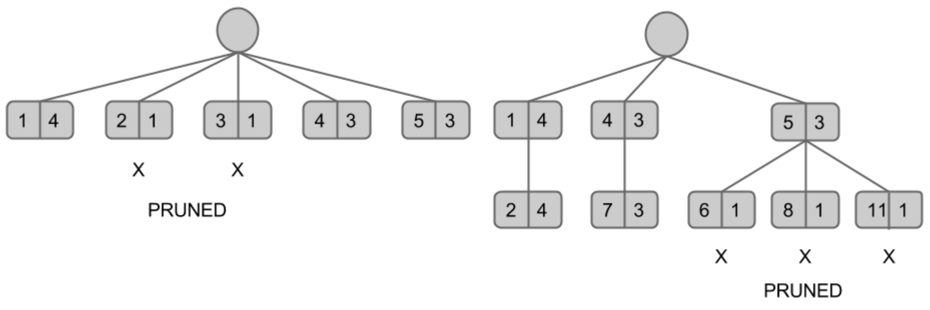

Figure 6.1: Hash tree at the end of the first level (left), Hash tree at the end of the second level (right)

In the third level, 3-sequences are added to the hash tree. None of the nodes are pruned in this level, since the support values are all 1. In the fourth level, after 4-sequences are added to the hash tree, the node <4,7,11,10> is pruned as it does not have the required support. The resulting hash trees can be seen in Figure 6.2

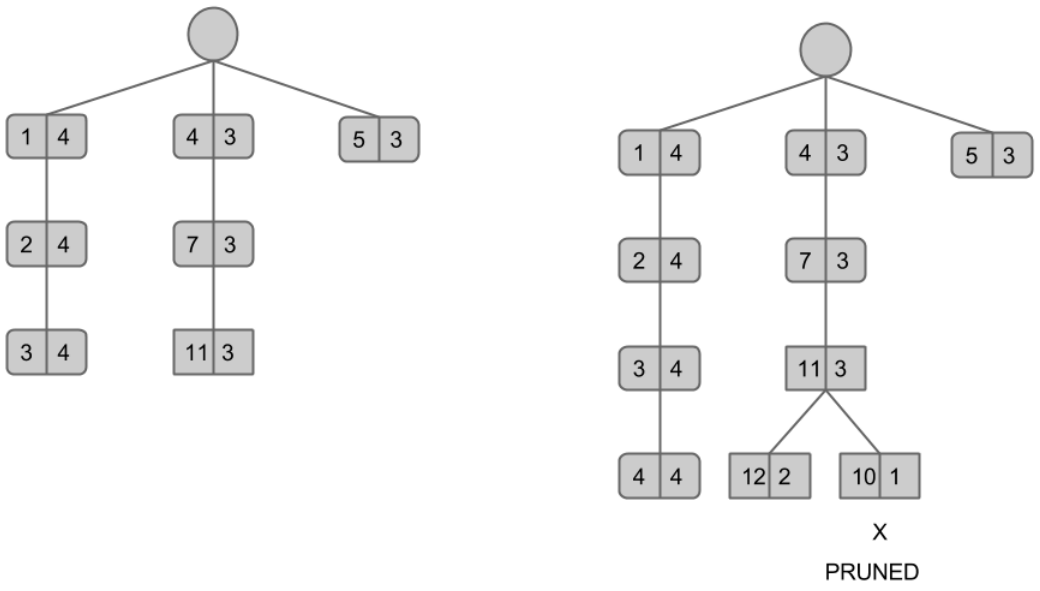

Figure 6.2: Hash tree at the end of the third level (left), Hash tree at the end of the fourth level (right)

In the final level (which is the last level of the hash tree), 5-sequences are added to the hash tree. Since the minimum support value for this level is 0, no pruning occurs. The resulting hash tree can be seen in Figure 6.3.

Using the hash tree constructed by model construction phase which is shown in Figure 6.3, the rules are extracted according to the $tolerance$ parameter. If the tolerance parameter is 0, the 5-rules shown in Table 6.2 are extracted.

Table 6.2: Rules with Tolerance Value 0 (5-rules)

$1, 2, 3, 4 \rightarrow 5$	$1, 2, 3, 4 \rightarrow 6$
$4, 7, 11, 12 \rightarrow 13$	$4, 7, 11, 12 \rightarrow 15$

In this case, for a sequence of $< 1, 2, 3, 4 >$, the algorithm gives the output of 5 and 6. However, for a sequence of $< 1, 2, 8, 3 >$, the algorithm does not generate any outputs.

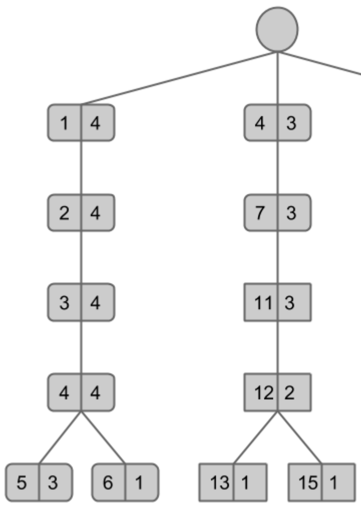

Figure 6.3: Hash tree at the end of the final level

If the tolerance parameter is 1, additional 4-rules shown in Table 6.3 are extracted from the hash tree.

Table 6.3: Extra Rules with Tolerance Value 1 (4-rules)

$1, 2, 3 \rightarrow 5$	$1, 2, 4 \rightarrow 5$	$1, 3, 4 \rightarrow 5$
$2, 3, 4 \rightarrow 5$	$1, 2, 3 \rightarrow 6$	$1, 2, 4 \rightarrow 6$
$1, 3, 4 \rightarrow 6$	$2, 3, 4 \rightarrow 6$	$4, 7, 11 \rightarrow 13$
$4, 7, 12 \rightarrow 13$	$4, 11, 12 \rightarrow 13$	$7, 11, 12 \rightarrow 13$
$4, 7, 11 \rightarrow 15$	$4, 7, 12 \rightarrow 15$	$4, 11, 12 \rightarrow 15$
$7, 11, 12 \rightarrow 15$		

By using the tolerance parameter, for a sequence of $< 1, 2, 8, 3 >$, the algorithm generates the output of 5 and 6, since the left-hand side of the rule $1, 2, 3 \rightarrow 5$ and $1, 2, 3 \rightarrow 6$ are contained in the given sequence.

CHAPTER 7

EVALUATION AND EXPERIMENTAL RESULTS

In this chapter, first we introduce our evaluation method and evaluation metrics, then we present the experimental results for location prediction with clustering algorithms (K-Medoids, K-means), classification algorithms (NBTree, Decision Table, AdaBoost), AprioriAll algorithm and lastly our proposed method ASMAMS.

7.1 Evaluation

For the experimental evaluation, CDR data obtained from one of the largest mobile phone operators in Turkey has been used. For each method, we applied the steps explained in the previous sections and build prediction models accordingly. We have used k-fold cross validation in order to assess the quality of predictions made. As the training phase, we run algorithms on preprocessed data. At the test phase, we predicted the instances in the test set and the result of the prediction is compared against the actual next location value.

7.1.1 Evaluation Metrics

Accuracy metric is used for evaluating the number of correctly predicted test set sequences. It can be defined as the ratio of true predicted test sequences to the total number of test sequences. However, for some test cases, the algorithm can not produce any prediction. Therefore, we defined two accuracy metrics. The first accuracy metric, *g-accuracy* (general accuracy), is the ratio of the number of correctly predicted test sequences to the number of all test sequences. The second one, *p-accuracy* (predictions' accuracy), is the ratio of the number of correctly predicted test sequences to the number of all test sequences which the model can produce a prediction output. In other words, p-accuracy metric does not take no output cases into consideration. In the first form of accuracy calculation, the accuracy result drops sharply for cases that no prediction is able to be performed. These accuracy measures have been described in more detail in our earlier work [19].

Memory Requirement metric measures the relative peak RAM requirement during the algorithm's execution. All memory requirement values are projected to the range [0-100], where 100 represents the maximum memory utilization.

Prediction Count metric is used to evaluate average size of the prediction set in correctly predicted test sequences.

Score metric is introduced since there are 4 different parameters that we want to optimize. It is used for evaluating general performance of our model by combining above metrics into a single one. This metric is only used to determine the minimum support parameters for the optimal model. It is defined as a weighted sum of *g-accuracy*, *p-accuracy*, *memory requirement*(mem_req) and *prediction count*(pred_count) in Equation 7.1.

$$Score = w_1 * \textit{g-accuracy} + w_2 * \textit{p-accuracy} + w_3 * \textit{(100-mem\_req)} + w_4 * \textit{(100-pred\_count)} \tag{7.1}$$

Considering the importance of the parameters the weights are set as follows; w1 = 0.6, w2 = 0.1, w3 = 0.1 and w4 = 0.2.

7.2 Experimental Results

In this section, we will give experimental results of methods explained in Chapter 5 and Chapter 6.

7.2.1 Location Prediction with Clustering Algorithms

Before applying clustering algorithms, we first calculate the same percentage value, which refers to the ratio of being in the same location as the previous location, on the preprocessed data. The same percentage value is nearly 76%. We take this as the baseline value.

We applied K-Medoids algorithm (with different k values) to cluster our daily movements data. As explained in Chapter 5, we used a different distance measure which is the nonmatching time slots of two records. It is observed that 80% of the data goes into the first cluster. There are many nonmatching slots in the records and records are assigned to clusters with respect to similarity with medoids. For this reason, the records whose matching slot count with all medoids is zero forms the first cluster.

Due to the fact that K-Medoids algorithm can not cluster our data properly, we decided that this method is not suitable for our problem. To make sure, the model is evaluated on test data and it gives a general accuracy of 2% which is not acceptable for our data since the threshold value is 76%.

We applied K-Means algorithm (with several k values) to our data as explained in Section 5.1.3. After forming clusters, we tested our model on test set. The model gives a general accuracy of 8% which is not acceptable because of the threshold value.

7.2.2 Location Prediction with Classification Algorithms

We applied NBTree, Decision Table and AdaBoost algorithms on the data which is obtained by applying preprocessing steps explained in Section 5.2.1. We explored several parameters of the algorithms and we built classification models for all time slots. The models are evaluated using k-fold cross validation.

Although the accuracy values of classification algorithms are better than the clustering algorithms, they are still not acceptable since the same percentage is really high as discussed below.

Table 7.1: G-Accuracy of Classification Algorithms for 15th Slot

Algorithm	**G-Accuracy**
AdaBoost	28.18%
NBTree	42.91%
Decision Table	48.29%

The general accuracy obtained by applying classification algorithms is close to the same percentage value. To illustrate the results, we included the accuracy values of algorithms for 15th slot in Table 7.1.

The same percentage value for the 15th slot is 50.24%. As shown in Table 7.1, the accuracy percentage of classification algorithms is not acceptable for the location prediction problem on our data.

7.2.3 Location Prediction with ASMAMS

In this section, we will give experimental results of our proposed algorithm ASMAMS.

For the experiments, we have used 5-sequences (i.e. level count in Algorithm 6 is set to 5), after trying longer and shorter sequences. While shorter sequences, such as 4-sequences or 3-sequences, were increasing prediction count, longer sequences, such as 6-sequences, were decreasing g-accuracy sharply, even though p-accuracy was increasing, since the number of predictable sequences was quickly decreasing. Therefore, 5-sequences seemed as the best for the data in hand, and shorter or longer sequences' results were not useful.

After determining the sequence length and level count for the experiments, we first narrowed down our search space by setting our support values to a set $\{10^{-5}$, 10^{-4}, 10^{-3}, 10^{-2}, $10^{-1}\}$ for each level. We have used the score parameter introduced above to determine this best support list as $[10^{-5}$,10^{-3}, 10^{-3}, 10^{-3}, $10^{-2}]$. Then we have tried all possible non-decreasing combinations as list of support parameters. For every level, we fixed other levels' support values to the support values of the best model

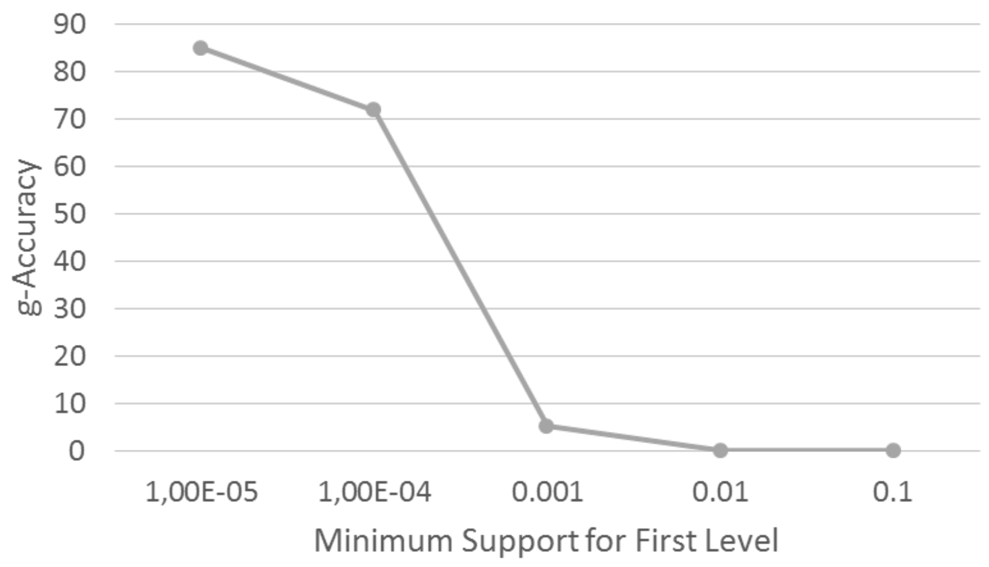

Figure 7.1: Minimum Support for First Level vs g-Accuracy

and we present results of changing this level's minimum support value according to evaluation metrics. Same percentage value refers to the ratio of being in the same location as previous location and is included in the figure to show the improvement provided by ASMAMS.

In a set of experiments, we have analyzed the effect of the minimum support parameter for all levels. In order to analyze the effect, for each level, the experiments are performed with the support values explained above and other levels' support parameters are set to the optimal values. In addition, the tolerance parameter is fixed to 0 for first set of the experiments.

7.2.3.1 G-Accuracy

In this set of experiments, we analyzed the effect of minimum support values on g-accuracy for all levels.

As shown in Figure 7.1, Figure 7.2, Figure 7.3, Figure7.4 and Figure 7.5, for all levels, g-accuracy drops as the minimum support increases. This is because of the fact that when minimum support increases, the number of sequences which survive

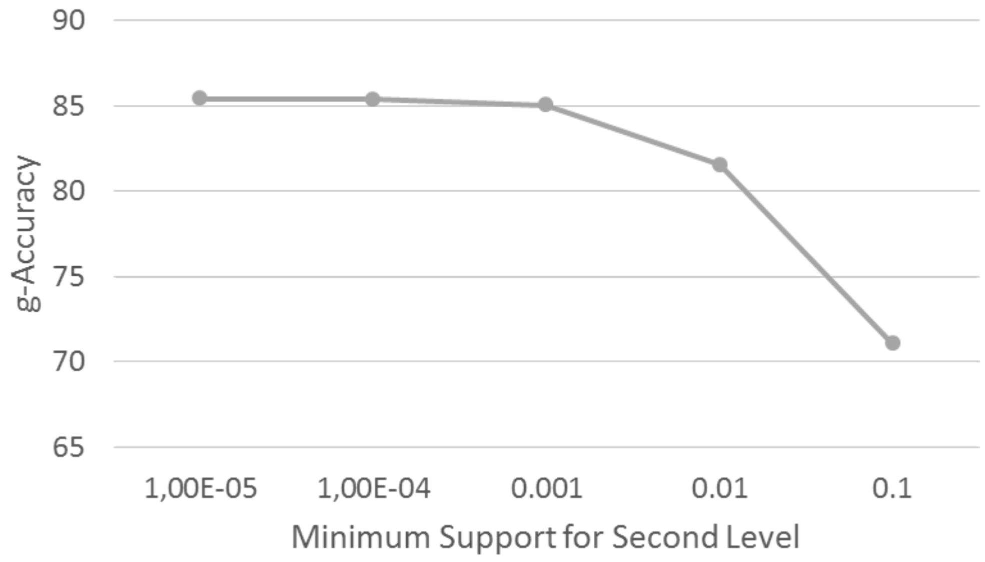

Figure 7.2: Minimum Support for Second Level vs g-Accuracy

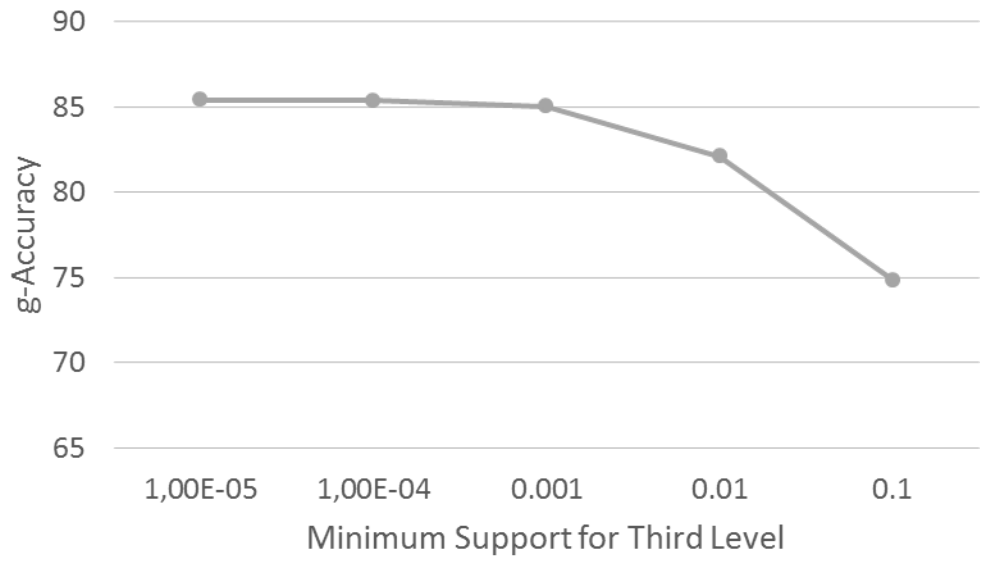

Figure 7.3: Minimum Support for Third Level vs g-Accuracy

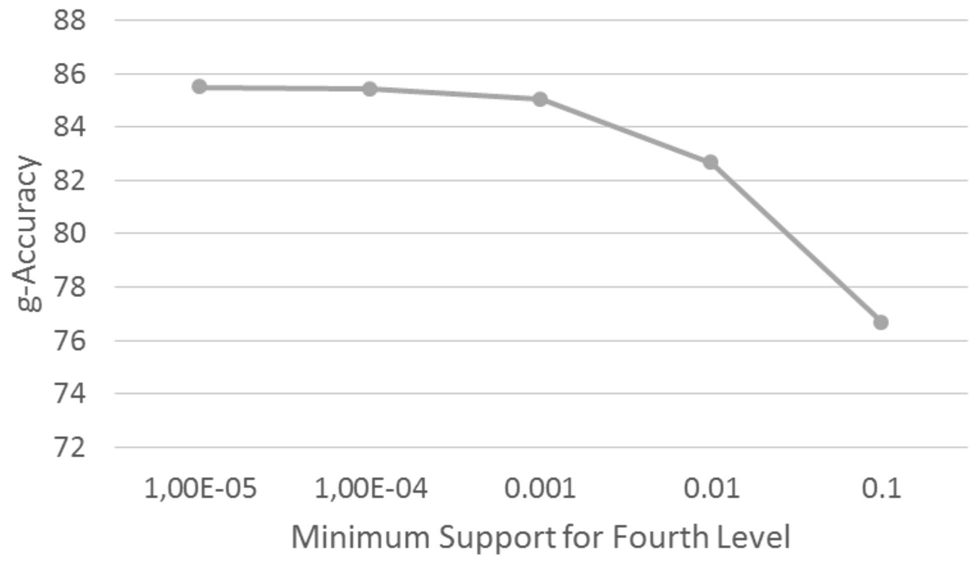

Figure 7.4: Minimum Support for Fourth Level vs g-Accuracy

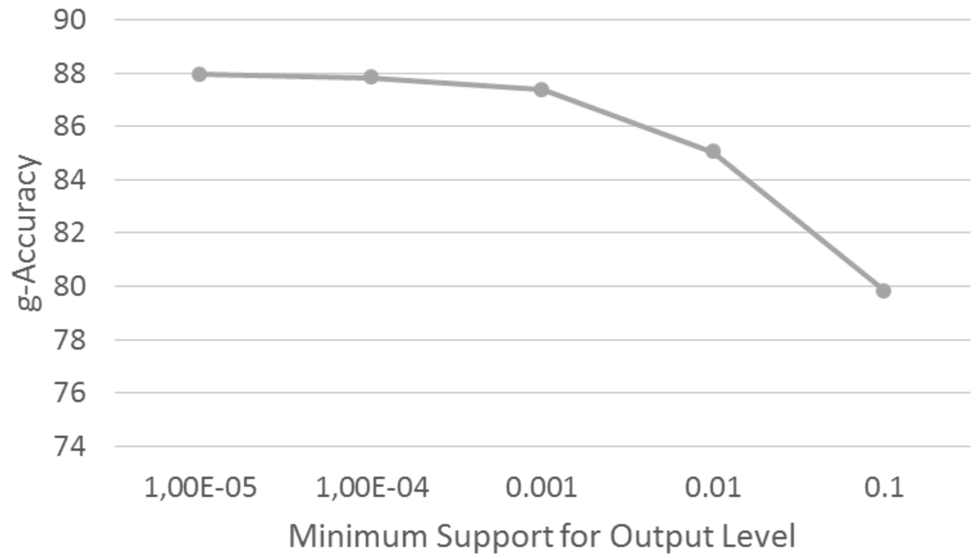

Figure 7.5: Minimum Support for Final Level vs g-Accuracy

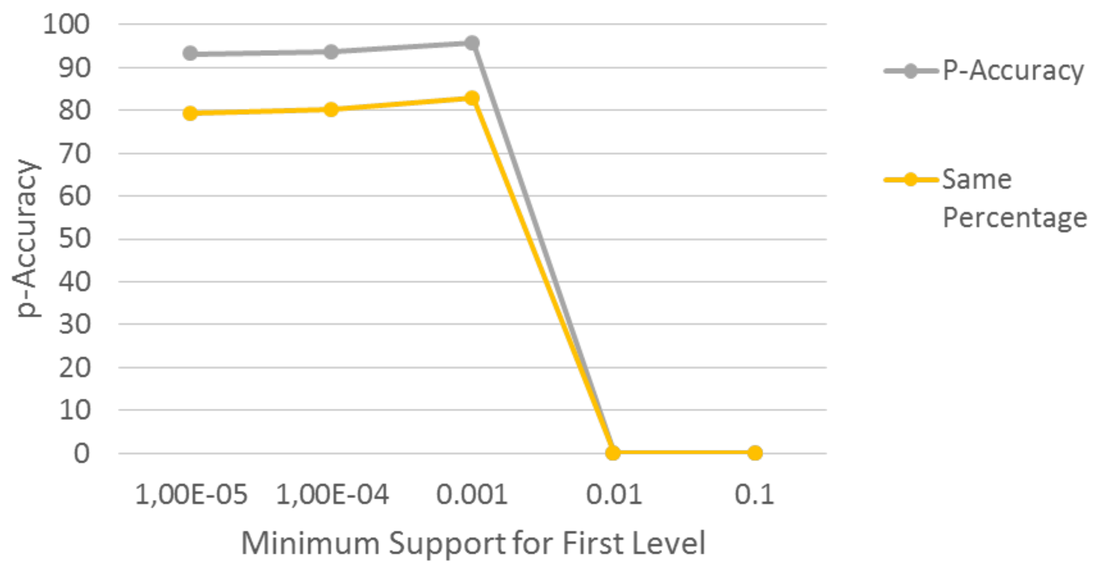

Figure 7.6: Minimum Support for First Level vs p-Accuracy

from pruning decreases. However, this drop is much sharper in the first level since the support variables of the preceding levels affect the successor levels' constructing and pruning phase too. These experiments also show that the algorithm can reach the general accuracy of 85% even setting the tolerance parameter to 0.

7.2.3.2 P-Accuracy

In this set of experiments, we analyzed the effect of minimum support values on p-accuracy for all levels.

Figure 7.6, Figure 7.7, Figure 7.8, Figure 7.9, Figure 7.10 indicate that p-accuracy shows slight increase in first and intermediate levels, and then, there is also a small drop in the final level. In the first level, it drops to 0 since no sequences are survived from pruning when minimum support for the first level is high. The slight increase in the intermediate levels is due to the fact that the number of predictable patterns decreases more rapidly than g-accuracy. The change in minimum support parameter for the final level effects the prediction size set. When the support is increased, the prediction set size decreases. For this reason, although the models are able to predict the same set of sequences, the models with high minimum support for the final level

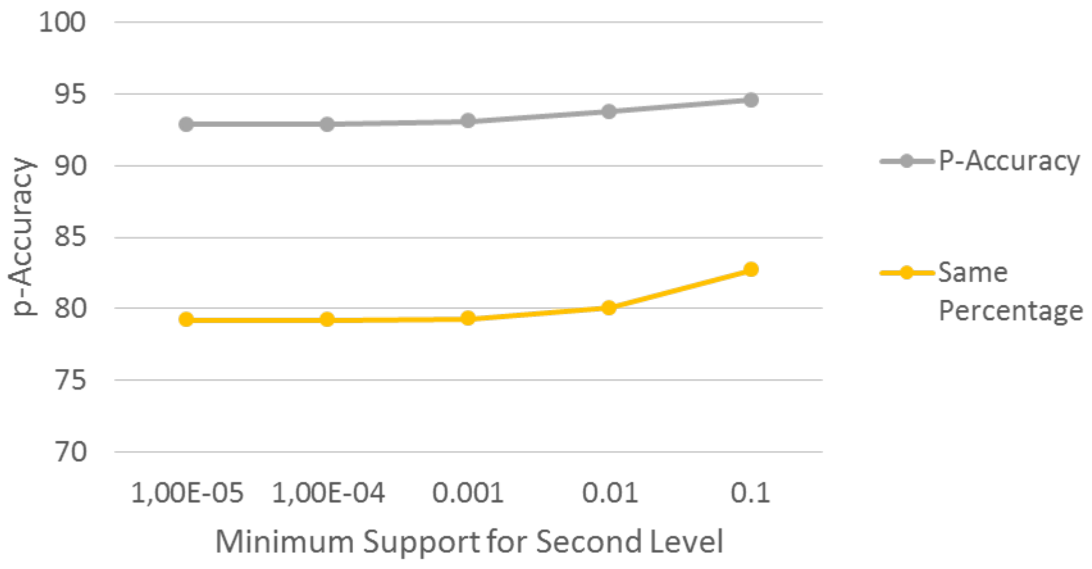

Figure 7.7: Minimum Support for Second Level vs p-Accuracy

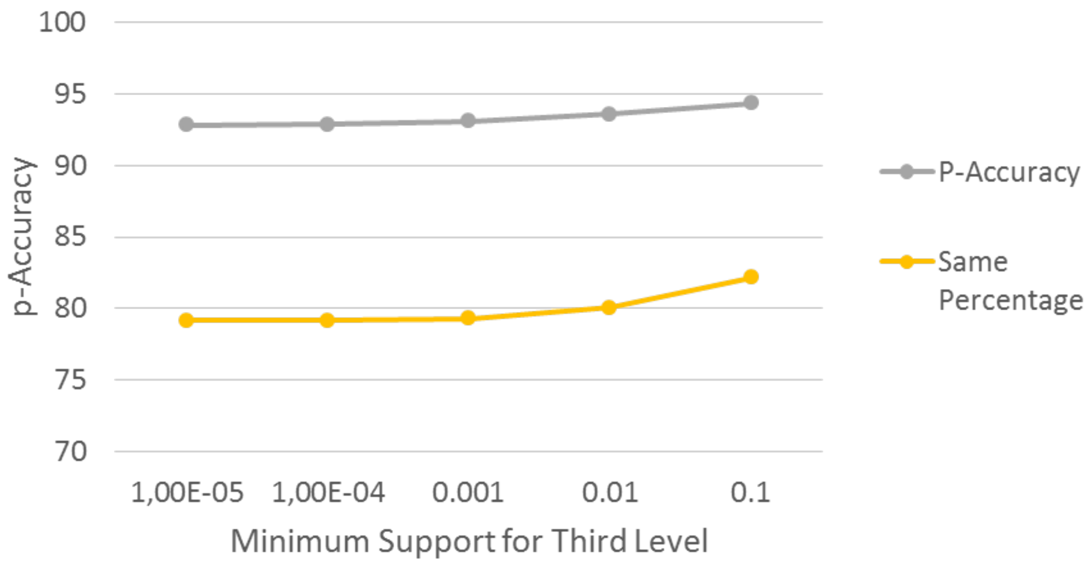

Figure 7.8: Minimum Support for Third Level vs p-Accuracy

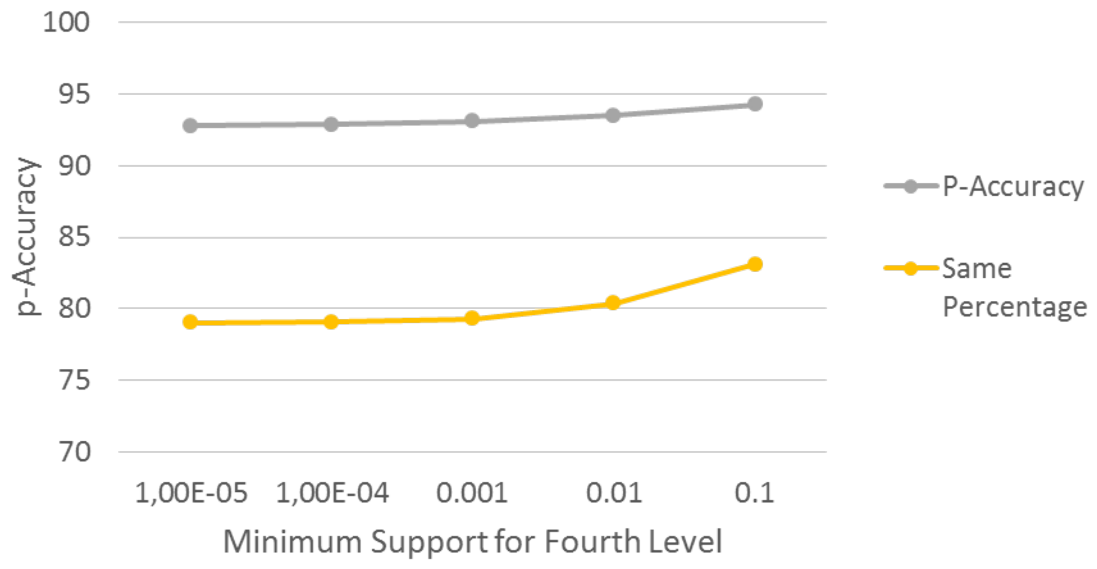

Figure 7.9: Minimum Support for Fourth Level vs p-Accuracy

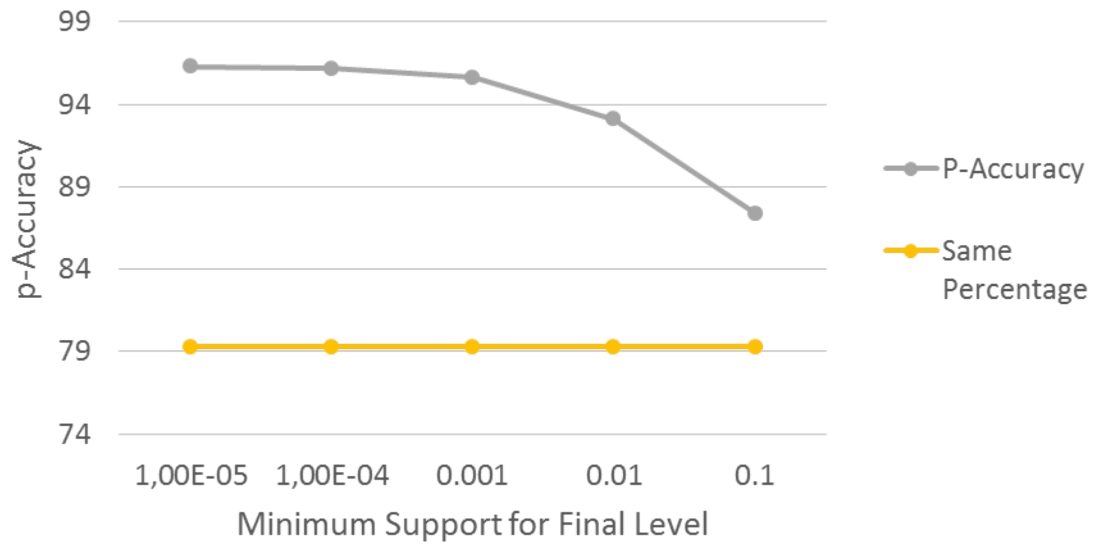

Figure 7.10: Minimum Support for Final Level vs p-Accuracy

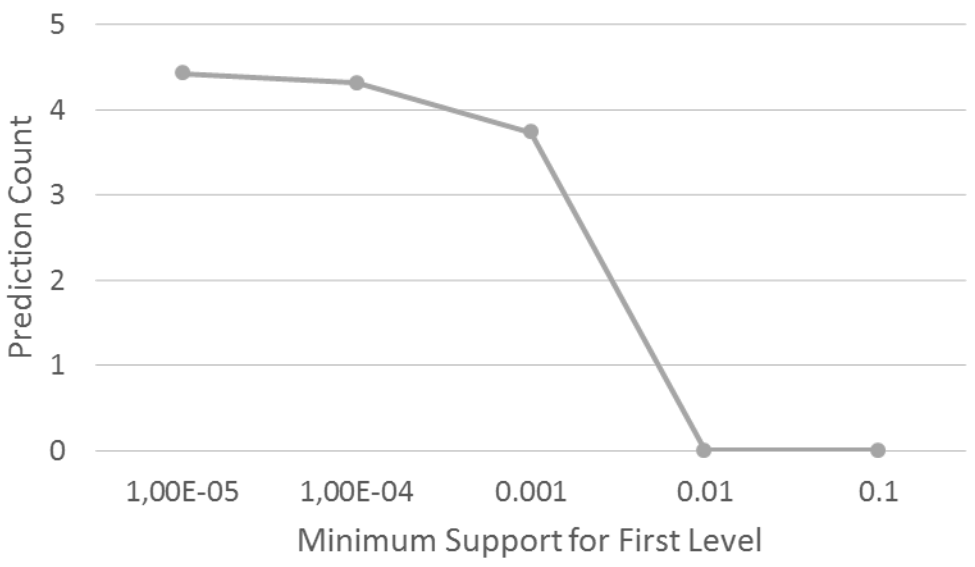

Figure 7.11: Minimum Support for First Level vs Prediction Count

have lower p-accuracy values. These figures also show the percentages of locations which are exactly the same as the previous ones for all the experiments as well. Our p-accuracy results show that, the correct prediction (of p-accuracy) can be increased even above 95% with our model which means nearly 15% of increase in accuracy with respect to the same percentage value.

7.2.3.3 Prediction Count

In this set of experiments, we analyzed the effect of minimum support values on prediction count for all levels.

The drop in the prediction count metric for all levels except the final level is negligible as displayed in Figure 7.11, Figure 7.12, Figure 7.13, Figure 7.14, Figure 7.15. In the first level, prediction count drops to 0 for the minimum support parameter values 0.01 and 0.1 since all of the sequences are pruned because of the high support. In the final level, prediction count decreases much faster as well since the change of minimum support for the final level effects the prediction set size directly. It is also observed that the prediction count values are at acceptable levels (4 over nearly 13000 base stations).

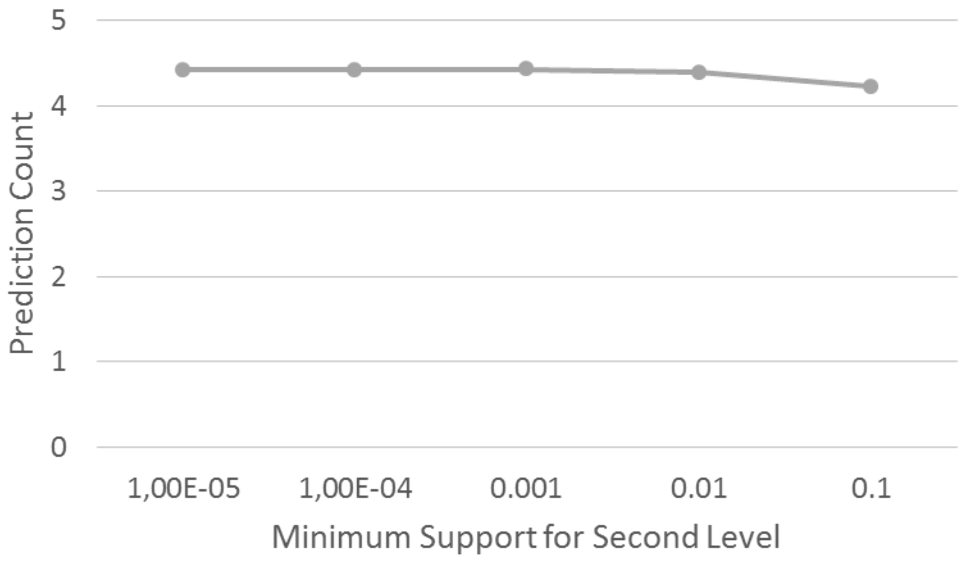

Figure 7.12: Minimum Support for Second Level vs Prediction Count

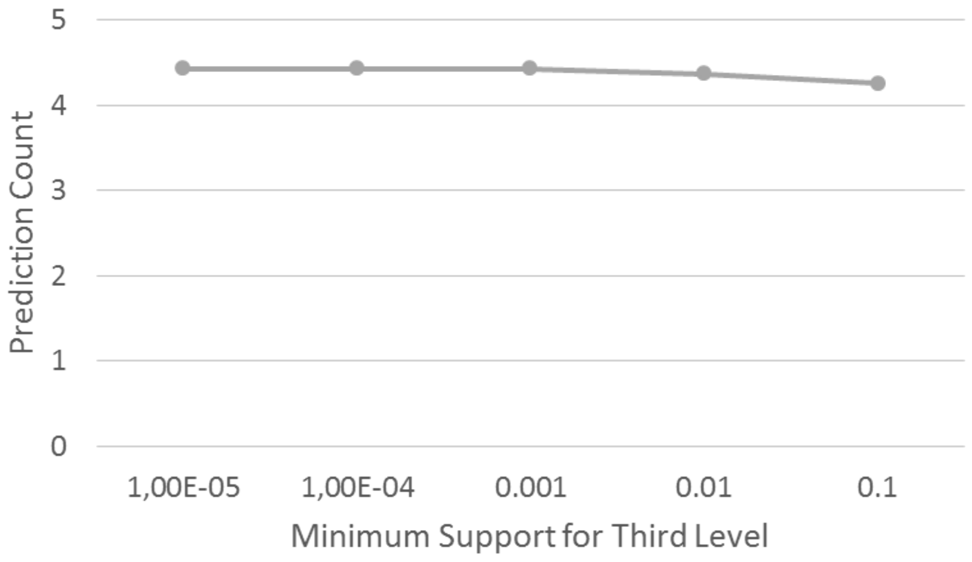

Figure 7.13: Minimum Support for Third Level vs Prediction Count

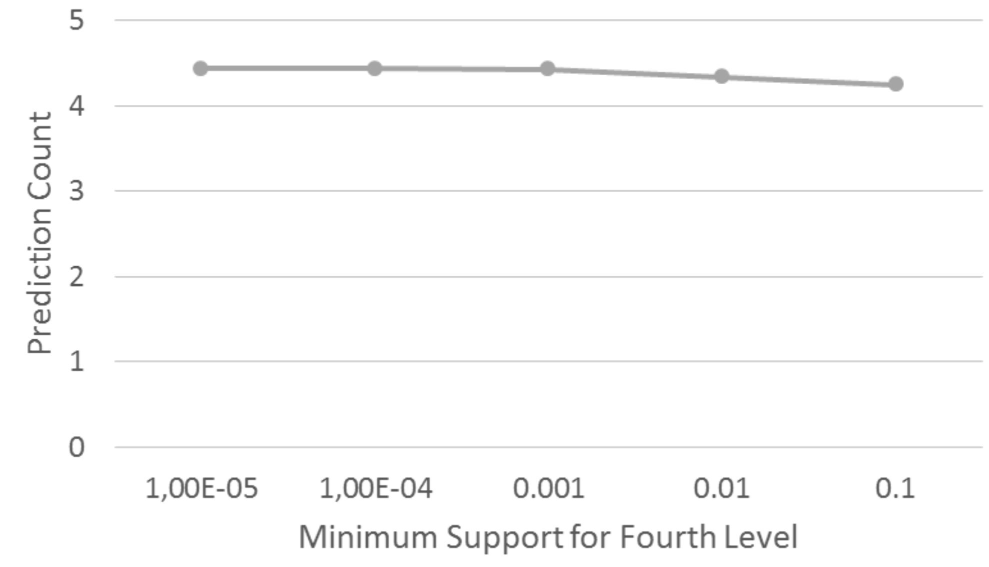

Figure 7.14: Minimum Support for Fourth Level vs Prediction Count

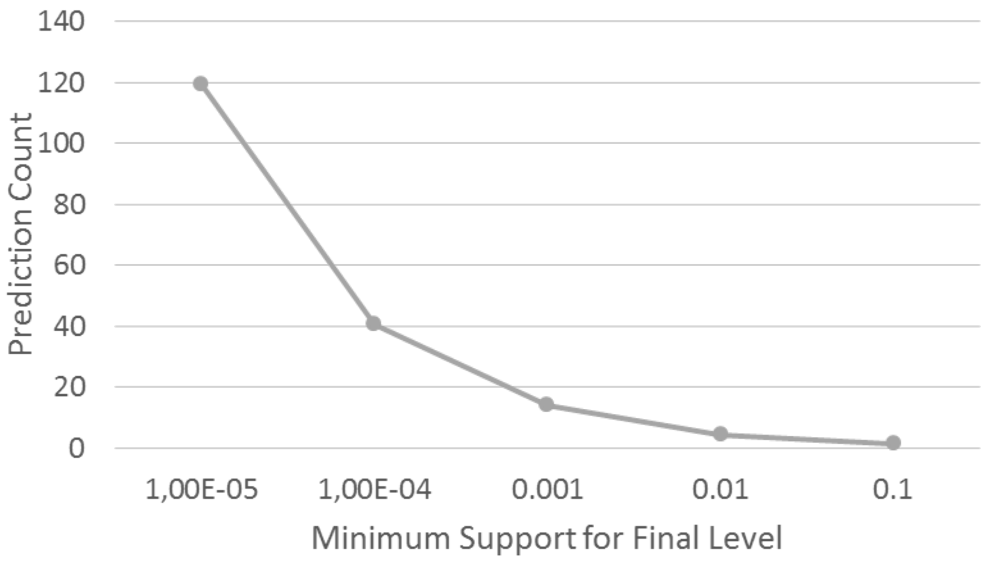

Figure 7.15: Minimum Support for Final Level vs Prediction Count

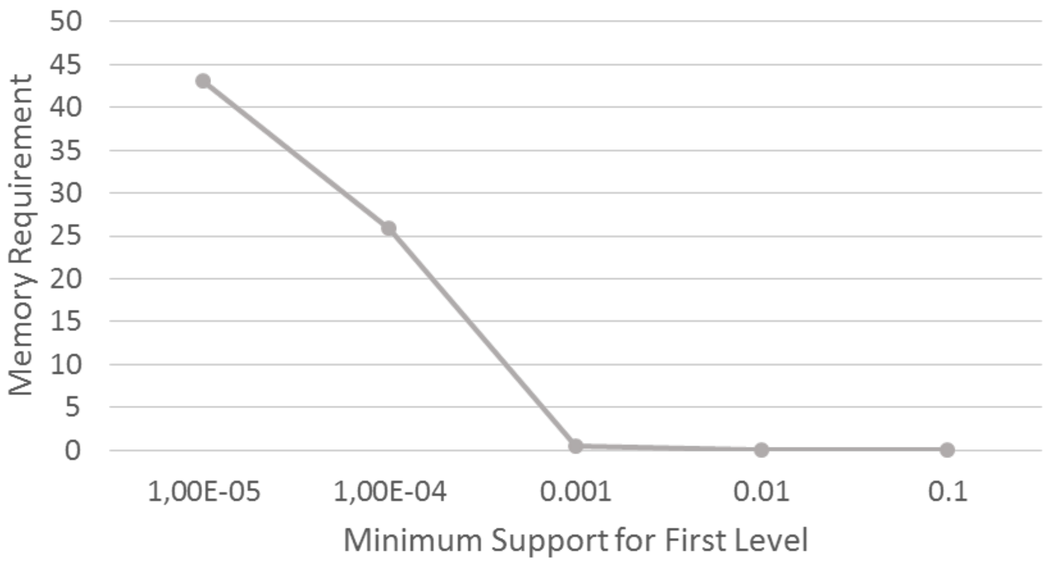

Figure 7.16: Minimum Support for First Level vs Memory Requirement

7.2.3.4 Memory Requirement

In this set of experiments, we analyzed the effect of minimum support values on memory requirement for all levels.

The amount of the drop in the memory requirement as the minimum support value increases slows down with the increase of the levels as can be seen in Figure 7.16, Figure 7.17, Figure 7.18, Figure 7.19, Figure 7.20. Especially in the first level since most sequences are pruned with high minimum support requirement, the memory requirement drops very quickly. It is also based on the fact that the minimum support values for the preceding levels have large scale effect on the successor levels. In the final level, there is almost no drop in the memory requirement. It can be observed that the line is almost horizontal, since the change in support value for this level does not have any effect on the other levels.

7.2.3.5 Comparison to AprioriAll Algorithm

In addition to the experiments mentioned above, we have also applied standard AprioriAll algorithm [1]. The main drawback of AprioriAll algorithm is the size of the

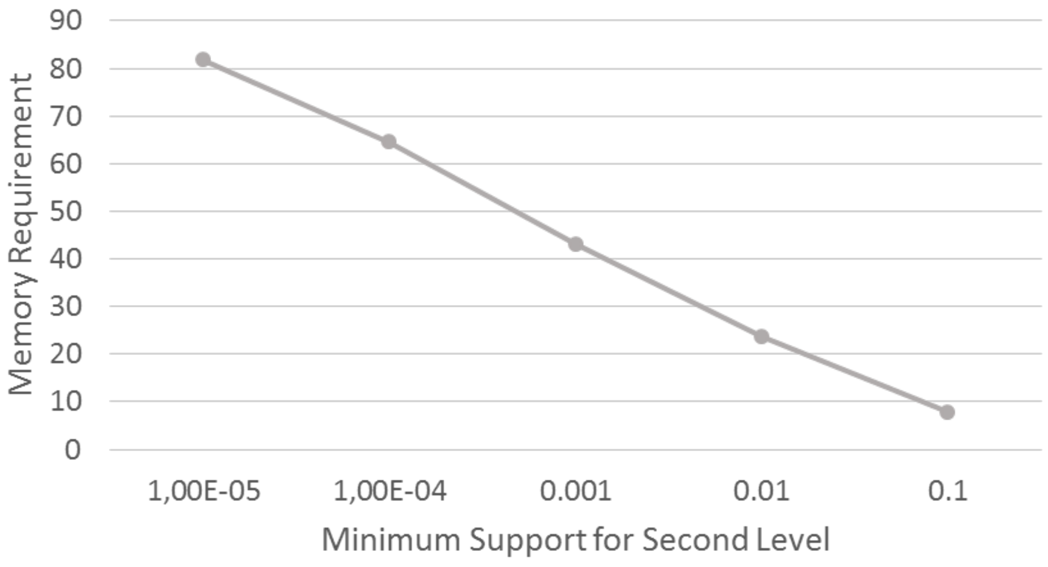

Figure 7.17: Minimum Support for Second Level vs Memory Requirement

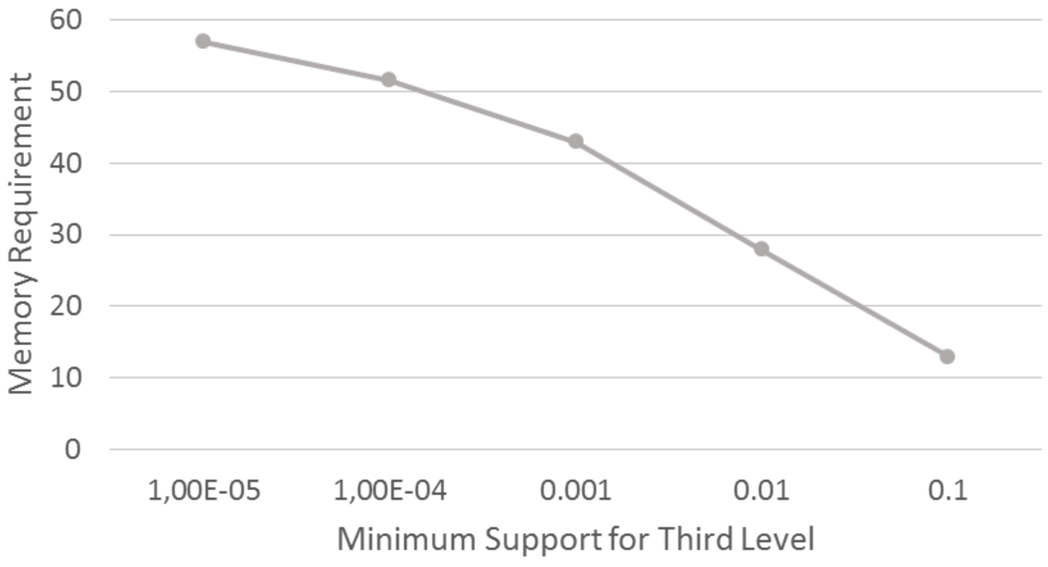

Figure 7.18: Minimum Support for Third Level vs Memory Requirement

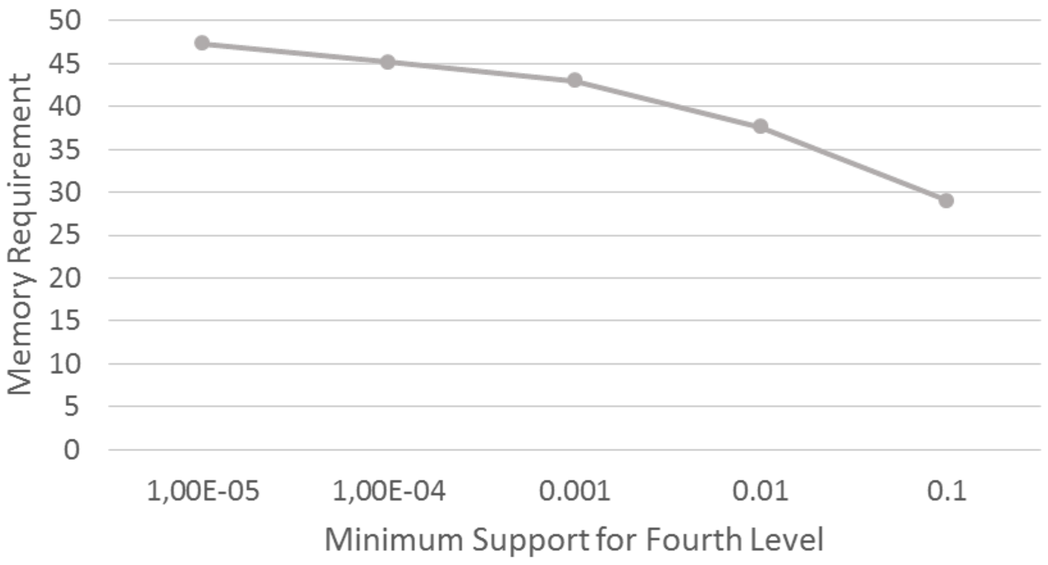

Figure 7.19: Minimum Support for Fourth Level vs Memory Requirement

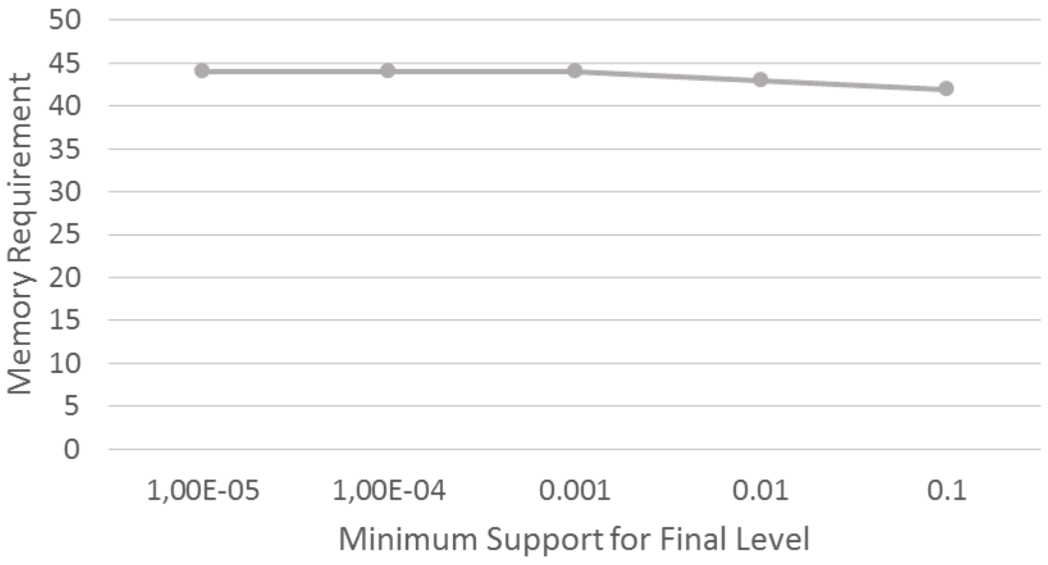

Figure 7.20: Minimum Support for Final Level vs Memory Requirement

prediction set. In order to obtain high accuracy results (g-accuracy) as in our model, the minimum support value must be chosen as a very small value (even zero), so that we can keep as much sequences as possible. However, this results in high prediction count as well as increasing the memory requirement. The accuracy obtained when no minimum support value is given is the upper bound that can be achieved with sequence matching approach. However, for that setting the memory requirement is also the maximum, since the hash-tree keeps all sequences without any pruning. As expected, this maximum accuracy can be obtained only with a very high prediction count, which is more than 133. Since this is unacceptably high, we tested AprioriAll with a non-zero, but very small minimum support value. This resulted slight decrease in the accuracy, while dropping the prediction count and the memory requirement significantly with pruning of large portion of the hash-tree. Even though the memory requirement has dropped a lot to a very good level, the decreased value of prediction count still stayed unacceptably high value, which is almost 40. Further increases in minimum support values had dropped the accuracy levels to around and below base-line levels. Therefore, they are not acceptable either. However, with ASMAMS we have achieved almost the same accuracy levels of the best and optimal AprioriAll accuracy values with a very low prediction count value, which is 4.43, with a memory requirement less than the half of the optimal (and maximal) results of AprioriAll setting. In addition to this, we have applied ASMAMS with a tolerance value 1 and we have achieved a general accuracy of 88.68 with nearly same prediction count. We have also applied ASMAMS with a tolerance value 2, however, since no prediction ratio is really low, it did not produce any improvement for our dataset. These results are summarized in Table 7.2.

Table 7.2: The results for ASMAMS and AprioriAll methods

G-Accuracy	P-Accuracy	Mem. Req.	Pred. Count	No Output Ratio	Description
88.68	89.44	44	4.42	0.8%	ASMAMS Min. Sup. List: [1e-5.1e-3.1e-3.1e-3.1e-2] Tolerance:1
85.04	93.08	44	4.43	8.6%	ASMAMS Min. Sup. List: [1e-5.1e-3.1e-3.1e-3.1e-2] Tolerance:0
51.47	88.66	0.01	1.29	41.94%	ApprioriAll Min. Sup: 1e-5
86.32	94.15	9.76	39.42	8.32%	ApprioriAll Min. Sup: 1e-8
89.82	95.38	100	133.48	5.84%	ApprioriAll Min. Sup: 0

CHAPTER 8

DISCUSSION AND CONCLUSION

In this work, we applied clustering, classification and sequential pattern mining algorithms on CDR data and we present an Apriori-based sequence mining algorithm for next location prediction of mobile phone users. The basic novelty of the proposed algorithm is a level-based support definition and the use of multiple support thresholds, each for different levels of pattern generation that corresponds to generation of sequence patterns of different lengths. The evaluation of the method is conducted on CDR data of one of the largest mobile phone operators in Turkey. The experiments compare the performance of the proposed method in terms of accuracy, prediction count and space requirement under changing thresholds for each level. Actually, these experiments serve for determination of the best minimum support list values for each level to obtain the highest accuracies, as well. We have also compared the performance with conventional method involving a single support threshold. We have observed that our method ASMAMS produces almost the optimal accuracy results with very small prediction sets, whereas the same accuracy can be obtained by AprioriAll with very low support thresholds and much larger prediction sets. Considering that there are more than 13000 different locations, the prediction sets' sizes obtained by ASMAMS, which is 4, with almost optimal accuracy can be considered as quite useful result for the mobile phone operator.

As the future work, we aim to extend this study by adding a region based hierarchy to this model in order to increase prediction accuracy.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *ICDE*, pages 3–14. IEEE Computer Society, 1995.

[2] E. Baralis and P. Garza. A lazy approach to pruning classification rules. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 35–42, 2002.

[3] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, 2002.

[4] Chiara Boldrini and Andrea Passarella. Hcmm: Modelling spatial and temporal properties of human mobility driven by users' social relationships. *Computer Communications*, 33(9):1056–1074, June 2010.

[5] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *KDD '11 Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1082–1090. ACM, 2011.

[6] Yves-Alexandre de Montjoye, César A. Hidalgo, Michel Verleysen, and Vincent D. Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific Reports*, 3(1376), March 2013.

[7] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm, 1996.

[8] Huiji Gao, Jiliang Tang, and Huan Liu. Mobile location prediction in spatio-temporal context. In *the Procedings of Mobile Data Challenge by Nokia Workshop at the Tenth International Conference on Pervasive Computing*. Nokia, June 2012.

[9] Győző Gidófalvi and Fang Dong. When and where next: individual mobility prediction. In *Proceedings of the First ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, MobiGIS '12, pages 57–64, New York, NY, USA, 2012. ACM.

[10] Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21th International Conference on Very*

Large Data Bases, VLDB ’95, pages 420–431, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[11] John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 99th edition, 1975.

[12] Ya-Han Hu and Yen-Liang Chen. Mining association rules with multiple minimum supports: A new mining algorithm and a support tuning mechanism. *Decis. Support Syst.*, 42(1):1–24, October 2006.

[13] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI’95, pages 338–345, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[14] L. Kaufman and P. Rousseeuw. *Clustering by Means of Medoids*. Reports of the Faculty of Mathematics and Informatics. Faculty of Mathematics and Informatics, 1987.

[15] Ron Kohavi. The power of decision tables. In Nada Lavrac and Stefan Wrobel, editors, *Machine Learning: ECML-95*, volume 912 of *Lecture Notes in Computer Science*, pages 174–189. Springer Berlin Heidelberg, 1995.

[16] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In *PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING*, pages 202–207. AAAI Press, 1996.

[17] Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple minimum supports. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’99, pages 337–341, New York, NY, USA, 1999. ACM.

[18] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1980.

[19] Mert Ozer, Ilkcan Keles, İsmail Hakki Toroslu, and Pinar Karagoz. Predicting the change of location of mobile phone users. In *Proceedings of the Second ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, MobiGIS ’13, pages 43–50, New York, NY, USA, 2013. ACM.

[20] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[21] S. Rajagopal, N. Srinivasan, R.B. Narayan, and X.B.C. Petit. Gps based predictive resource allocation in cellular networks. In *Networks, 2002. ICON 2002. 10th IEEE International Conference on*, pages 229–234, 2002.

[22] V. Chandra Shekhar Rao and P. Sammulal. Article: Survey on sequential pattern mining algorithms. *International Journal of Computer Applications*, 76(12):24–31, August 2013. Published by Foundation of Computer Science, New York, USA.

[23] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[24] Nguyen Thanh and Tu Minh Phuong. A gaussian mixture model for mobile location prediction. *The 9th International Conference on Advanced Communication Technology*, 2(9):914 – 919, February 2007.

[25] IsmailH. Toroslu and Murat Kantarcioglu. Mining cyclically repeated patterns. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, *Data Warehousing and Knowledge Discovery*, volume 2114 of *Lecture Notes in Computer Science*, pages 83–92. Springer Berlin Heidelberg, 2001.

[26] Ming-Cheng Tseng and Wen-Yang Lin. Mining generalized association rules with multiple minimum supports. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, *Data Warehousing and Knowledge Discovery*, volume 2114 of *Lecture Notes in Computer Science*, pages 11–20. Springer Berlin Heidelberg, 2001.

[27] Gökhan Yavas, Dimitrios Katsaros, Özgür Ulusoy, and Yannis Manolopoulos. A data mining approach for location prediction in mobile environments. *Data Knowl. Eng.*, 54(2):121–146, August 2005.

[28] Josh Jia-Ching Ying, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S. Tseng. Semantic trajectory mining for location prediction. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 34–43, New York, NY, USA, 2011. ACM.

[29] Daqiang Zhang, Athanasios V. Vasilakos, and Haoyi Xiong. Predicting location using mobile phone calls. *SIGCOMM Comput. Commun. Rev.*, 42(4):295–296, August 2012.