SELF-CONFIGURING DATA MINING FOR UBIQUITOUS COMPUTING

by
AYŞEGÜL ÇAYCI

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Sabancı University
January 2013

SELF-CONFIGURING DATA MINING FOR UBIQUITOUS COMPUTING

APPROVED BY

Assoc. Prof. Dr. Yücel Saygın (Thesis Supervisor)	.....
Assoc. Prof. Dr. Ernestina Menasalvas	.....
Asst. Prof. Dr. Gürdal Ertek	.....
Assoc. Prof. Dr. Albert Levi	.....
Assoc. Prof. Dr. Erkay Savaş	.....

DATE OF APPROVAL: .....

to my son

Acknowledgments

I would like to express my deepest gratitude to my supervisor Assoc. Prof. Dr. Yücel Saygın, whose patience and kindness, as well as his academic experience, have been invaluable to me.

My sincere gratitude goes to Dr. Ernestina Menasalvas for her constant support during all the phases of my research. Her guidance helped me in all the time of research and writing of this thesis.

I thank Assoc. Prof. Dr. Albert Levi, Asst. Prof. Dr. Gürdal Ertek, and Assoc. Prof. Dr. Erkay Savaş for their kind attendance to the thesis committee and for their valuable contributions.

Can Tunca and Engin Doğusay from Sabanci University contributed to this study by developing the supporting software. I thank them for their efforts. Later on, Ahmet Can Kan from Sabanci University ported the supporting software to Android platform. I'm grateful to him for making the necessary changes to adapt the programs to Android platform in a very short time.

I also thank to my friends in Universidad Politecnica de Madrid: Dr. Santiago Eibe, Andrea Zanda, João Bartolo Gomes. I benefited all the discussions we had done together.

Dr. Özlem Çetinoğlu provided me the formatting material for the thesis. Thanks for supporting me on formatting the thesis.

Finally, I am forever indebted to my son for his endless love, support, and patience. I am so lucky to have him.

SELF-CONFIGURING DATA MINING FOR UBIQUITOUS COMPUTING

Ayşegül Çaycı

Electronics Engineering and Computer Science

Ph.D. Thesis, 2013

Thesis Supervisor: Assoc. Prof. Dr. Yücel Saygın

Keywords: Data Mining, Ubiquitous Computing, Machine Learning

Abstract

Ubiquitous computing software needs to be autonomous so that essential decisions such as how to configure its particular execution are self-determined. Moreover, data mining serves an important role for ubiquitous computing by providing intelligence to several types of ubiquitous computing applications. Thus, automating ubiquitous data mining is also crucial. We focus on the problem of automatically configuring the execution of a ubiquitous data mining algorithm. In our solution, we generate configuration decisions in a resource-aware and context-aware manner. We propose to analyze the execution behavior of the data mining algorithm by mining its past executions. In order to extract the behavior model from algorithm's executions, we make use of two different data mining methods which are Bayesian network and decision tree classifier.

Bayesian network is constructed in order to represent the probabilistic relationships among device's resource usage, context, algorithm parameter settings and the performance of data mining.

Other data mining method that has been used is the decision tree classifier. The effects of resource and context states as well as parameter settings on the data mining quality are discovered through decision tree classifier. In this approach, a taxonomy is defined on data mining quality so that tradeoff between prediction accuracy and classification specificity of each behavior model that classifies by a different abstraction of quality, is scored for model selection.

We formally define the behavior model constituents, instantiate the approach for association rules and validate the feasibility of the two of the approaches by the experimentation.

MOBİL VERİ MADENCİLİĞİNDE OTOMATİK YAPILANDIRMA

Ayşegül Çaycı

Elektronik Mühendisliği ve Bilgisayar Bilimi

Doktora Tezi, 2013

Tez Danışmanı: Doçent Dr. Yücel Saygın

Anahtar Sözcükler: Veri Madenciliği, Mobil Sistemler, Makine Öğrenimi

Özet

Mobil cihazlarda kullanılan yazılımlar otonom olmalı ve kendilerini yapılandırmak gibi elzem kararları verebilmelidirler. Ayrıca, mobil platformlarda veri madenciliğinin çeşitli uygulamalarda daha akıllı kararlar almaları doğrultusunda kullanılmaları önemlidir. Dolayısıyla, mobil cihazlarda veri madenciliğinin de otonom olması gereklidir. Bu tezde, mobil cihazlarda veri madenciliği algoritmalarını otomatik olarak yapılandırma konusunu ele aldık. Sunulan çözümde, konfigürasyon önerileri üretilirken cihazın kaynaklarının kullanımı ve cihazın kullanıldığı bağlam göz önüne alınmıştır çünkü mobil cihazların kullanıldıkları bağlam sıkça değişmektedir ve cihazın kaynakları da genellikle kısıtlıdır. Veri madenciliği algoritmasının önceki çalıştırılışlarından işleyiş modelinin çıkarılarak yapılandırılmasında kullanılmasını önermekteyiz. Bu amaçla iki farklı yöntem denenmiştir: Bayesian network ve decision tree classifier.

Bayesian network kullanarak, cihaz kaynaklarının durumu, hangi bağlamda kullanıldığı ile veri madenciliği yapılandırma değerleri ve elde edilen performans arasındaki ilişki olasılıksal olarak gösterilmiştir. Bu bilgiye dayanarak, veri madenciliği uygulamasının ilerki çaliştırılışlarında mevcut duruma uygun yapılandırma kararları çıkarılmaktadır.

Veri madenciliği algoritmasının işleyiş modelini çıkarmakta kullandığımız diğer yöntem ise decision tree classifier'dır. Cihaz kaynaklarının kullanım durumları ve cihazın hangi bağlamda kullanıldığı ile algoritma yapılandırmasının elde edilen veri modeli kalitesine etkisi decision tree yöntemiyle sınıflandırma yapılarak araştırılmıştır. Veri modeli kalitesi hiyerarşik olarak sınıflandırılmak suretiyle elde edilen olası veri madenciliği algoritması işleyiş modellerinden en yüksek tahmin doğruluğuna sahip olup aynı zamanda en özgül sınıflandırma yapan modeli seçmek için bir yöntem önerilmiştir.

Mobil cihazlarda çalışacak bir veri madenciliği algoritması işleyiş modelini oluşturan unsurlar tanımlanmış, yöntem association rule mining algoritması için örneklenmiş ve yöntemin kullanabilirliği deneysel olarak gösterilmiştir.

Contents

List of Figures

List of Tables

List of Abbreviations

P'	Configuration of an algorithm
C'	Circumstance
Q'	Data Mining Quality
B_S	Bayesian Network Structure
P	Relation schema for parameters
P_I	Set of parameter tuples
C	Relation schema for circumstance
C_I	Set of circumstance tuples
Q	Relation schema for quality features
Q_I	Set of quality feature tuples
E	Relation schema for execution data
E_I	Set of execution data
Q_D	Relation schema for discretized quality features
Q_{D_I}	Set of discretized quality features
Q^{tuple}	Singleton set of quality features. $Q^{tuple} \subseteq Q_{D_I}$
Q_A^{tuple}	Singleton set containing an aggregated quality feature
f_A	Aggregation function
Q_A	Relation schema for aggregated data mining quality
Q_{A_I}	Set of tuples of aggregated data mining quality
Q_M	Set of data mining attributes
Q_G	Set of abstractions on data mining attributes
Q_T	Set of data mining attributes and abstractions
ρ	Partial ordering on Q_T

Q_{g_i}	Predecessor set of $g_i \in \rho(Q_T)$ derived from taxonomy
L_{set}	Set of class label attributes sets
G	Relation schema for abstractions on quality
G_I	Set of quality abstraction tuples
f_{g_i}	Mapping from predecessors set to their abstraction g_i
G^{tuple}	Singleton set of abstract quality features. $G^{tuple} \subseteq G_I$
f_{AL_i}	Aggregation function to form $i'th$ class label
Q_{AL}	Relation schema for class labels of aggregated quality
Q_{AL_I}	Set of class label tuples
S_{set}	Screened set of class label attribute sets
Q_{AS}	Relation schema for screened quality class labels
Q_{AS_I}	Set of screened class label tuples

Chapter 1

INTRODUCTION

1.1 Motivation

Ubiquitous computing turned out to be today's prominent computing paradigm as a result of the advances in related technologies, especially, wireless, mobile and sensor technologies coupled with the dissemination of these technologies in prices affordable by large masses. Another important reason for the rise of this computing paradigm, is the availability of diverse application areas which benefit ubiquitous computing. In a variety of ubiquitous computing applications such as ubiquitous health care systems, intelligent transportation systems and personal recommender systems, data mining is a preferred method for incorporating intelligence. Consequently, special consideration should be given to ubiquitous data mining which is complementary for a number of ubiquitous computing applications.

Ubiquitous computing defines an environment where resources for computing are spread rather than centralized and moreover, ubiquitous computing devices are operated most of the time by individuals who are not computer savvy and even devices lie unattended in the environment. Data mining, on the other hand, is notorious for high demand of computing resources and often requires domain experts for tuning the process. Therefore, new principles and mechanisms for mining data on a platform consisting of restricted resource devices with versatile context where the expert interaction

is not available, are needed. In that respect, the essential features of a service providing ubiquitous data mining are resource and context-awareness as well as autonomous behavior and adaptability.

1.2 Approach

We address the problem of automatic configuration of the execution of a data mining algorithm in a context and resource aware manner as a first step towards deploying an autonomous ubiquitous data mining service that adapts to changing conditions. It is important to note that, autonomous behavior of a service is a broader concept which also involves decisions about scheduling the service, prioritizing its execution and others along with automatic parameter tuning.

Cao, Gorodetsky and Mitkas ([9]) discuss the contribution of data mining to agent intelligence. They argue that a combination of autonomous agents with data mining supplied knowledge provides adaptability whereas knowledge acquisition with data mining for adaptability relies on past data (past decisions, actions, and so on). Our approach to provide adaptability is similar: we use machine learning approach in order to generate adaptable parameter setting decisions and enhance ubiquitous data mining with autonomy and adaptability.

Following list summarizes the principles our approach:

- We propose to extract what we call the behavior model of a data mining algorithm's execution for configuring its parameters and we define formally what constitutes a behavior model in a ubiquitous computing environment.
- We present a solution that is based on learning from past experiences for future configuration decisions which implies that the configuration decisions can be adapted to changing conditions.
- We aim a general-purpose solution for configuring ubiquitous data mining. Thus,

the proposed solution is not for a specific data mining algorithm.

- We propose a solution so that no restrictions are imposed on the types of the algorithm parameters when we configure by using the decision tree classifier. On the contrary, it is possible to configure continuous parameters as well as categorical.

- We analyze algorithm's execution conditions against the quality of the acquired results. For the analysis, a combination of multiple quality indicators is considered rather than individual ones and moreover the number of quality indicators may be extensive. Besides, behavior model classifies execution data on various measurements of quality indicators. Thus, a single behavior model can be used for analysis of several performance criteria on a quality indicator.

1.3 Outline of the Thesis

The organization of this thesis is as follows:

Chapter 2 introduces the problem of automatically configuring data mining in a ubiquitous computing environment while providing brief information on the methods used for the solution. Survey of related work is also provided.

In both of the approaches data collected during past executions of the data mining algorithm is used as the training set. In Chapter 3, we formally define the data model used in the approaches.

In Chapter 4 we present our approach to predict data mining algorithm behavior in ubiquitous computing environments using Bayesian Network.

The approach presented in Chapter 5 makes use of decision trees for the prediction of data mining algorithm behavior.

Instantiations of the approaches by making use of two motivating examples from the ubiquitous computing environment are given in Chapter 6.

Chapter 7 elaborates on the experimental evaluation of both of the approaches where the software designed and implemented for the experiments is also explained.

In Chapter 8, our approach is shown on mobile computing by making use of an Android device which runs one of the prominent mobile operating systems.

Chapter 9 closes the thesis with discussion and conclusion.

Chapter 2

PRELIMINARIES AND RELATED WORK

We present a mechanism to predict the appropriate settings of a data mining algorithm's parameters in a resource-aware and context-aware manner. The mechanism is based on learning from past experiences, that is, learning from the past executions of the algorithm in order to improve the future decisions.

2.1 Analysis of the Problem

Our goal is to configure automatically a data mining algorithm which will run on a ubiquitous device. Since circumstantial factors such as the conditions of the resources and the context in which the device is used are important in a ubiquitous computing environment, availability of the knowledge on the following is useful for determining the algorithm's appropriate configuration:

- the resources that the algorithm needs in order to accomplish its task,
- the algorithm parameters that have an effect on the resource usage or on the data model quality,
- the context features which may have an effect on the efficacy of the data mining model or the efficiency of data mining,

- the features of the mining data set,
- the quality indicators which show the efficacy of the data mining model and efficiency of the data mining.

On the other hand, the problem that we tackle also implies the solution to address an important issue which is to change or improve the configuration setting decisions as the circumstances change. That means that, automatically generated configuration decisions must be adapted to the changing conditions just like a data miner expert who adapts his decisions when the conditions change.

Next, we define the factors for configuration that we derive from the items outlined above

2.2 Problem Definition

When deciding how to set the parameters of an algorithm for a specific run, in a ubiquitous computing environment circumstantial factors (conditions of the device's resources and the context in which the device is in) should be taken into account as well as the required quality. For this reason, we grouped the relevant factors for the configuration as circumstance and quality. Formal definition of automatic configuration of ubiquitous data mining problem is as follows:

C': Circumstance is defined by a set of ordered pairs *(f,s)* where f is either a resource or context feature and s is the state of this feature.

Q': Quality is defined by a set of ordered pairs *(q,l)* where q is a quality feature and l is the required level for this quality. Quality features are metrics of efficiency or efficacy of the algorithm.

P': Parameter settings constituting the configuration of the algorithm is defined by a set of ordered pairs *(p,v)* where p stands for a parameter and v is the value it takes.

$\boldsymbol{f}$**:** Let C' and Q' that are defined above, be the circumstance sensed and the required quality respectively, then automatic configuration for ubiquitous data mining which is defined as P' above, is obtained by the mapping:

$$f : C' \times Q' \rightarrow P'$$

In this way, we covered all but the "features of the mining data set" outlined in problem analysis (subsection 2.1). We deliberately disregarded the effect of mining data set features on the configuration decision for the time being because we want to focus on the ubiquitous aspect in this work. On the other hand, we performed experiments to understand the effects of mining data set feature variation on the behavior model over time and discussed how to assess the deterioration of the behavior model performance.

We propose to use data mining techniques to discover configuration of a data mining algorithm (P'), aiming to attain the requested quality (Q'), for the circumstance (C') observed when a data mining request is issued. Our approach is to analyze the past behavior of algorithm under different circumstances and learn the appropriate configuration(s) for data mining which satisfies the efficiency and efficacy requested. Thus a behavior model is created by mining data collected during past executions of the algorithm. Fig. 2.1 illustrates an overall view of the approach which consists of the following basic steps:

- Collect relevant information during the execution of the algorithm,
- Maintain a collection of past execution data,
- Learn a behavior model from the past execution data, and
- Use behavior model for automatic configuration of data mining.

We proposed two approaches based on two data mining methods to solve self-configuring data mining problem. Next, brief information on the data mining methods employed is given.

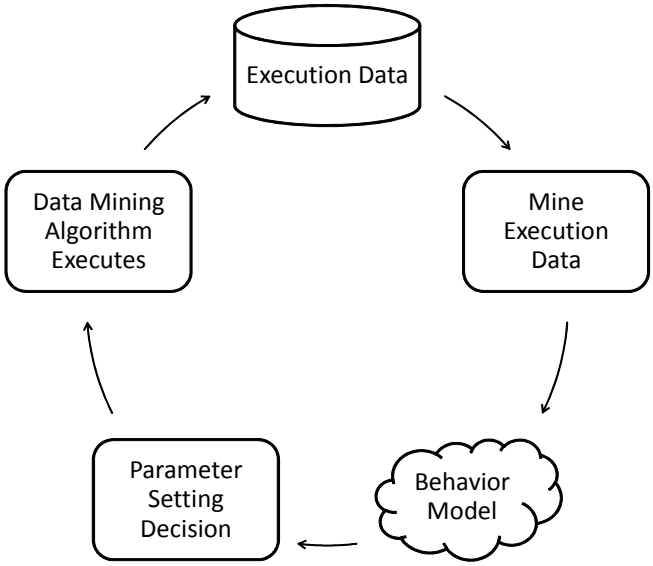


Figure 2.1: Overall view of automatic parameter setting

2.3 Bayesian Networks

Bayesian networks which represent the joint probability distributions for a set of domain variables are proved to be useful as a method of reasoning in several research areas. Medical diagnosis([4]), language understanding ([17]), network fault detection([38]) and ecology([3]) are just a few of the diverse number of application areas where Bayesian network modeling is exploited. An in depth knowledge on Bayesian networks can be found in [53].

Classification by using Bayesian networks is based on Bayes theorem which is given in Equation 2.1:

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \tag{2.1}$$

where X and H is a pair of variables, $P(X)$ and $P(H)$ are the ***probabilities*** of X and H respectively, $P(X|H)$ and $P(H|X)$ are the ***conditional probability*** of X and H

respectively.

A Bayesian network is a directed acyclic graph that shows the conditional dependencies between domain variables and may also be used to illustrate graphically the probabilistic causal relationships among domain variables. The nodes of the network represent the domain variables and an arc between two nodes (parent and child) indicates the existence of dependency among these two nodes. Conditional probabilities of the dependencies among each variable and its parents are also represented along with Bayesian networks. The joint probability of instantiated n variables (i.e. variable x_i has an assigned value) in a Bayesian network is computed by:

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i)) \tag{2.2}$$

where $parents(X_i)$ denotes the instantiated parents of the node of variable X_i.

Learning the Bayesian network structure rather than creating the structure by analyzing the dependencies of domain variables, is a field of research which was studied extensively. Algorithms that learn the structure are most useful when there is a need to construct a complex network structure or when domain knowledge does not exist as in a ubiquitous computing environment. In depth information on topics of studies related to Bayesian networks can be found in [37] whereas a survey of literature on Bayesian networks is given in [8].

2.4 Decision Tree Classification

Classification by decision tree is inquiring the properties of an instance to find out the class it belongs. For this purpose, a hierarchical structure named decision tree consisting of nodes and directed edges is used. Nodes of the tree correspond to the attributes of the instances whereas the leaf nodes constitute the class labels. Edges emanating from the non-leaf nodes are labeled by possible values or range of values of

the attribute represented by that node. It is possible to perceive each node as a test condition applied to an attribute and the edges from that node as the possible outcomes of the test.

A decision tree is built from a set of instances having preset class labels (training set) and then this structure is used to infer the unknown class labels of other instances which have the common attributes with the training set. A simple decision tree construction algorithm is to partition recursively the instances in the training set into smaller subsets by applying attribute test conditions one at a time until the class that the instances belong is found ([39]). Although the main logic of this algorithm is simple and it constitutes the basis of several decision tree algorithms, since the number of possible decision trees that can be generated from a given set of attributes is huge and moreover, some of the decision trees are more accurate than the others, successor algorithms were designed to construct a tree with reasonable accuracy without generating all the possible decision trees. While constructing a decision tree, giving precedence to the attributes that generate purer partitions with skewed class distribution is the preferred strategy. Entropy-based information gain, gini impurity and classification error are three measures for calculating the impurity of an attribute. There are a number of well-known decision tree construction algorithms that make use of this strategy. ID3 [56], its extension C4.5 [57] and CART [6] are three important examples.

Due to the advantages this method possesses, decision tree classification has been used at various domains such as medicine for disease diagnosis, finance for fraud detection, credit approval and marketing to manage campaigns. Computational inexpensiveness of decision tree construction can be counted as the foremost advantage as well as the fast classification that can be performed via a built decision tree. Furthermore, accuracy of decision tree is comparable to other classification methods and is preserved even there exist redundant attributes. Robustness to the presence of noise is yet another advantage. Decision trees are also favorable by being easy to interpret models.

2.4.1 Decision Tree Design Issues

The main objective of decision tree classification is to minimize the classification errors by avoiding the following:

- **Training errors** are incorrect classifications of training data. A possible cause is a training set where attribute combinations result in overlapping classes.
- **Overfitting** is high percentage of incorrect classifications of test data despite low training errors.

The following are among the most important issues that should be considered for generating a good model for classification.

- **Dependent attributes.** A model built by not taking into account the dependencies among the attributes of training data although related attributes exist.
- **Nonpredictive attributes.** Existence of a unique attribute such as a key or any attribute of training data that produces too many tiny partitions that is insufficient for reliable classification.
- **Plethora of classes.** The number of instances in the training set that pertain to a class is lower and less representative due to high number of different classes.

2.5 Related Work

We attempt to solve the problem of ubiquitous data mining. In that respect, our work is related to existing study in ubiquitous data mining since we also consider the resource conditions and the context when generating the configuration decisions similar to a number of studies in ubiquitous data mining. At the same time, our work bears similarities with automatic parameter configuration which is a well searched area.

Hence, related work on both of the topics are given in separate subsections below. We finalize this section by discussing the differences among our work and others.

2.5.1 Ubiquitous Data Mining

Our focus in this work is ubiquitous data mining. Therefore, we determine the essential features of ubiquitous data mining considering the characteristics of the devices where the processing will take place. Consequently, when developing a data mining service for a ubiquitous device the following need to be taken into account:

- Resource-awareness
- Context-awareness
- Autonomous behavior
- Adaptability

In this subsection, we discuss our perspective and we mention the related work on ubiquitous data mining whereas our analysis of research challenge of ubiquitous data mining can be found in [12].

Resource and Context Awareness

Resource-awareness is assessing the availability of the required resources and reacting accordingly. The aim of resource-aware data mining service is to optimize the resource usage which necessitates knowing the necessary resources, being able to measure the availability of the device's resources and knowing the effects of the resources on its processing. Ubiquitous devices may have limited resources like processor power, memory and battery. Even if there is scarcity of a resource like memory, CPU or battery in the system, a data mining service may wisely switch to an alternative algorithm than the

desired one or alter its parameter settings to optimize the usage of the scarce resource and continue to service.

A number of studies has been proposed for ubiquitous data mining in resource constrained environments. Majority of these studies apply to data stream mining techniques. The approach in [27] [28] is for mining data streams where output granularity is adapted to the data rate of the stream, available memory and time constraints. In a later study ([30]), the idea of adapting output granularity is defined within a generic framework for resource aware stream mining where input rate and data mining algorithm are also adapted in a resource aware manner. A resource aware clustering algorithm for ubiquitous data streams is proposed in [16] where the algorithm settings are adapted and stream data is compressed based on available resources so that clustering with acceptable accuracy is possible even under constrained memory. A quality aware data stream mining model in [26] is able to adapt according to output quality as well as the resource consumption patterns. Succeeding work in [45] improves the former model by assessing the quality in real time. At a recent work, a general model of resource and quality aware data stream mining is proposed in [44] where its applicability is shown by the use of an example clustering algorithm. There are also resource aware stream mining solutions that apply only to specific algorithms. For instance, a frequent itemset stream mining algorithm is presented In [15] that utilizes an adaptive memory scheme to maximize the mining accuracy for confined memory space.

Context-awareness refers to the capability of sensing the environment and reacting accordingly. Context is domain/application specific most of the time but two common context features almost always used are location and time. In this work, we will refer to context-awareness in ubiquitous data mining as the capability of the device to adjust data mining preferences depending on circumstances in order to obtain better/more accurate results or improve the efficiency of the process. Context versatility of ubiquitous computing makes possible to fine tune data mining by considering the current context states. A number of examples are appropriate in order to give insight on how context can be used for ubiquitous data mining but certainly the usage is not restricted to these

examples. For example, time of day can be a criteria on determining the amount of mining such that more time consuming mining can be preferred during night. Context indicating the urgency of the situation that induces to use an already available model rather than re-generating the results is another example.

Similar to resource-aware solutions, context-aware ubiquitous data mining were also proposed for data stream mining. Context-aware stream mining was proposed by [33] where input and output granularity as well as algorithms of data stream mining are adjusted dynamically and autonomously according to context. An approach for situation-aware adaptive processing of data streams was described in [34] and implementation for a health monitoring application was also shown. A domain specific context-aware ubiquitous stream mining model for intersection safety can be found in [58].

Autonomous and Adaptable Behavior

Autonomy and adaptability are two complementing features for a service. In general, autonomy is the ability of a service to determine independently what actions to take whereas adaptability is the ability to change the decision as the circumstances change.

A ubiquitous data mining service behaves autonomously if whenever a mining request is received; all the decisions about the mining process are taken independently by the service. Simply put, the decision is a set of actions to perform against the current situation. Setting a parameter value of data mining algorithm or selecting the appropriate input to mine are two examples of the actions. Context or availability of the resources which data mining service need may constitute a situation. The decisions in an adaptable ubiquitous data mining service, on the other hand, are dynamic and are expected to improve in terms of achieving the goals by learning from experience. Existent work on autonomous ubiquitous data mining focus on determining the parameters of data mining algorithm either by statically binding situations (e.g., [29]) or dynamically determining the actions by correlation functions [34].

In [11], we proposed a ubiquitous data mining service and in [10], a mechanism

for self-configuration of ubiquitous data mining aiming to fulfil the aforementioned requirements.

2.5.2 Automatic Parameter Configuration

We propose a method to automatically determine the configuration of a data mining algorithm to execute in a ubiquitous computing device. Algorithm selection, configuration setting or parameter tuning are similar research areas where similar solutions are proposed for the automatization. Present work on the automatization of either of them aims to speed up the process or to increase the probability of finding a solution to a specific problem instance. We emphasize automatic parameter tuning to provide autonomy in ubiquitous computing environments and it is important to use context and situation information when deciding and decisions need to be adaptive.

Taxonomy of varying approaches for solving the algorithm selection, configuration setting or parameter tuning problems is presented in several of the present research on algorithm selection, configuration setting and parameter tuning. Since existing taxonomies are important resources for determining the lacking work of the research area, we provide a summary of the categories defined for automatizing algorithm selection/ configuration setting/parameter tuning by different authors before presenting our taxonomy. In [40], three approaches are stated by taking into consideration the target problem. First approach they defined aims to find the *best default configuration across a set of given instances*. Some of the mentioned related work of this approach involve racing algorithm, ILS search, fractional experiment design together with local search and decision tree classification. Second approach is defined as solving the algorithm selection problem which is selecting *the most appropriate algorithm given a problem instance*. Usage of algorithm portfolios to choose among several algorithms is the general term where usage of empirical hardness model or case base reasoning are two example solutions given. Third approach in their categories is the online approach which is online in the sense that within a group of solutions *alternation between different problem solving strategies during execution* is possible. In this approach tuning of parameters

and even algorithm selection decision is dynamically adjusted during execution. Examples supplied that belong to that category employ a learning mechanism which is reinforcement learning most of the time, to improve the settings done and algorithm choice based on the information collected from the previous phases of the execution.

Another source of taxonomy for automatic parameter setting approaches is given in [52] where they distinguished the following approaches: evolutionary algorithm run, model selection and statistical estimation. In an evolutionary algorithm run which apply to evolutionary algorithms, programs and strategies, parameters are adapted during execution to obtain the best default configuration. In model selection, optimization is performed based on a (usually) single objective to determine the best parameter setting among the existing models. Statistical approach is defined as the estimation of the parameter setting using one of the estimation methods such as maximum likelihood (ML), expectation maximization (EM), maximum a posteriori (MAP) or hidden markov model (HMM).

In [31], the existent parameter tuning or algorithm selection techniques are discriminated using a number of orthogonal features: depending on the interval that tuning/selection is performed (performed once for a set of problem instances or repeated for each instance), whether the decision is made statically before execution or determined dynamically during execution and whether the learning technique is offline (a separate training phase) or online (criteria is updated on every instance solution).

We distinguished two alternatives disciplines in the literature which are dominantly used to handle the parameter setting problem. These disciplines are (combinatorial) optimization methods and machine learning methods. Parameter tuning by optimization is a well searched area where proposed optimizations either tune parameters of a specific algorithm or provide optimizations to general cases.

Only a brief list of representative optimization solutions to parameter tuning is compiled below as our work deviate a lot from them due to our preference of a machine learning technique for automatizing parameter tuning. The main idea behind optimiza-

tion is to determine performance criteria to be optimized and find the configuration that satisfies best this criteria. A racing algorithm by [5] for configuring metaheuristics, iterated local search approach by [41] for configuration determination of an algorithm, a dynamic and online algorithm selection based on algorithm portfolios paradigm by [31], experimental design combined with local search to fine tune parameters of an algorithm by [1], are examples which employ an optimization technique.

Other prevailing technique proposed for automatic parameter tuning is based on machine learning classifiers. In general terms, classifiers are used to learn the parameters to set the configuration. In [59], usage of decision trees for automatic tuning of search algorithms is suggested. They describe both an online version where training data is not available and offline version in which training data is used for their method. In the offline version, a J48 decision tree using Weka is constructed to classify training data. The nodes of the tree are parameters of the algorithm, the branches from each node correspond to different values that parameter may take and the value on the leaf node classify the group of parameters as positive or negative with respect to a performance criteria. In their experiments runtime of the algorithm is used as the performance criteria. In order to derive candidate parameter settings, ranking functions are applied to the part of the decision tree which end to leaf nodes having positive values.

Bayesian networks are used by [52] to automatize the parameter tuning process. They distinguish two types of algorithm parameters as external and internal such that the former are the ones that must be tuned and the latter are established and updated in the model learning process. The "adjustment" model that they propose recommends values for the external parameters after the learning and inference phase. In the learning phase, Bayesian network is constructed from the data collected on previous runs. The domain variables are parameters of the algorithm and some efficiency measures. The inference mechanism updates the probability tables and obtains the most probable parameter values to obtain a "good" result from the algorithm.

In [51], the adjustment model is enhanced by a combined case-based reasoning system with the argument that optimal performance in different problem domains is

attained by different parameter settings. A case base which contains the Bayesian networks from the adjustment model and the characteristics of their associated problems is used for finding the similar problems of the domain. Similarities among problems are calculated using Euclidean distance function.

2.5.3 Characteristics Differentiating Our Approach

Existing resource-aware and/or context-aware adjustments of ubiquitous data mining parameters are proposed for data streams where data arrive continuously in a rapid speed. Hence, proposed solutions are specific to data stream mining and some are applicable to data stream mining algorithms with certain characteristics. On the other hand, we anticipated that all types of data mining will be required by ubiquitous computing applications. For example, mining multi-media data on the mobile device for the organization of music, picture and video files is one potential application area of ubiquitous data mining while data is not in streams ([47],[49]). Similarly, there are other prospective ubiquitous computing application areas such as user profiling ([32]), activity planning ([46]) and personal health monitoring ([19]) where there is a need to apply machine learning or mining techniques on data which is not streaming but batch. Thus, we worked on a general purpose solution to automatize the configuration of any data mining algorithm running on a ubiquitous computing environment without imposing any restrictions on the type of data mining algorithm or parameters.

The approach which we use for determining the configuration of data mining is also quite different from the work mentioned in the subsection 2.5.1 such that we employ data mining to discover the appropriate parameter settings from the history of execution results whereas proposed resource/context aware stream mining techniques do not use data mining methods to adjust stream mining parameters. The reasons we use a data mining technique for generating configuration decisions are twofold: to discover the effects of algorithm's parameters to the quality of its results and to be able to adapt the configuration decisions to the changing conditions. In our solution, configuration decisions are adaptable in the sense that if there is a change on the discovered effects

due to a factor such as the growth of the data set which algorithm to be configured mines, new parameter to quality effects can be tracked by regenerating or updating the behavior model.

Decision tree classifiers were suggested for automatic parameter setting like us in [59]. On the other hand, the method they suggested for automatically setting the parameters of an algorithm lacks being a general purpose solution due to following:

- Classification can only be made on a single quality (performance) criteria. It is not possible to classify by combination of quality criteria.
- Different parameter settings were classified into positive and negative examples with respect to a performance criteria which implies that only two levels of quality can be assessed.
- Assignment of class labels is static, quality attained after running the algorithm by the derived parameter settings is not used to correct the class labels.
- A new model would be needed when the performance criteria changes.
- Moreover, the suggested method is only for search-based algorithms.

Chapter 3

DATA MODEL

Behavior model generation process uses data collected during past executions of the data mining algorithm in order to learn its behavior. In each execution of the algorithm, data specific to this run is captured and stored. This execution data is mined to construct behavior model. Since we have used execution data to construct Bayesian network and also to build decision tree, we formally define execution data before elaborating on either of the approaches used for self-configuring data mining:

Definition 1 *Let* $P(p_1 : D_1, ..., p_n : D_n)$ *be a relation schema defining a data mining algorithm's parameters* p_i*, where* $1 \leq i \leq n$*. Let* dom_i *be the set of values associated with the domain named* D_i*.*
An instance of P *that satisfies the domain constraints is a set of tuples with n fields:* $P_I = \{< p_1 : d_1, ..., p_n : d_n > | d_1 \in dom_1, ..., d_n \in dom_n\}$

Definition 2 *Let* $C(c_1 : D_1, ..., c_n : D_n)$ *be a relation schema defining context and resource features (circumstance),* c_i*, where* $1 \leq i \leq n$*. Let* dom_i *be the set of values associated with the domain named* D_i*.*
An instance of C *that satisfies the domain constraints is a set of tuples with n fields:*

$C_I = \{< c_1 : d_1, ..., c_n : d_n > | d_1 \in dom_1, ..., d_n \in dom_n\}$

Definition 3 *Let $Q(q_1 : D_1, ..., q_n : D_n)$ be a relation schema defining quality features, q_i, where $1 \leq i \leq n$. Let dom_i be the set of values associated with the domain named D_i.*
An instance of Q that satisfies the domain constraints is a set of tuples with n fields:
$Q_I = \{< q_1 : d_1, ..., q_n : d_n > | d_1 \in dom_1, ..., d_n \in dom_n\}$

Definition 4 *Let $E(a_1 : D_1, ..., a_n : D_n)$ define a relation schema for execution related data. An instance of execution data E, named E_I, is the subset of the Cartesian product (cross product) of the instances P_I, C_I, Q_I:*
$E_I \subset P_I \times C_I \times Q_I$

In Table 3.1, sample relational schemas for C, P, and Q together with small set of tuples as instantiations of each are given. For the given example, we assume that circumstance components (C) which may have an impact for the configuration decision of data mining are *location* of the device and the *time* of day when the data mining is requested as well as the free *memory* in the device. A number of possible circumstances are sampled in the set C_I such that each tuple in C_I has a *location*, a *time* and a *memory* value chosen from *ldom*, *tdom* and *mdom* respectively. We based our examples on association rule mining throughout the thesis for the coherence of explanations. On the other hand, we propose general guidelines for configuring any data mining algorithm. For this purpose, we exemplify in Table 3.1, k-means clustering as well as association rule mining as the data mining algorithms to be configured. We assume that association rule mining algorithm (ARM) that we configure has minimum support and minimum confidence parameters whereas number of clusters, maximum number of iterations and seed which is the number to be used for initial assignment of instances to clusters are the parameters of k-means. Memory usage (*memusg*) and the run time (*duration*)

Table 3.1: Sample C, P and Q

		Relational Schema	Domain
	C	($location : ldom$,	$ldom = \{indoor, outdoor\}$
		$time : tdom$,	$tdom = \{sunset, midday, night\}$
		$memory : mdom$)	$mdom = \{x \mid 0 < x \leq MAXMEM\}$
	C_I	{ $< location : indoor, time : midday, memory : 500M >$,	
		$< location : outdoor, time : sunset, memory : 10K >$,	
		$< location : outdoor, time : night, memory : 1G >$ }	
ARM	P	($minsupp : sdom$,	$sdom = \{x \mid 0.3 < x \leq 1\}$
		$minconf : cdom$)	$cdom = \{x \mid 0.6 < x \leq 1\}$
	P_I	{ $< minsupp : 0.5, minconf : 0.8 >$,	
		$< minsupp : 0.5, minconf : 0.9 >$,	
		$< minsupp : 0.5, minconf : 0.95 >$,	
		$< minsupp : 0.6, minconf : 0.7 >$ }	
	Q	($memusg : udom$,	$udom = \{x \mid 0 < x \leq MAXMEM\}$
		$duration : ddom$,	$ddom = \{x \mid 0 < x \leq 1440\}$
		$model : odom$)	$odom = \{strong, weak\}$
	Q_I	{ $< memusg : 5K, duration : 10, model : strong >$,	
		$< memusg : 730K, duration : 3, model : weak >$,	
		$< memusg : 200M, duration : 125, model : strong >$ }	
K-means	P	($numClust : Cdom$,	$Cdom = \{x \mid 1 < x \leq 30\}$
		$seed : edom$)	$edom = \{10, 15, 20, 25, 30\}$
		$maxIter : idom$)	$idom = \{x \mid 1 < x \leq 50\}$
	P_I	{ $< numClust : 5, seed : 10, maxIter : 5 >$,	
		$< numClust : 5, seed : 15, maxIter : 5 >$,	
		$< numClust : 5, seed : 20, maxIter : 5 >$,	
		$< numClust : 6, seed : 15, maxIter : 5 >$ }	
	Q	($memusg : udom$,	$udom = \{x \mid 0 < x \leq MAXMEM\}$
		$duration : ddom$,	$ddom = \{x \mid 0 < x \leq 1440\}$
		$WCSS : wdom$)	$wdom = \{high, low\}$
	Q_I	{ $< memusg : 5K, duration : 10, WCSS : high >$,	
		$< memusg : 730K, duration : 3, WCSS : low >$,	
		$< memusg : 200M, duration : 125, WCSS : high >$ }	

of data mining are assumed to be the common quality metrics for both data mining. Interestingness degree of the model ($model$) and within-cluster sum of squares ($WCSS$) are the data mining quality metrics of ARM and k-means respectively.

Chapter 4

SELF-CONFIGURATION USING BAYESIAN NETWORK

4.1 Behavior Model in the Form of Bayesian Network

A Bayesian network is learned from the execution data and, afterwards, this Bayesian network representing the behavior model, is used to predict the appropriate configurations for the algorithm. Fig. 4.1 illustrates Bayesian network construction steps that we propose. K2 algorithm by [21] was used when constructing the Bayesian network. A comprehensive explanation of the method proposed in the papers [20],[21] can be found at Appendix A. Initial step of behavior model generation is to discretize the execution data since K2 assumes database variables to be discrete. K2 learns Bayesian network structure from database of cases (E in our case) by determining the most probable network structure B_s given E:

$$\max_{B_s}[P(B_s|E)] \tag{4.1}$$

We made use of the open source code of Weka software ([35]) to construct the network and updated it to fit our needs. The original algorithm seeks relationships among all the variables. However, execution data has three groups of variables (C,Q,P) and the relationships among the variables within a group such as the relationship among

circumstantial variables *location* and *memory available* are not interesting. For this reason, modification of the K2 algorithm is necessary to look for relationships among nodes belonging to different groups of variables. A level (lvl) is assigned to each group based on the possible cause-effect relationship between them. The levels are used to prevent the nodes in the lower level groups to be the parents of the nodes in the upper level groups. Let $V = \{v_{l,k} | l = 1, ..., lvl \text{ and } k = 1, ..., x_l\}$ be the set of nodes of Bayesian network where x_l is the number of nodes in level l and $v_{i,j}, v_{m,n} \in V$. Then, i) nodes $v_{i,j}$ and $v_{m,n}$ can not be related if $i = m$, ii) $v_{i,j}$ can be the immediate parent node of $v_{m,n}$ only if $m = i + 1$.

The Bayesian network that is constructed from past execution data represents the probabilistic relationship between circumstance states, discretized possible parameter settings, and measured as well as discretized quality indicators. Appropriate setting of an algorithm's parameter is extracted from the Bayesian network as explained in the next section.

4.2 Mechanism to Predict Ubiquitous Data Mining Configuration

Once the behavior model is built, the steps that lead to automatic parameter configuration are as follows (See Fig. 4.1 for details):

- A data mining model is needed for a specific data set,
- Current circumstance (C') is observed and the quality requirements (Q') are acquired,
- Configuration (P') of the data mining algorithm is determined autonomously by inferencing from the behavior model

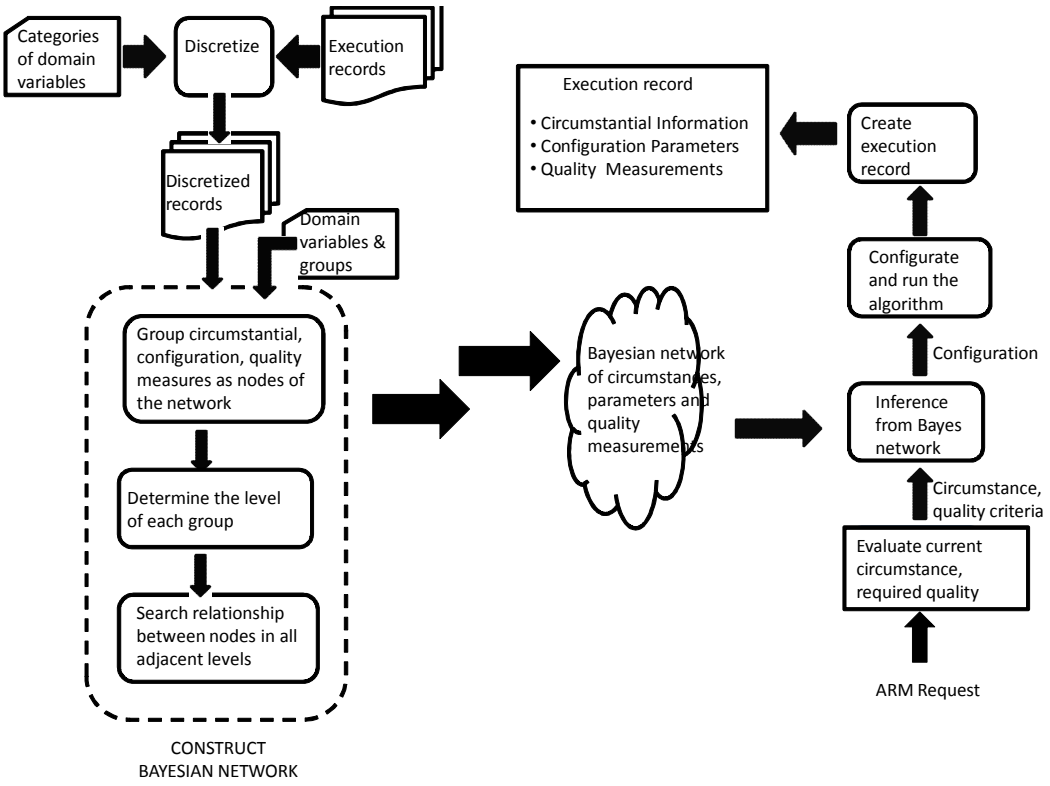


Figure 4.1: Data mining configuration using Bayesian network

The most likely configuration is inferred from the behavior model by estimating the probabilities of possible parameter settings from previous runs of the algorithm in execution circumstances similar to current in order to obtain quality levels similar to the required. In particular, $p(x|F)$ which is the conditional probability of x (an instantiation of parameter variable) given F (instantiations of circumstance and quality variables), is evaluated. Formal definition of inferring a value of a parameter from a Bayesian network structure is as follows:

Definition 5 *Consider C_I and Q_I defined in Definition 2 and Definition 3 respectively. Let c^{tuple} be any single tuple from C_I and q^{tuple} be the associated quality tuple from Q_I. Consider the relation schema $P(p_1 : D_1, ..., p_n : D_n)$ that defines data mining algorithm's parameters(Definition 1). Let $dom_i = \{x_1, x_2, ...\}$ be the domain set of p_i. The most likely setting of p_i by a value from dom_i under the circumstance c^{tuple} in order to attain the data mining quality q^{tuple} is the maximum of the φ calculated by:*
$\varphi_k = Probability(p_i = x_k \mid c^{tuple}, q^{tuple})\ \forall k \leqslant |dom_i|$

Chapter 5

SELF-CONFIGURATION USING DECISION TREES

5.1 Modeling the Behavior of a Data Mining Algorithm with Classifiers

Predictive data mining is discovering from training data, patterns that can be generalized to forecast explicit values. Since, our approach for predicting future parameter settings is learning a model from past executions of the algorithm, we have chosen predictive data mining as the appropriate technique for discovering configurations.

Classification is a predictive data mining technique where a training set is used for discovering patterns to predict categorical values. We propose to use classification of execution data, E given in Definition 4 to create the behavior model of the data mining algorithm with the aim to use the model for predictive analysis of the algorithm's behavior. Thus past execution data of the algorithm is used as the training data required for supervised learning of classification methods.

Efficiency of the data mining process and/or efficacy of data mining model, which will be referred as data mining quality thereafter, are the objectives of parameter settings for a particular execution of a data mining algorithm. For that reason, we analyze

under different circumstances the effect of parameter settings on the data mining quality and thereupon we determine data mining quality as the class label to be predicted.

We will first elaborate on the properties of the class label chosen while discussing the necessary transformations and later explain in detail behavior model construction by using a specific classifier, decision tree. We have chosen decision trees classifiers due to the following reasons: i) Behavior model is constructed on a ubiquitous computing device where lowest resource consumption is essential. Existence of several computationally inexpensive and fast decision tree construction algorithms makes decision tree classifier a suitable choice. ii) Data mining to be configured may have any kind of parameters. Decision trees can deal with continuous data as well as categorical data so that every kind of data mining parameters can be configured. iii) In general, accuracy of decision trees is comparable to other classification techniques. iv) It is possible to extract classification rules from decision trees which provide a convenient way to infer configurations.

5.1.1 Data Mining Quality as the Class Label

Since we have determined to use classifiers for solving automatic parameter setting problem, data mining quality attributes (each q_i in Definition 3) are converted to categorical attributes. Formal definition of discretized data mining quality Q is as follows:

Definition 6 *Let $Q_D(q_1 : D_1, ..., q_n : D_n)$ be a relation schema defining quality features, q_i, where $1 \leq i \leq n$. Let dom_i be the set of pairs (l, u) associated with the domain named D_i such that each pair corresponds to the lower and upper boundaries of a bin interval after discretization.*
An instance of Q_D that satisfies the domain constraints is a set of tuples with n fields: $Q_{D_I} = \{< q_1 : (l, u)_1, ..., q_n : (l, u)_n > | (l, u)_1 \in dom_1, ..., (l, u)_n \in dom_n\}$

In order to use data mining quality as the class label of the classifier, Q given in Definition 6 is converted to a unary relation having a single attribute (say q_A). Next, we define the aggregation function to derive aggregated data mining quality. The aggregation function, f_A that will be used for this purpose may consist of arbitrary operations given that a single value, q_A is obtained by making use of all other quality attributes $q_1, , q_n$ and f_A should be an invertible function so that $q_1, , q_n$ could be re-generated given q_A:

Definition 7 *Let Q^{tuple} be a set containing any single tuple from Q_{D_I}. Let Q_A^{tuple} be a singleton set containing a unary tuple. Aggregation function for data mining quality, f_A is an invertible function that defines the mapping from Q^{tuple} to Q_A^{tuple} given as:*
$f_A : Q^{tuple} \longrightarrow Q_A^{tuple}$

Finally, formal definition of aggregated data mining quality is as follows:

Definition 8 *Let $Q_A(q_A : D_A)$ define a relation schema for aggregated data mining quality and $dom_A = R_{f_A}$ is the set of values associated with the domain named D_A. An instance of aggregated data mining quality, Q_A that satisfies the domain constraints is a set of tuples with 1 field:*
$Q_{A_I} = \{< q_A : d_A > | d_A \in dom_A\}$

5.2 Predicting the Behavior of a Data Mining Algorithm with Decision Trees

We propose to use decision tree classifier to obtain a model that maps the attribute sets consisting of circumstance (Definition 2) and parameters (Definition 1) to the class

label aggregated data mining quality (Definition 8):

$$f : C \times P \longrightarrow Q_A$$

Since aggregated data mining quality (Q_A) is a composite attribute formed by aggregation of a number of attributes, the number of possible data mining quality classes is denoted with the following equation:

$$\kappa = \prod_{i=1}^{n} k_i \tag{5.1}$$

where n is the number of attributes in Q_D, and k_i is the cardinality of ith attribute's domain (number of bins).

Although classifying by Q_A will provide classes with exact data mining quality information, the resulting number of aggregated quality classes (given in equation 5.1) can be too high preventing accurate classification. For this reason, we consider abstractions of Q_A as well as Q_A as possible class label attributes. We aim to find a tradeoff between estimated accuracy and classification specificity by ranking the possible behavior models that can be generated using different abstractions of data mining quality as the class label attribute.

5.2.1 Abstractions over the Class Label

We consider different abstraction levels of data mining quality as possible class labels. A hierarchical structure that shows the taxonomy of data mining quality attributes in Q_D is used to abstract the data mining quality:

Definition 9 *Data mining quality abstraction is composed of:*

- *A tree structure T representing data mining quality taxonomy where Q_T is the node set of T and data mining quality attributes $Q_D \subset Q_T$ are the leaf nodes. Let*

Table 5.1: Relation schema: discretized data mining quality

	Relational Schema	Domain
Q_D	($avg\_mem : adom$,	$adom = \{(0, 100000), (100001, 1000000), (1000001, 10000000)$
	$max\_mem : mdom$,	$mdom = \{(0, 250000), (250001, 4000000),$
	$prc\_cycles : cdom$,	$(4000001, 10000000)\}$
	$\%\_prc : pdom$,	$cdom = \{(0, 200K), (200K, 4M), (4M, 10M), (10M, 20M)$
	$battery\_usg : bdom$,	$pdom = \{(0, 45), (46, 80), (81, 100)\}$
	$support : sdom$,	$bdom = \{(0, 25), (26, 100)\}$
	$confidence : fdom$)	$sdom = \{(0, 0.50), (0.51, 0.80), (0.81, 1)\}$
		$fdom = \{(0, 0.89), (0.9, 1)\}$

Mappings to higher levels of abstractions:	***Domains of abstract data mining quality:***
$Q_{Processor} = \{prc\_cycles, \%\_prc\}$	$Memorydom = \{VeryLow, Low, Average,$
$cdom \times pdom \rightarrow Processordom$	$High, VeryHigh\}$
$Q_{Memory} = \{avg\_mem, max\_mem\}$	$Processordom = \{VeryLow, Low,$
$adom \times mdom \rightarrow Memorydom$	$Average, High, VeryHigh\}$
$Q_{Resource} = \{Processor, Memory, battery\_usg\}$	$Resourcedom = \{Low, Average,$
$Processordom \times Memorydom \times bdom \rightarrow Resourcedom$	$High, VeryHigh\}$
$Q_{Model} = \{support, confidence\}$	$Modeldom = \{Low, Average, High\}$
$sdom \times cdom \rightarrow Modeldom$	$Overalldom = \{Good, Bad\}$
$Q_{Overall} = \{Model, Resource\}$	
$Modeldom \times Resourcedom \rightarrow Overalldom$	

$Q_G = \{g_1, g_2, ...\} = Q_T - Q_D$ *be the set of abstract data mining quality attributes.* Q_G *is partially ordered such that a quality attribute in* Q_G *comes before its parent in* T*.*

- *Domain sets* $g_i dom$ *of each* $g_i \in Q_G$*.*

- *Stepwise mappings to higher abstract levels.*
  For each $g_i \in Q_G$ *where* $i = 1, ..., \mid Q_G \mid$*:*

  - *Let* Q_{g_i} *be the successor set of* g_i *in* T*.*

  - *Every combination of elements from the domain sets of* Q_{g_i} *is mapped to the domain values of* g_i *such that:*
    $f_{g_i} : q_1 dom \times q_2 dom \times .... \times q_{|Q_{g_i}|} dom \rightarrow g_i dom$
    where $q_i dom$ *is the domain set of i'th member of* Q_{g_i}*.*

Data mining quality abstraction given in Definition 9 is explained by the following example. Discretized data mining quality schema, Q_D in Table 5.1 is used in the example to define usage measurements of device's resources such as memory (avg_mem, max_mem), processor ($prc\_cycles, \%\_prc$) and battery($battery\_usg$) by the data mining process as well as the calculations obtained from the data mining model such as

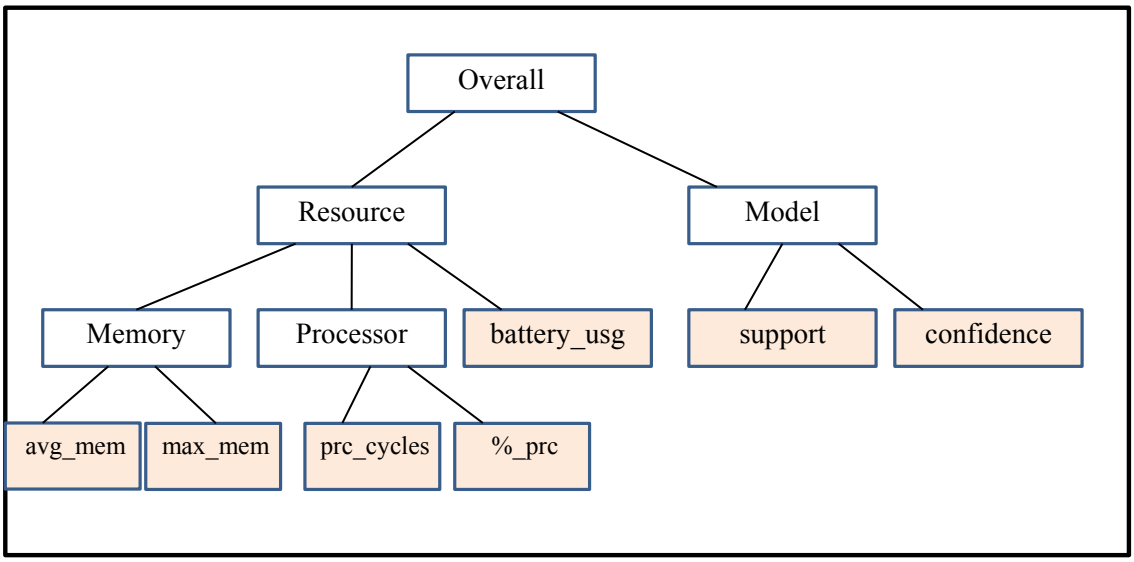


Figure 5.1: Data mining quality taxonomy specific to association rule mining

confidence and *support*. Figure 5.1 is the data mining quality taxonomy where the leaf nodes are the "actual" data mining quality features (Q_D) whereas interior nodes are the quality abstractions (Q_G).

The domains ($g_i dom$) of abstract data mining quality features which are the generalizations of the *Memory*, *Processor* and *Resource* usage as well as the data mining *Model* and *Overall* quality are shown in Table 5.1. Successor sets of abstract data mining quality features ($Q_{Processor}$, Q_{Memory} and so on) are derived from the taxonomy T according to Definition 9. The values of the features in its successor set determine the value of abstract feature. For this reason, each combination of values from the domains of the features in the successor set of an abstract feature is mapped to a value in its domain. For example, when *average memory usage* and *maximum memory usage* are in the range *(0,100000)* and *(250001,4000000)* respectively, then *Memory usage* of the process is *Average*, is a possible mapping that gives the value of an abstract data mining quality feature based on the quality features in its successor set. The appearance order of successor sets in Table 5.1 follow the partial order that is determined from the taxonomy T.

Next, we will use the relational schemas, relations and functions that are defined to establish a method for constructing adequate behavior model(s) for algorithm configurations.

5.2.2 The AS/BM Strategy

We propose a strategy that we call AS/BM (Accuracy Specificity Balanced Behavior Model Selection) in which we enumerate the alternative abstraction levels of Q_A as possible class label attributes and then evaluate the classification specificity and accuracy of the models that are created as a result of classifications by alternative abstraction levels. AS/BM has the following phases:

ENUM : Enumerate possible class label attributes based on data mining quality taxonomy.

SCRN : Apply a pre-screening to possible class label attributes for elimination of inappropriate ones for classifications.

CONS : Construct a separate model in the form of decision tree by using each enumerated class label attribute that passes pre-screening.

EVAL : Evaluate the performance of the models by observing the accuracy.

MSEL : Select the most appropriate model by taking into account accuracy and specificity of classification provided by the models.

We first ENUMerate the class label attributes sets and obtain L_{set}.

Definition 10 *Given a data mining quality taxonomy (T) and successor sets for abstract quality attributes (Q_{g_i}), $L_{set} = \{l_1, l_2, ...\}$ which is the set of class label attributes sets enumerated from data mining quality taxonomy (T) is obtained as follows:*

1. *Initially $L_{set} = \{Q_D\}$ and $O_{set} = \{Q_D\}$.*

Table 5.2: L_{set}: set of possible class label attribute sets

$\{\{avgmem, maxmem, prccycles, \%prc, batteryusg, conf, support\}$,
$\{\textbf{Memory}, prccycles, \%prc, batteryusg, conf, support\}$, $\{avgmem, maxmem, \textbf{Processor}, batteryusg, conf, support\}$, $\{avgmem, maxmem, prccycles, \%prc, batteryusg, \textbf{Model}\}$,
$\{Memory, \textbf{Processor}, batteryusg, conf, support\}$, $\{Memory, prccycles, \%prc, batteryusg, \textbf{Model}\}$, $\{avgmem, maxmem, \textbf{Processor}, batteryusg, Model\}$,
$\{\textbf{Resource}, conf, support\}$, $\{Memory, Processor, batteryusg, \textbf{Model}\}$,
$\{Resource, \textbf{Model}\}$,
$\{\textbf{Overall}\}\}$

2. *Repeat* $a - c$ *below until* $\mid O_{set} \mid = 0$

   (a) *Repeat for each* o_k *in* O_{set} *(where* $k = 1, .., \mid O_{set} \mid$*),*

      i. *form a new class label attributes set by replacing successors of an abstract quality attribute with itself. e.g.* $\{a, q_3, ...\}$ *is formed from* $o_k = \{q_1, q_2, q_3...\}$ *if* $Q_a = \{q_1, q_2\}$.

      ii. *repeat step* (i) *until all possible abstractions for* o_k *is done.*

   (b) *Union the class attribute sets formed in* (a) *to* L_{set}.

   (c) *Replace* O_{set} *with the class attribute sets formed in* (a).

Set of class label attributes sets (L_{set}) shown in Table 5.2 is enumerated according to Definition 10 from the data mining taxonomy given in Figure 5.1. Group of sets that is placed between a pair of horizontal lines in the table corresponds to the sets merged to L_{set} after each iteration of line (2) in Definition 10 and also constitutes the contents of O_{set} for the next iteration. Abstract data mining quality features that are replaced in the last iteration are shown in bold in the table.

Next, we augment E_I with abstract data mining quality attributes and subsequently with class labels which are the abstract data mining quality attributes aggregated according class label attributes sets in L_{set}.

Definition 11 *Definitions of abstract data mining quality relation schema (G), aggregation function for class labels (f_{AL_i}) and class labels relation schema (Q_{AL}) are in order:*

- *Abstract data mining quality*

  Let $G(g_1 : D_1, ..., g_n : D_{|Q_G|})$ be a relation schema defining abstract data mining quality such that $g_i \in Q_G$.

  G_I for a particular Q_{D_I} is a set of tuples with $|Q_G|$ fields such that f_{g_i} given in Definition 9 maps successors of g_i (Q_{g_i}) to g_i in Q_{D_I}.

- *Aggregation function*

  Let $L_{set} = \{l_1, l_2, ...\}$ be the set of class label attributes set enumerated from T (Definition 10).

  f_{AL_i} (for $i = 1, ..., |L_{set}|$) is an invertible function that aggregates tuples in Q_{D_I} and G_I based on class label attributes in l_i.

- *Class labels*

  Let $n = |L_{set}|$ and $Q_{AL}(q_{al_1} : D_{al_1}, q_{al_2} : D_{al_2}, ..., q_{al_n} : D_{al_n})$ define a relation schema for class labels and $dom_{al_i} = R_{f_{AL_i}}$ is the set of values associated with the domain named D_{al_i}.

  Q_{AL_I} for a particular Q_{D_I} and G_I is a set of tuples with n fields such that f_{AL_i} given above maps attributes in l_i to q_{al_i}.

In the extreme case, classifying the execution related data by using the class label formed by aggregation of attributes in Q_D results in a model with the most exact quality information but the predictive accuracy of the model is also important for generating adequate parameter setting recommendations. For this reason, we considered model's accuracy as well as the classification specificity that the class label attribute provides when choosing the most adequate class label attribute for the behavior model. Since the accuracy of the model can be assessed once it is built, we pre-screened the class label attributes by using a test in order to reduce the number of decision trees needed. One of the known reasons for the model with high error rates is to use a training set

with insufficient number of instances per class. SCRN (Algorithm 1) tests whether the number of instances per class for each class label attribute set in L_{set} is sufficiently large and eliminates the ones that contain high number of classes with small number of instances in E_I.

Algorithm 1 SCRN

Require: Class label attributes are enumerated
{Input is L_{set}, Q_{AL_I} }
{Output is S_{set}, Q_{AS_I} }
$S_{set} \leftarrow \{\}$ {Screened set of class label attributes sets}
$S_{attr} \leftarrow \{\}$ {Screened attributes from Q_{AL}}
$recs \leftarrow \mathtt{g}_{count}(Q_{AL_I})$ {Total number of instances}
$t\sharp \leftarrow threshold$ {Number of instances in a class}
$t\% \leftarrow threshold\_percent$ {Number of instances in a class as % of $recs$}
$n \leftarrow numberof\_classes\_below\_threshold$
$t \leftarrow smaller\_of(t\sharp, recs * t\%)$
for $k = 1$ **to** $|L_{set}|$ **do**
  $\downarrow thresh \leftarrow 0$ {Number of classes below threshold}
  for $i = 1$ **to** $|dom_{al_k}|$ **do**
    $al_{k_i} \leftarrow member(dom_{al_k}, i)$
    {Returns the i^{th} class in the domain}
    $c \leftarrow \mathtt{g}_{count}(\sigma_{q_{al_k}=al_{k_i}}(Q_{AL_I}))$
    if $c < t$ **then**
      $\downarrow thresh++$
    end if
  end for
  if $\downarrow thresh > n$ **then**
    $S_{set} \leftarrow S_{set} \cup member(L_{set}, k)$
    $S_{attr} \leftarrow S_{attr} \cup q_{al_k}$
  end if
end for
$Q_{AS_I} \leftarrow \Pi_{S_{attr}}(Q_{AL_I})$

SCRN returns S_{set} and Q_{AS_I} which are the pre-screened class label attributes sets and projection of pre-screened class labels on Q_{AL_I} respectively. Let relation schema of Q_{AS_I} be $Q_{AS}(q_{as_1} : D_{as_1}, q_{as_2} : D_{as_2}, ..., q_{as_n} : D_{as_n})$ where D_{as_i} is the name of the domain set of q_{as_i}.

As a result of classification of execution related data by decision tree using each q_{as_i} as the class label attribute, the number of models that are CONStructed is $|S_{set}|$:

$$M_i : C \times P \longrightarrow q_{as_i}$$

Each M_i is EVALuated separately by using accuracy as the performance metric. Let $S_{set} = \{s_1, s_2, ...\}$ be the screened set of class label attributes sets. M_i is the model

obtained by classifying on the class label attribute q_{as_i} which is the aggregation of attributes in the set s_i. The observed accuracy of M_i is represented by acc_i:

$$s_i \longrightarrow q_{as_i} \longrightarrow M_i \longrightarrow acc_i$$

Definition 12 *Accuracy of M_i (acc_i) is estimated by k-fold cross-validation method where training and testing are repeated for k times. Let $E_{I/k}$ be a partition of E_I which is divided into k equal sized mutually exclusive subsets and let $E_{I/rest}$ represent data in the rest of the partitions. $E_{I/rest}$ is used as the training set to build a model say m_{ij} at j^{th} iteration whereas $E_{I/k}$ is used for testing m_{ij}. At each iteration of j, $E_{I/k}$ is replaced by a partition from E_I which is not used as test set previously and $E_{I/rest}$ holds the remaining subsets other than $E_{I/k}$. Let acc_{ij} be the number of correct classifications from m_{ij} at j^{th} iteration, then the accuracy estimate of M_i is the proportion of overall accuracy estimations from the k-iterations to the number of instances in E_I:*

$$acc_i = \sum_{j=1}^{k} acc_{ij}/|E_I|$$

Specificity of classification by q_{as_i} which is the aggregation of quality attributes in s_i, is calculated by making use of s_i's every attribute's level in data mining quality taxonomy (T). MSEL (Algorithm 2) evaluates the model constructed for each s_i by estimating the model's accuracy as suggested in Definition 12, quantifies the specificity that s_i provides and computes a score for s_i. A coefficient is added in the formula that computes the score so that the weights of the two factors contributing to the score can be adjusted. Behavior model is built using the class label which is scored highest in terms of accuracy and specificity of classification.

Algorithm 2 MSEL

Require: Enumerated class labels are screened and data transformations on E_I is done such that Q_{AS_I} is produced.
{Input is A_I, S_{set} , E_I, $Q_{AS_I}, coefficient$}
{Output is M }
1: $max\_score \leftarrow 0$
2: $top\_s \leftarrow return\_finestspecificitydegree(A_I)$
3: $choose \leftarrow 0$
4: **for** $i = 1$ **to** $|S_{set}|$ **do**
5: $s_i \leftarrow member(S_{set}, i)$
{Estimate accuracy when s_i is the class label}
6: $accuracy \leftarrow EVAL(E_I, Q_{AS_I}, s_i)$
7: $specificity \leftarrow 0$
8: **for** $j = 1$ **to** $|s_i|$ **do**
9: $qual\_attr \leftarrow member(s_i, j)$
{Returns the j^{th} attribute in class label set}
10: $l \leftarrow taxonomy\_level(A_I, qual\_attr)$
{Returns the attribute's level in the taxonomy}
11: $specificity \leftarrow specificity + l$
12: **end for**
{Normalize specificity degree }
13: $specificity \leftarrow specificity/top\_s * 100$
{Calculate score of classification by s_i }
14: $score \leftarrow accuracy + specificity * coefficient$
15: **if** $score > max\_score$ **then**
16: $max\_score \leftarrow score$
17: $choose \leftarrow i$
18: **end if**
19: **end for**
{Build a decision tree with highest scored class label}
20: $M \leftarrow BUILD(E_I, s_{choose}, Q_{AS_I})$

Chapter 6

INSTANTIATION OF THE APPROACH

In this chapter, we illustrate our approach through two ubiquitous computing applications in which data mining is used.

6.1 A Museum Equipped with Ambient Intelligence

In the given example, ubiquitous devices are used by many people performing the same activity and somehow sharing information about their activity among themselves. In this sense, they form a social network in the ambient intelligence environment that we delineate. We explain how our approach can be employed in this application and finally instantiate our approach with data specific to this example application.

The museum depicted in this example is huge so that it takes a lot of time of the visitors to see all the exhibitions in it and it may even be impossible within the visit time. Museum is conceived as an ambient intelligence environment where visitors are fed information from the environment. The aim is to guide the visitors so that they can visit the pieces (art work) that they would like to see within the available time rather than trying to see all of the exhibitions. For this purpose, an application that we call *museum guide* is loaded to the smartphones of the visitors upon request. *museum guide* directs a visitor to the pieces that he would like in the museum.

As in a common web based book or movie recommender system, the objective of this system is to discover common likes of users. In order to do that, information indicating whether the visitor liked or not liked the last artwork he looked at, is needed. For this purpose, the amount of time each visitor spends in front of a artwork is sensed and collected and is taken as the indication of the visitor's liking or not liking of that piece. If an indoor positioning system is not present, visitor may also submit his feedback by making use of the *museum guide* application. Sensed data or feedback of each visitor is transformed to a record that we call *visitor record*. *visitor record* contains the pieces liked by that visitor.

museum guide which runs on the smartphone of a specific visitor while recording the pieces he liked to his smartphone, also downloads other *visitors' records*. Common likes are discovered by association rule mining of all *visitors' records* on the smartphones of the visitors. The purpose is to find the associations such as "visitors who liked Picasso's Three Musicians also liked Matisse's Dance". This museum has special offers for different types of visitors such as tourists, students and elderly. For this reason, visitor profile in one season, month or week of day may be quite different than the other. Since more associations can be found when similar people's likings are mined and the museum has dynamic visitor profile by day, rather than discovering the associations only once from data of one set of visitors, it is preferred to extract the association rules dynamically on visitors' smartphones. Moreover, continuous rotation of artwork also necessitates mining to be performed dynamically. Since association rule mining is not done by a central computer thus *visitors' records* are not stored centrally, the exchange of *visitors' records* among the visitors is achieved by making use of the internet and social media. After being updated each time, *museum guide tweets* the *visitor record* by tagging it with a pre-determined *hashtag*. *Tweets* with the pre-determined *hashtag* are downloaded by the *museum guide* to generate the records for association rule mining.

We focus on determining the configuration of the association rule mining algorithm which must run autonomously since in a ubiquitous computing environment it is assumed that the user can not provide this information. We exploit context and consider

the availability of the resources of the visitor's smartphone as the determining factor of the association rule mining parameter settings. The data mining process is referred as *discover artwork associations*. Next, we describe a number of principles of *museum guide* and how it interacts with *discover artwork associations*:

- *museum guide* calls *discover artwork associations* to discover frequent itemsets from the set of *visitor records*. The attribute of each *visitor record* is the list of pieces liked by that visitor.

- *museum guide* downloads fresh data and calls *discover artwork associations* several times during his visit for a visitor in order to incorporate data of newcomers and to try different parameters for better recommendations.

- Association rule mining algorithm Apriori [2] which accepts two parameters: *minimum support* and *minimum confidence* is used to *discover artwork associations*.

- The resulting data mining model generated by Apriori are the association rules demonstrating which pieces that are exhibited in the museum are liked by the same persons. It is out of the scope of this work to speculate on how *museum guide* uses the model to recommend artwork to the visitor or whether the recommendations are ordered by physical location or some other criteria as well as when a new model is needed. On the other hand, we are concerned on the automatic configuration of *discover artwork associations*.

- Past executions of *discover artwork associations* are mined to discover the appropriate configuration that fulfils the required quality under a given circumstance. Initial past execution data is downloaded to the smartphone together with the application and is augmented by data collected during executions of Apriori locally on the visitor's smartphone.

6.1.1 Circumstantial Factors Effecting Parameter Setting

We argue that in a ubiquitous computing environment, in order to find appropriate settings of the data mining algorithm parameters, context from the environment as well as the conditions of the device's resources need to be utilized. Next, we define the relevant circumstantial factors for determining configuration of *discover artwork associations*.

Context: Context, that we assume have an effect on the required quality of the final model are:

- time left to the museum's closing (*remaining time to close*)
- time left to average visit time since the start of visit time (*remaining time to leave*)
- visitor's past attitude against the recommendations (*feedback*)
- number of visitors in the gallery entered (*no of visitors*)

Resources: Since *discover artwork associations* runs on the smartphones that are restricted resource devices, the resource usage of the data mining process need to be considered when setting the parameters of the data mining process. We assume memory and processor are the resources whose availability are critical for *discover artwork associations*:

- amount of memory available (*memory available*)
- processor idle percentage (*processor idle percent*)

6.1.2 Heuristics for Parameter Setting

In this subsection, we give example heuristic configuration decisions for *discover artwork associations*. We assume there exist default settings for each *discover artwork*

associations parameter: *minimum support* and *minimum confidence* and in the configuration decision whether to increase or decrease the defaults of the relevant parameters depending on the circumstance is indicated. The following are some heuristics which may be used when configuring *discover artwork associations* autonomously where the reasoning explaining the heuristic follows it.

1. if *memory available* is low then increase the *minimum support* (with a higher *minimum support* value it is expected to decrease the size of the frequent itemsets and to optimize memory usage consequently),

2. if *remaining time to close* is small then increase both the *minimum confidence* and *minimum support* (since limited time is left, provide less rules with higher confidence so that the visitor will not miss the pieces that he would like most),

3. if *remaining time to leave* is small given that *memory available* is not low, decrease the *minimum support* (the objective is to make the average visit time longer by providing more pieces to the visitor that he would regret if he would leave without seeing them)

4. if *no of visitors* is high then decrease the *minimum support* but increase the *minimum confidence* (as the visitor may prefer to skip the pieces with a crowded audience in front of them, produce a list of high confidence containing sufficient number of pieces to bypass some of them)

5. if *feedback* is negative then decrease the *minimum support* (if the visitor is not satisfied with the previous recommendations then provide more accuracy)

Circumstance is the motive of each parameter setting but there is also an objective of each recommended setting which is given in parentheses after each heuristic. Objectives can be quantified by making use of the measurements obtained from the operating system of the device as well as the data mining model quality indicators.

Quality Measures: Quality measurements are the means to control whether the heuristic for a setting achieves the objective. The suggested quality measurements are

as follows:

- maximum amount of memory used by *discover artwork associations* (*max memory usage*)
- number of association rules in the model (*no of rules discovered*)
- minimum confidence that an association rule in the model may have (*model min conf*)
- minimum support that association rules of the model should have (*model min support*)

Whether the objective of assigning certain value(s) to *discover artwork associations*'s parameter(s) is attained at a specific run of *discover artwork associations* can be assessed by checking the related quality measure(s) after the *discover artwork associations* runs with those settings. Similarly, the objective given is the required quality for a given circumstance.

6.1.3 Instantiation for Apriori

In this subsection, automatic configuration setting is instantiated for the well known association rule mining algorithm Apriori. Instantiation is based on the intelligent museum example and consists of appropriate *discover artwork associations* configurations for the heuristic parameter setting decisions given in subsection 6.1.2 as well as the possible circumstances and quality determined in subsection 6.1.1 for the intelligent museum example.

Circumstance. It is assumed that context and resource availability values are discretized such that *'low'*, *'high'* and *'moderate'* categories are used for *memory available*, *'few'* and and *'many'* for *no of visitors*, *'not much'* and *'plenty'* are used for *remaining time to close* and *remaining time to leave* and finally, *'positive'* and *'negative'* are for *feedback*. Some possible instantiations of circumstances using discretized values are as

follows:

C_1= {(memory available, 'low')}

C_2= {(remaining time to close, 'not much')}

C_3= {(memory available, 'high'),(remaining time to leave, 'not much')}

C_4= {(no of visitors, 'many')}

C_5= {(feedback, 'negative')}

Quality. Similarly, it is also assumed that quality measurement values are discretized such that *'low'*, *'high'* and *'moderate'* categories are used for *max memory usage*, *'few'* and and *'many'* for *no of rules discovered*, *'low'* and *'high'* are used for both *model min conf* and *model min support*. Some possible instantiations of circumstances using discretized values are as follows:

Q_1 ={(max memory usage, 'low')}

Q_2 ={(model min conf, 'high'), (model min support, 'high')}

Q_3 ={(no of rules discovered, 'many')}

Q_4 ={(model min conf, 'high'), (no of rules discovered, 'few')}

Q_5 ={(model min support, 'low')}

Algorithm Configuration. Assuming that the default values of *minimum support* and *minimum confidence* are 0.75 and 0.9 respectively, in each of the parameter setting below either one of them or both are altered to meet the parameter setting decisions of the example heuristics.

P_1 = {(minimum support, 0.9), (minimum confidence, 0.9)}

P_2 = {(minimum support, 0.85), (minimum confidence, 0.98)}

P_3 = {(minimum support, 0.75), (minimum confidence, 0.9)}

P_4 = {(minimum support, 0.5), (minimum confidence, 0.98)}

P_5 = {(minimum support, 0.70), (minimum confidence, 0.9)}

Configuration Decisions. Instantiation of configuration decisions are based on the instantiations of circumstance, quality criteria and algorithm configuration such that

for each circumstance C_i given above, Q_i is the corresponding quality aimed for C_i and P_i is the appropriate configuration setting given C_i and Q_i. The following pseudo code generalizes the instantiation of configuration decisions for the heuristics of subsection 6.1.2:

```
forall i < 6
   if C_i is sensed/gauged and
      Q_i is the corresponding required quality
      then P_i is an appropriate configuration.
```

6.2 FESTweets, Movie Recommendations for a Film Festival

FESTweets is an application that is designed to help film festival audience on choosing the films. Film festival organization presents films from all around the world such that a vast number of films that are not shown at the local cinemas during the cinema season are screened within a limited amount of days during the festival.

Since most of the films in the festival program are screened only a few times and the screening times of the films do overlap, it is impossible for a person to watch all the films in the festival program. Hence, film festival audience who plans to follow the festival, must be selective. On the other hand, it's not easy, at least time-consuming to analyze reviews and to acquire detailed information about each and every film appearing in the festival program. Consequently, recommendations on which films to watch is very likely to be welcomed by the film festival audience.

How festival audience obtains the film recommendations is an important issue. Most of the people living in a metropolis use mobile phones categorized as smartphone and usually prefer to connect to the internet by their mobiles. Thus, it is important that film festival audience can get the recommendations to their smartphones and also able

to get a new set of recommendations everywhere that can be connected to the internet.

What type of recommendation system is appropriate for generating film recommendations is another issue to be answered. There are two types of recommender systems: collaborative and content-based. In contrast to content based recommender systems that make use of the information and characteristics derived from the several attributes of the items, collaborative recommender systems rely on users' preferences instead of the content for predicting the common tastes. It is unfeasible to collect content information for the festival films that will be aired due to their large number and the limited time between the announcement of the festival program and the ticket release date. Moreover, it had been observed at previous years that majority of the film viewers bought tickets to several films rather than a single film indicating that suitable data for collaborative filtering can be collected this year too.

Yet another important issue is how to collect the information needed for film recommendations. There is no doubt that a substantial number of people in the world like to share their interests, photos, opinions and even daily activities with their friends and also with the related community through social media/network. Especially, widespread usage of social network/media sites, Facebook and Twitter resulted in almost every internet user to sign up the mentioned sites. Social media which is the most appropriate means for the people to collaborate must be the channel to share the film preferences.

In the design of FESTweets, we considered the aforementioned requirements. Collaborative filtering is preferred to content-based filtering when designing FESTweets so that films are recommended based on the collaboration of film audience who shows their interests in the film festival by buying tickets. The recommendations are based on the extracted "people who buy ticket to film x also buy ticket to film y" associations. We have chosen Twitter([43]) as the media where festival audience can share their preferences. On the other hand, associations among film preferences are extracted and presented on the smartphones.

As of today, Twitter which lets the subscribed users to share content in the form of

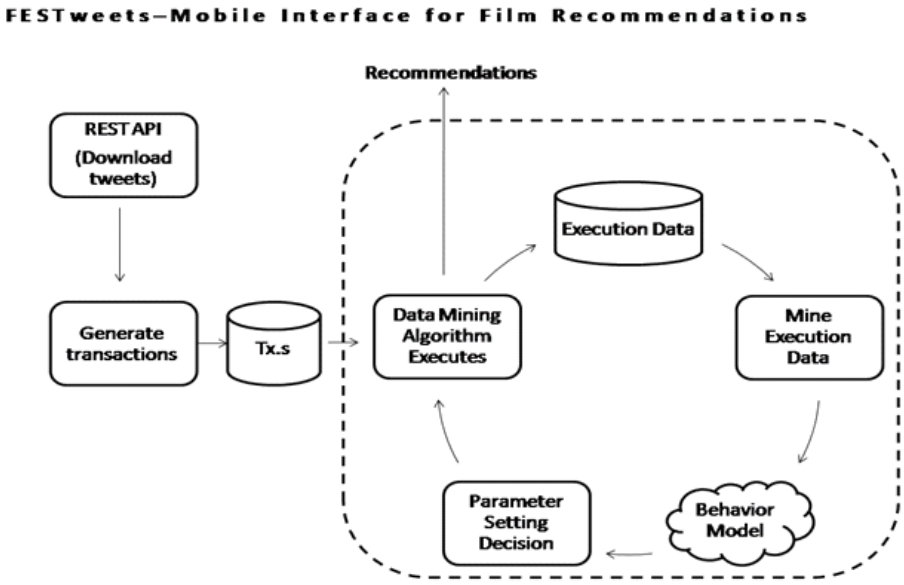
short messages (tweets) not exceeding 140 characters, is the most popular microblogging site without dispute. Twitter is not merely a media to form a social network among individuals but it is also used by the organizations as a means to disseminate news and publish information for several purposes such as marketing and public relations. Furthermore, contents gathered by Twitter are made available so that third parties offer a wide variety of applications that exploit Twitter data. For the mentioned reasons, we have chosen Twitter as the data source of our data mining application that mines data ubiquitously on a smartphone.

In short, FESTweets acquires film preferences of the festival audience who bought tickets, discovers the associations between their choices and generates film recommendations to festival audience who plans to buy tickets based on the associations discovered. An overall view of the application is given in Figure 6.1(a) and 6.1(b). FESTweets has two independent parts. A social media interface allows the festival audience to enter their film preference lists. Mobile interface generates film recommendations by mining downloaded film preferences on the smartphone of the user.

As seen in Figure 6.1, all festival film audience who wants to participate FESTweets, must use a Twitter account. Festival audience tweets their film preferences by hashtagging them using a predetermined hashtag (such as $\sharp festweet$). In this way, tweets containing $\sharp festweet$ are accumulated. Users who want to receive recommendations on films download the $\sharp festweet$ hashtagged tweets by a mobile application through Twitter API's. We provide a brief information on Twitter in Appendix B.

FESTweets will be used by the festival audience at any time between the festival ticket release date and the end of the festival. During this period, as more and more film viewers "tweet" their film choices, film recommendations that are generated may increase in quantity and also may change. For this reason, mobile interface of FESTweets should be repeatedly run in order to trigger the download of newly added "tweet"s of film audience and mine the transactions obtained from "tweet"s. It is reasonable to mine this ever increasing data with different configurations depending on the situation. In Section 8.5, we discuss in detail the requirements of the data mining

(a) Social media interface for data gathering



(b) Mobile interface for recommendations

Figure 6.1: Overall view of FESTweet

model and probable processing constraints that would affect the data mining model under different circumstances.

Chapter 7

EXPERIMENTAL EVALUATION

This section explains the experiments that we have performed in order to show the applicability of the proposed approaches for obtaining a behavior model that can be used for recommending data mining configuration. After introducing the experiment software that we used, in the next two subsequent sections empirical evaluations of the approaches are given.

7.1 Experiment Software

We have developed a software that we call *execution data generator* to generate experiment data. Execution data generator (EDG) collects execution related data (E) for the experiment by running the data mining algorithm with various configurations under various circumstances created by EDG.

7.1.1 Execution Data Generator Architecture

The main task of EDG is to run a data mining algorithm and to collect relevant data from each execution of the algorithm. EDG also creates the planned bottlenecks on the device's resources before running the algorithm.

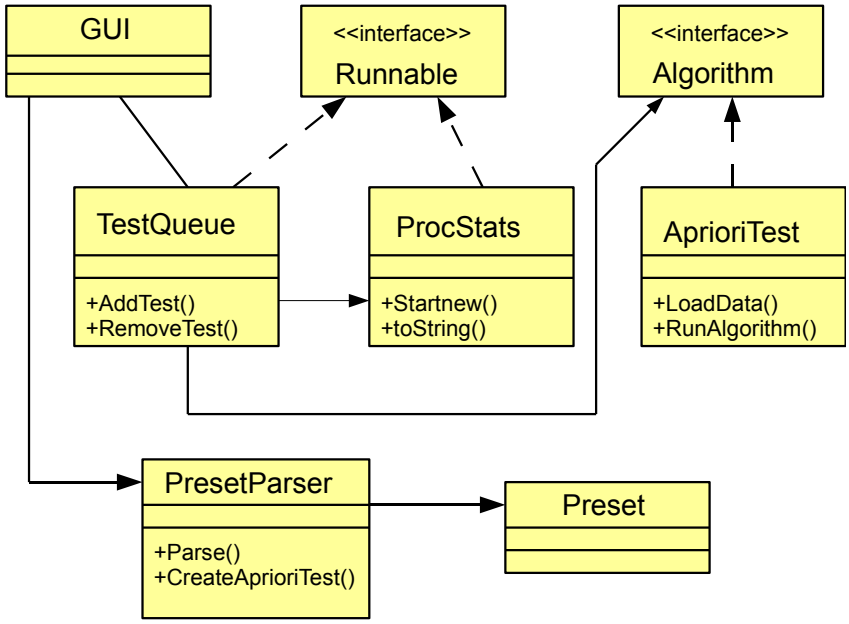


Figure 7.1: Class descriptions of EDG

We have chosen well known association rule mining algorithm, Apriori ([2]) as the sample algorithm that is run by EDG for the experiment. Data generator software consists of JAVA programs except the bottleneck creator modules which are C++ programs. Apriori is run by calling Weka ([35]) API's within EDG.

EDG input (*preset* file). Each record in *preset* file defines a particular execution of Apriori and contains associated context data for this execution, resource bottleneck requests, data set to be mined and configuration of Apriori. Resource bottleneck requests state the amount of memory and/or processor consumption in the device by the workload other than Apriori during execution.

EDG output (*execution* file). A record which consists of circumstance (C), parameter (P) and quality (Q) attributes is written for each execution of Apriori. EDG output is real data collected before, during and after Apriori execution such that the gauges showing resources' availability when Apriori was run, actual resource usages by Apriori, quality indicators from the data model generated and Apriori configuration are stored in C, Q and P attributes respectively.

Briefly, EDG reads a record from the *preset* file, generates the resource scarcity conditions if the given circumstance requires and runs Apriori with the given parameters. For example, if the stated resource state is the scarcity of memory, EDG starts dummy processes to use up the memory in order to run Apriori in memory constrained situation. Upon completion, an execution record which is populated by real statistics collected during the execution of Apriori, is created.

Class descriptions of EDG are shown in Figure 7.1. There is a graphical interface (*GUI*) to set the name of the *preset* file and the execution file as well as to start the data generation. *PresetParser* is used to parse the contents of *preset* file and responsible for invoking bottleneck creators to call some "dummy programs" that will consume the requested amount of related resource. *TestQueue* is typically a queue that contains *Algorithm* instances. *AprioriTest* represents tests of the Apriori algorithm and implements the interface *Algorithm*, thus its instances can be added to *TestQueue*. *ProcStats* performs the gathering of performance statistics before, after and during the execution of the algorithm tests. Specific system metrics related to memory or processor usage are gathered using specific methods. This class is designed as an independent cohesive unit to measure performance metrics, gather system information and statistics.

7.2 Evaluation of Self-Configuration by Bayesian Network

We conducted an empirical study to demonstrate that parameter setting decisions by the proposed mechanism are appropriate in the sense that they are good at delivering the quality requested for the circumstances. To validate the proposed mechanism, we selected Apriori as the data mining algorithm to be configured and created its behavior model in Bayesian network representation to derive configuration decisions. Besides, we employed another approach for parameter setting, full factorial experiment design and compared the inferences made from the Bayesian network against the results of the

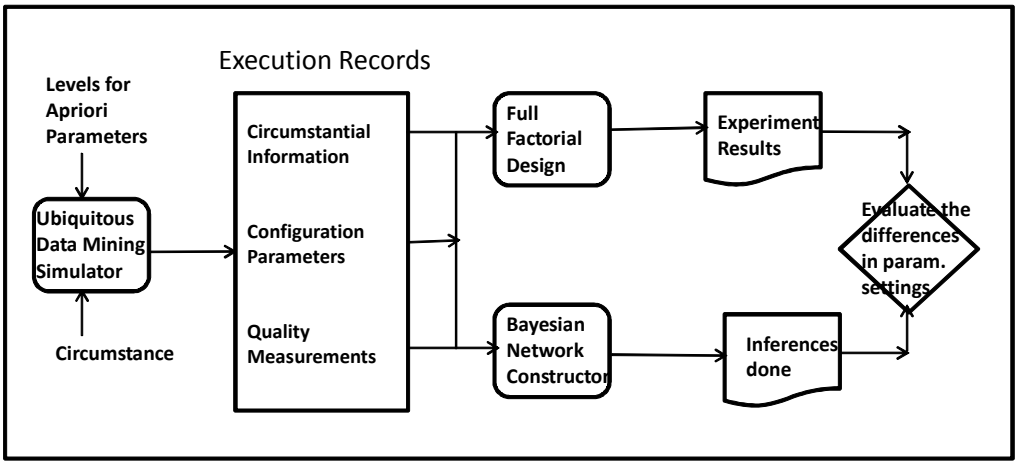


Figure 7.2: Experiment phases

full factorial experiment design. The experiment has the following steps:

1. Ubiquitous data mining simulator is developed to generate experiment data.
2. Bayesian network is constructed in order to discover the probabilistic relationships among parameters, circumstances and quality.
3. Multi-level full factorial design is used to find out the parameters that are effective on the quality under a given circumstance.
4. The results of the two methods are compared to assess the proposed mechanism.

Fig. 7.2 shows the interaction of the experiment steps and the ubiquitous data mining simulator.

7.2.1 Experiment Dataset

Execution data for the experiment was generated using the ubiquitous data mining simulator. The states of the context and the type of resource constraints that were

Table 7.1: Levels used for parameters

Context state	Mnemonic	Parameter	Settings
Home	U	upper bound minimum support	0.7, 0.8, 0.9
	M	lower bound minimum support	0.1, 0.2, 0.3, 0.4, 0.5, 0.6
	D	delta	0.01, 0.05, 0.1, 0.15, 0.2
	N	number of association rules	1, 5, 10, 15, 20
	C	minimum confidence	0.5, 0.6, 0.7, 0.8, 0.9
Office	U	upper bound minimum support	0.7, 0.8, 0.9
	M	lower bound minimum support	0.4, 0.5, 0.6
	D	delta	0.01, 0.05, 0.1, 0.15, 0.2
	N	number of association rules	15, 20
	C	minimum confidence	0.8, 0.9

used in forming the circumstances of the test cases are {*home*, *office*} and {*short on memory*, *cpu bottleneck*, *none*} respectively. All the types of resource constraints were simulated for each context state, resulting in six different circumstances. Settings used for each Apriori parameters are given in Table 7.1. There are different sets of settings for *home* and *office*. In the experiment Apriori was run for all combinations of the determined settings for each of the six circumstance. Therefore, the number of test cases for each circumstance having *home* as context state are 2250(3x6x5x5x5) and *office* as context state is 180(3x3x5x2x2).

7.2.2 Parameter Setting by Bayesian Network Inferences

In this step, we applied our mechanism to predict Apriori configurations from the Bayesian network. Bayesian network construction and inferencing from the network are two main tasks of this step.

Execution data generated by ubiquitous data mining simulator were first discretized before constructing the Bayesian network given in Fig. 7.3. While discretizing, we used equal frequency bins and chose the number of bins that produced the highest number of relationships the among nodes. While constructing the network we made use of the K2 algorithm ([21]) by modifying it to group the nodes and searched causal relationship among these groups of nodes. The nodes in the upper level of the network in Fig. 7.3 represent the circumstance, middle level nodes represent Apriori parameters, and finally the lowest level nodes are quality measures. The cause and effect relationships between

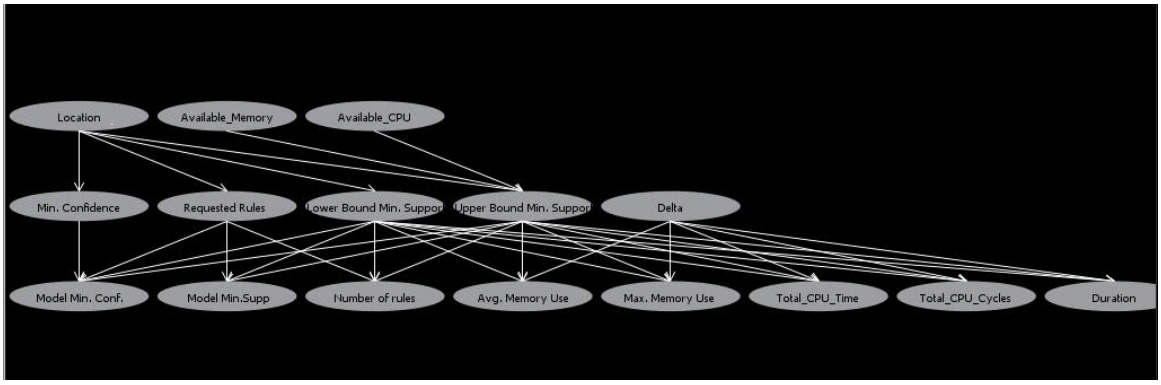


Figure 7.3: Bayesian network of Apriori runs

circumstances and parameters present which parameter settings are appropriate under which circumstances, whereas the cause and effect relationships between parameters and quality measures show which parameters are effective on which quality measures.

While producing the experiment data for this Bayesian network, we did not determine appropriate parameter settings for circumstances but we ran Apriori for every combination of parameters in each circumstance because our purpose is to find the effect of parameters to quality measurements in the first place. Therefore, at this stage the relationships between circumstances and parameters are not meaningful. We assumed each circumstance node relates to each parameter node in order to include circumstances in the inference mechanism. The relationships between the parameter nodes and quality measure nodes represent the effectiveness of parameters against quality measurements. The Bayesian network in Fig. 7.3 shows that *minimum confidence* and *requested rules* are related only to efficacy; *delta* to all efficiency measurements as well as *lower and upper bound minimum support*, are related to all.

We determined parameter settings decisions by inferencing from the Bayesian network given in Fig. 7.3. The pseudocode of the estimation is given in subsection 4.2.

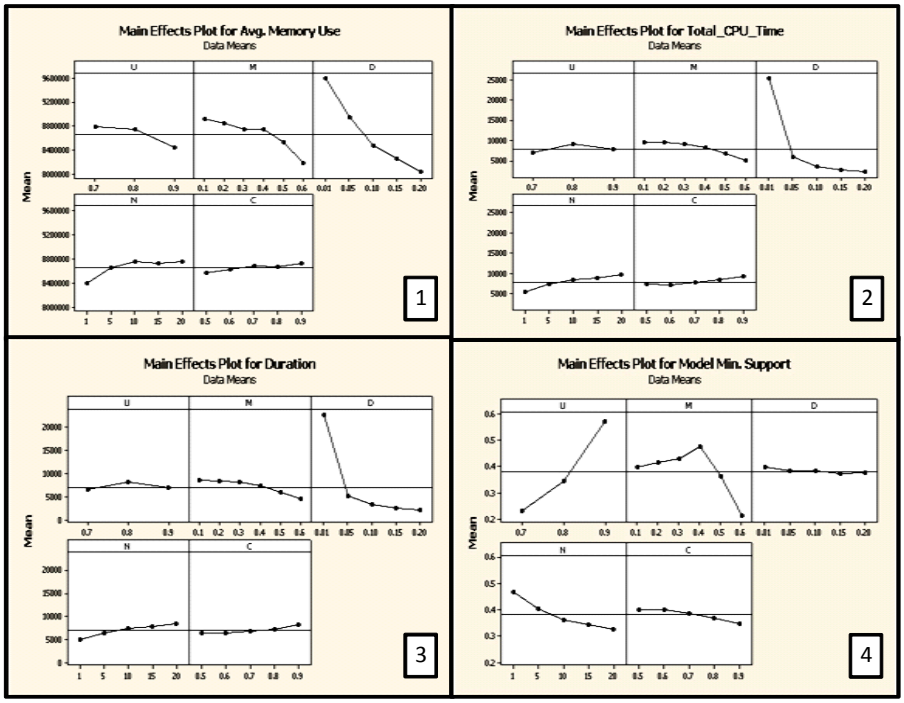


Figure 7.4: Main effects plot of 4 quality measurements for *home-short on memory*

7.2.3 Multi-level Full-Factorial Experiment Design

In this step, we applied multi-level full factorial experiment design which is one of the Design of Experiment (DoE) methods [48]. Full factorial experiment design is statistically determining the effects of the factors of a process to its response by systematically varying the levels of the factors during testing of the process. In DoE terminology, *response* is the output variable of the process, *factors* are its input variables and *level* is a possible setting for a factor. The process that we want to analyze is the behavior of Apriori, more specifically, to find out which Apriori parameters affect which quality measurements. Therefore, Apriori parameters are the *factors*, their possible settings are the *levels* and resulting quality is the *response*. In full factorial experiment design, data is collected by running the process with all combinations of determined levels of its factors. Hence, we generated execution data similarly by running Apriori for all combinations of settings as explained in subsection 7.2.1. Moreover, since we ran Apriori by simulating six specific circumstances, we are able to analyze the effects for each circumstance.

We used experiment software Minitab([42]) to estimate the effects and to plot the analysis results. Fig. 7.4 illustrates the full factorial experiment design results obtained for *home-memory low*. We analyze the results for this circumstance in detail in order to explain the method. In the figure, the means of quality measurements for the utilized levels of parameters are plotted. In quadrants of Fig. 7.4, plots for *average memory use*, *total CPU time*, *duration* and *minimum support of the model* are given respectively. Each plot (U, M, D, N, C) within a quadrant is for a parameter. The mean of the measured value is plotted for every level we tested for that parameter in the experiment. If the plot is not flat which indicates the means of measured values vary with different value assignments of this parameter, then this parameter is effective on the measured value. We considered the F test values to determine the significance of the effect. While determining the appropriate value of the parameter which is designated as effective on the measured criteria, we have chosen the value that has the smallest mean of response for its factor level combinations. We compare the results of full factorial experiment design against the results of the Bayesian network in the next subsection.

7.2.4 Comparison of Results

In order to determine the parameter settings of an algorithm, we explained two different approaches, Bayesian networks and full factorial experiment design where the former is a probabilistic approach and the latter a statistical approach. The outcomes of the approaches are summarized as follows:

- Full factorial design provides
  - The list of parameters which are not effective on a quality measure
  - The parameter setting which has the highest/lowest least square mean for a quality measure
- Inference from Bayesian network provides
  - The list of parameters which are not related to a quality measure

Table 7.2: Comparison of results

Circumstance	Efficiency		Efficacy	
	(i)	(ii)	(i)	(ii)
home-short on memory	73	77	90	100
home-CPU bottleneck	80	89	100	100
home-no constraints	73	77	80	80
office-short on memory	100	67	100	75
office-CPU bottleneck	100	89	90	75
office-no constraints	100	78	90	75

– Most likely parameter setting given the circumstance(s) and the quality measure(s) as evidence

To compare the results, we used two criteria: i) the percentage of alike parameter/quality measure relationships and, ii) the percentage of identical parameter settings, obtained by the two approaches. In Table 7.2, for each circumstance, we present, (i) and (ii) by grouping quality measures as efficiency related and efficacy related.

It is possible to say based on the results (Table 7.2) that in majority of the cases, parameters that are found to be effective on a quality measure under a circumstance in full factorial design, are represented as related to that quality measure under the same circumstance in the Bayesian network. The appropriate parameter settings decided in order to optimize a quality measure in full factorial design is identical in most of the cases to the parameter settings inferred from the Bayesian network given the same quality measure.

7.2.5 Effects of Mining Data Set Feature Variations on the Behavior Model

In the simulation phase of the experiment (subsection 7.1), we mined always the same data set with Apriori and in this way we eliminated the effects of the data set feature changes on the behavior model constructed. On the other hand, in a real life situation data set to be mined may grow or shrink either by addition or deletion of instances into

the data set or by attribute set changes. Variations on the mining data set features may necessitate refreshing the behavior model that is used to recommend algorithm configurations for mining this data set. Thus, we performed a series of experiments to affirm that mining data set size may have an effect on the parameter setting recommendations and also to speculate on how to detect that the behavior model is decayed. In the experiments, we made use of quality measurement figures collected during the execution of Apriori to assess whether the recommended parameter settings provide the requested quality. Experiments rely on the behavior model that we call *basis behavior model* generated in the same way explained in subsection 7.2.2 from the execution records collected by running Apriori with input data set (*DSx1*) in a simulated ubiquitous computing environment similar to the one explained in subsection 7.1. Brief explanation of data set size variations effect evaluation experiments are as follows:

- **Verify the recommendations.** In this experiment, Apriori was configured by the recommended settings acquired from the *basis behavior model* and ran with input *DSx1* in the simulated ubiquitous computing environment for every possible recommendation. Afterwards, we determined the appropriateness of each recommendation by comparing the relevant quality measurement value collected during Apriori's execution against the requested quality used when deriving the recommendation from Bayesian network. For example, if an Apriori configuration is recommended to minimize the memory usage of Apriori, we assess the parameter setting objective by comparing the memory usage figures of Apriori's execution with this configuration against the lowest memory usage figures in the behavior model. The percentage of the Apriori executions which achieve the objective of the parameter settings grouped by relevant quality measurement are given in Figure 7.5. Percentage of deviation from the requested quality is also analyzed for each quality measurement group. The maximum amount of deviation is ten percent of the requested quality whereas the average amount of deviation does not exceed five percent of the requested quality for any of the groups (Figure 7.5). The results obtained are satisfactory to verify the appropriateness of the recommendations.

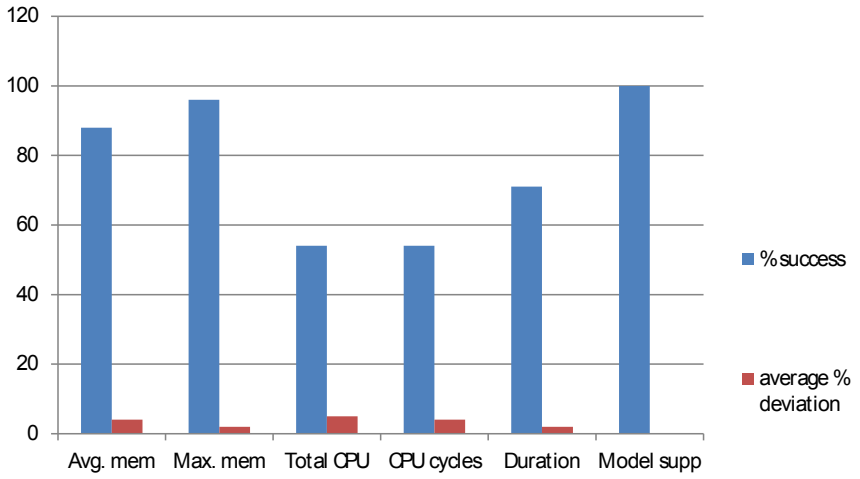


Figure 7.5: Assessment of recommendations derived from Bayesian network

- **Demonstrate the data set size effect.** We try to find out in this experiment whether the behavior models extracted from the executions of the same data mining algorithm with same configuration settings but with different data mining data set sizes, are different. For this purpose, we generated another behavior model, *behavior model 10* in a similar way that we generated *basis behavior model* but the size of the data set (*DSx10*) used as input to Apriori in this experiment is ten fold bigger than *DSx1*. After generating the behavior model (*behavior model 10*) for Apriori mining *DSx10*, we compared *behavior model 10* against *basis behavior model* and detected that half of the recommendation decisions are changed. By this way, we have shown that input data set's size of a data mining algorithm may have an impact on certain parameter settings decisions given in order to achieve certain quality objectives.

- **Estimate behavior model decay.** In the final experiment, we gradually increased the size of the mining data set mimicking a possible real life situation in which a data set grows in time. Our purpose is to analyze the deterioration of the recommendations in terms of achieving the quality requested as the data set

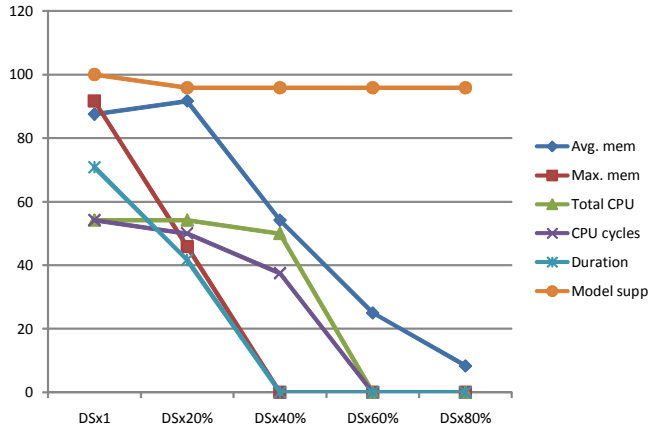


Figure 7.6: Behavior model decay

grows. We iteratively increased the size of the mining data set by twenty percent, run Apriori with all the possible recommended parameter settings extracted from the *basis behavior model* while simulating the relevant circumstance, recorded the requested quality for the recommendation and compare it against the achieved quality. During this process we used the same behavior model (*basis behavior model*) without populating new execution data or refreshing it completely. Figure 7.6 shows for different mining data set sizes the percentage of Apriori executions where the objective of the parameter setting is achieved in terms of the quality obtained. Experiment results show that the correctness of the configuration decisions derived from the behavior model in order to obtain the requested qualities of all types except the model minimum support are affected by the data set size change. Furthermore, *basis behavior model* decays needing a refresh before *DSx1* grows by forty percent. This experiment revealed that mining data set size change do not effect every parameter setting decision but if a parameter setting decision do not provide the requested quality, it is possible to detect.

7.3 Evaluation of Self-Configuration by Decision Trees

This section explains the experiments that we have performed in order to show the applicability of the approach by decision tree for recommending data mining configuration.

Table 7.3: Experiment fact table

1	Data Mining Algorithm					Apriori
2	Number of configurable parameters					5
	Mining data set	Size (in bytes)	Number of attributes			Number of instances
3		4,955,737	11			325,610
	Circumstantial Settings					
4	Number of context features					2 (c_1,c_2)
5	Number of resource features					2 (c_3,c_4)
			c_1	c_2	c_3	c_4
6	Number of states		6	5	3	3
7	Number of situations					150
8	Number of repetitions of a situation					10
9	Number of configuration templates					30
10	Number of configurations generated					1500
	Data Mining Quality Results					
			Resource usage		Data mining model	
11	Number of attributes		5		3	

The objectives of the experimental evaluation are: i) compare in terms of accuracy and specificity, the behavior models that classify execution data by different data mining quality abstractions extracted from a taxonomy, ii) assess the appropriateness of the heuristic used for pre-screening by calculating the accuracy of the models that would be eliminated due to pre-screening, iii) assess the configuration decisions derived from the behavior model.

Experiment Dataset

Experiment data was generated using the execution data generator that we have designed and implemented. We have collected 1500 execution records of Apriori by running the algorithm through EDG. Figures related to experiment setup are shown in Table 7.3. We chose five of the parameters Weka receives for Apriori API's as configurable parameters (line 2 in Table 7.3) and eliminated the parameters that are not subject to tuning. Throughout the experiments, we have used the same mining data set whose properties are given in line 3 in Table 7.3.

We incorporated circumstantial factors into the experiment as we were generating data for a ubiquitous computing environment. Two context features (c_1 and c_2) with

six and five states respectively as well as two resources features (c_3 and c_4) each having three states, were used in the experiments (line 4 thru 6 in Table 7.3). We selected arbitrary names for the features aiming a neater presentation. On the other hand, it is possible to associate them to any ubiquitous computing application domain. For example, the following context features and state sets may be used: *location* $\{indoor - confinedspace, indoor - highroof, outdoor - urban, outdoor - landscape, outdoor - forest, outdoor - coast\}$ and *time* $\{sunset, midday, night, sunrise, other\}$ instead of c_1 and c_2. Likewise, resource features can be associated to *available memory* and *processor idle percentage* with a state set such as $\{plenty, sufficient, scarce\}$.

During the experiments, we formed one hundred and fifty different circumstances by combining different context and resource states and we setup EDG to execute Apriori ten times for each circumstance (line 7 and 8 in Table 7.3).

We associated to every possible c_1 and c_2 state combination a configuration template which was used for setting the parameters of Apriori that would run in the associated context states. In a configuration template, either an interval of values or an exact value is used as a setting of a parameter . When an interval of values is used as a parameter setting, a random number within the given interval was generated by the *PresetParser* to be used as the setting of the associated parameter. Consequently, we coded thirty different configuration templates containing intervals in the *preset* file but the number of different configurations that EDG generated and used while running Apriori was a lot more since EDG generated the settings randomly within the given interval (line 9 and 10 in Table 7.3).

Generally, in order to determine how to set the parameters of an algorithm, we need to know the objectives of running the algorithm. In our case, we need to know the requirements of the context so that we can determine the parameter settings in its configuration template. For this reason, we associated context states with data mining model and processing requirements. Figure 7.7 shows the data mining model and processing requirement assumptions that we made on c_1 (c_1-coordinate of the cube) and c_2 (c_2-coordinate of the cube). For example, first state of c_1 implies to generate

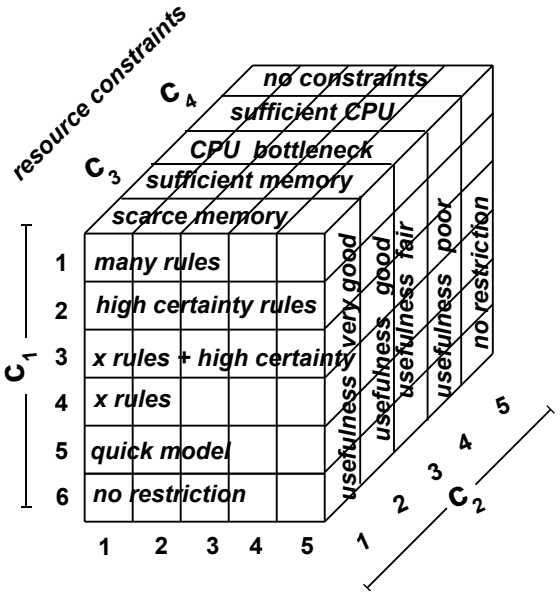


Figure 7.7: Cube of circumstances

a data mining model with many association rules, second state of c_1, a data mining model consisting of rules bearing high certainty and so on. After then, we heuristically determined intervals or exact values of parameters in the configuration templates of the context state based on each of their requirements.

Resource constraints dimension in Figure 7.7 shows the resource states simulated by EDG during the experiment. c_3's and c_4's all state combinations were not used instead a subset of c_3's and c_4's states were selected to create five resource constraints for the experiment. In order to produce *scarce memory* condition, we setup EDG to consume all the memory leaving only an amount which is equal to 10% of the size of the data set to be mined whereas for *sufficient memory* available memory left was equal to 50% of the size of the data set to be mined. At *CPU bottleneck* and *sufficient CPU* situations 10% percent and 70% of available CPU were left respectively.

We run Apriori under every resource state given in Figure 7.7 ten times with each configuration generated from every configuration template of c_1's and c_2's state combinations. Hence, we produced 1500 execution records.

Finally, c_3 and c_4's (resources') usage measures by Apriori and quality indicators

from the data mining model generated by Apriori are collected by EDG to constitute the base for the class label formation (line 11 in Table 7.3). In the next subsection, we explain in detail the transformations made on the data mining quality and the taxonomy used in the experiment.

Data Mining Quality Transformations and Taxonomy

In the execution data of Apriori, we had eight quality attributes that we applied discretization, aggregation and abstraction operations in order to produce the class labels for decision tree. Let $Q(q_{111} : D_1, q_{112} : D_2, q_{121} : D_3, q_{122} : D_4, q_{13} : D_5, q_{211} : D_6, q_{212} : D_7, q_{22} : D_8)$ be the relation schema defining the quality attributes in the *execution* file of the experiment.

Firstly, we discretized each quality attribute since associated domains of each $D_i, i = 1, ..., 8$ were continuous. Nominal values for class label attributes were obtained by using unsupervised discretization filter of Weka. There are two strategies for discretization: equal-interval and equal-frequency binning.We have chosen equal-intervals for the bins because data mining quality ranges which have low number of tuples are better preserved compared to equal-frequency binning. For example, with equal-interval binning, the minimum range of memory usage observed as the result of the executions is preserved as a separate bin even though the number of executions that use memory in the minimum range is not high. Additionally, rather than using a constant value for the number of bins, we preferred the well-known method, entropy-based discretization that utilizes entropy of intervals to determine the number of bins. As a result, data defined by Q was transformed to comply with Q_D given in Definition 6.

Secondly, we aggregated the attributes in Q_D to generate aggregated data mining quality which is defined by Q_A (Definition 8). The aggregation function that we used consists of three simple steps:

- encode bins in the associated domain of every Q_D's attribute with ordinal values,

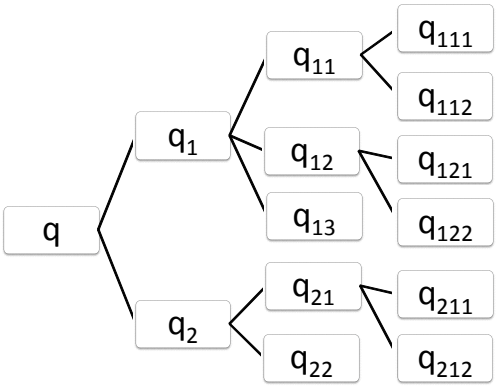


Figure 7.8: Data mining quality taxonomy used in the experiment

- find the ordinal value for every tuple's every attribute in Q_{D_I},
- concatenate in the order they appear in Q_D, all the attributes' ordinal values of each tuple in Q_{D_I}.

Next operation on experiment data, is to generate the abstract data mining quality attributes. Data mining quality taxonomy given in Figure 7.8 was used for this purpose. We again prefer to use symbols instead of the names describing the *execution* file attributes and the abstract attributes. On the other hand, corresponding attribute names can be found in Table 7.4. As can be seen in Figure 7.8 abstract data mining quality attributes are $Q_G = \{q, q_1, q_2, q_{11}, q_{12}, q_2, q_{21}, q_{22}\}$. First of all, domain of each abstract data mining quality attribute in Q_G was determined. Afterwards, mappings from the domains of the attributes in the abstract data mining quality attribute's predecessor set to its domain were defined for each element of Q_G. For these mappings, we used either a two or three dimensional coordinate system depending on the number of attributes in the predecessor set of the abstract data mining quality attribute (Figure 7.9). The axes of each coordinate system were labeled by the ordinal values assigned to the bins in the domains of the attributes in the predecessor set. The space represented by the coordinate system was divided into areas in two dimensional coordinate system and into cuboids in three dimensional coordinate system where each area/cuboid was assigned a corresponding value from the domain of abstract data mining quality. Figure 7.9(a)

Table 7.4: Attributes corresponding to symbols in taxonomy

q_{111}	average memory usage
q_{112}	maximum memory usage
q_{11}	memory usage
q_{121}	total processor time in msec
q_{122}	total number of processor cycles
q_{12}	processor usage
q_{13}	duration
q_{1}	resource usage
q_{211}	model minimum support
q_{212}	model minimum confidence
q_{21}	interestingness of the model
q_{22}	number of rules in the model
q_{2}	model quality
q	overall quality

shows how we mapped the domains of q_{111} and q_{112} to the domain of q_{11}. Both q_{111} and q_{112} have nine bins in their domain sets. The ordinal values that are associated with the bins label the axes. For this example, we combined three consecutive bins from the domains of each attribute (q_{111} and q_{112}) to map to a member in the domain of q_{11}. In this way, we reduced the size of q_{11}'s domain from eighty one to nine. Similarly, Figure 7.9(b) shows how three domains are mapped. Afterwards, we used the mappings to generate the abstract data mining quality (G in Definition 11) for *execution* file.

Finally, fifteen class label attribute sets were formed in L_{set} from the taxonomy by enumeration (Definition 10). In the *execution* file, the ordinal values of attributes in each of the fifteen class label sets were aggregated and fifteen alternative class label attributes were formed (Q_{AL} in Definition 11).

7.3.1 Experiment Results

During the experiments, transformed content of the *execution* file was classified by building a separate decision tree for each of the fifteen class label attributes obtained from each member of L_{set}. J48 classifier of Weka was used for classification.

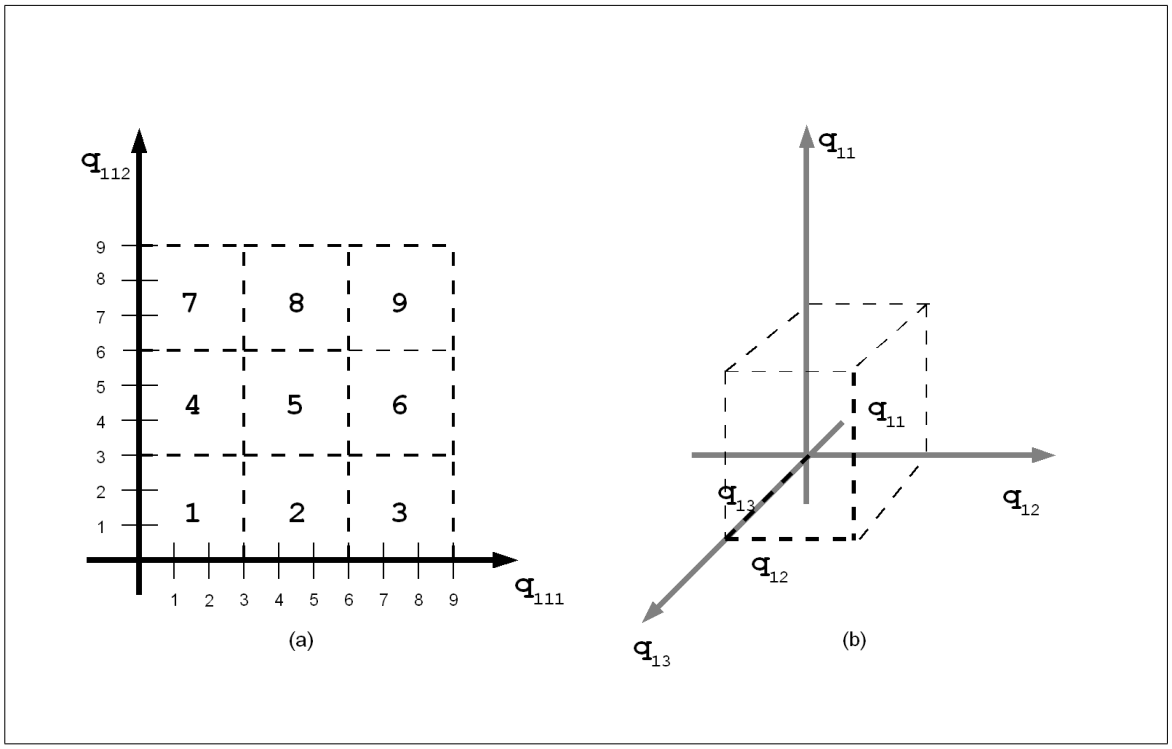


Figure 7.9: Mappings from predecessor set domains to abstract domains

Analysis of AS/BM Strategy

We first analyzed the decision tree models to justify that data mining quality abstraction was necessary and also to understand the significance of finding a model balanced in terms of accuracy and specificity. For this purpose, we compared the accuracies of the decision tree models which classify experiment data by various data mining qualities. The specificity degree versus the accuracy for each decision tree model is plotted in Figures 7.10a,b. The X axis shows the decision tree models that are ordered by their specificity degree. Decision tree's specificity degree which was computed by using Algorithm 2, indicates the specificity of the information that the class label attribute has. The decision tree specificity degrees in Figures 7.10a,b were normalized by dividing to the specificity degree of the decision tree that had the highest specificity. In Figure 7.10a, accuracy was computed from the training data which was used to build the decision tree whereas accuracy in Figure 7.10b was computed by using ten-fold cross validation as suggested in Definition 12. As usual, training data accuracy is higher than

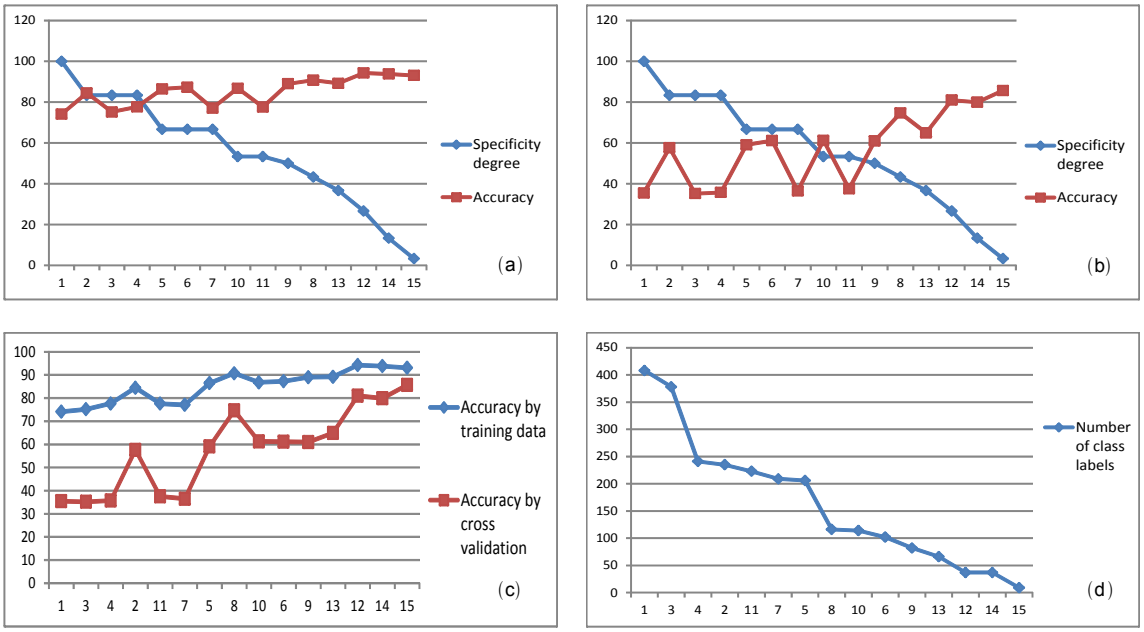


Figure 7.10: Analysis of decision tree models

generalization accuracy estimated by cross validation.

General trend observed in both of the graphs is that the accuracy of the decision tree increases as its specificity degree deteriorates. Accuracy derived after ten-fold cross validation is very low for some of the decision tree models. Clearly, if the model that provides most specificity was used for configuration decisions, without leveraging its accuracy by abstracting a subset of the data mining features, predictive accuracy would be very low. Hence, we conclude that abstraction of data mining quality is necessary.

However, accuracy is not always better when specificity is less. If a model having an average specificity without estimating its accuracy, is chosen by assuming that it will provide an average accuracy, it is a possibility to have the lowest accuracy. For instance the decision tree model 7 in Figure 7.10b. Therefore, considering only the specificity of the model when choosing the most appropriate decision tree for parameter configuration is not sufficient.

These results are in accordance with our predictions and explain the reason why we

proposed our AS/BM strategy to choose a model that possesses a balanced amount of accuracy and specificity.

Analysis of the Pre-Screening Presumption

Decreasing the number of decision tree constructions is the main reason for pre-screening. However decision trees are eliminated without estimating accuracy in the pre-screening phase. In this section, we question whether among the pre-screened ones are there decision tree models which have high accuracy-preciseness scores.

While pre-screening we presumed that the predictive accuracy of a decision tree is low if the associated class label attribute contains a high number of (garbage) classes that do not have representative examples in the training data. To validate the presumption, we contrasted decision tree models in terms of the number of class labels they have and their accuracy. In Figure 7.10c, we plotted the decision tree models' accuracy figures derived from training data and computed by ten-fold cross validation respectively by ordering the decision tree models according to the number of classes they possess. Figure 7.10d shows the number of classes that decision tree models have. According to the results, accuracy generally deteriorates as the number of classified class labels increases which complies with the presumption.

Furthermore, we applied the pre-screening criteria given in Algorithm 1 to determine the class label attributes that we expected to classify poorly due to high number of garbage classes. In Figure 7.11, we compare the predicted accuracy figures of the decision tree models against the number of garbage classes their class label attributes have. In general, it is possible to say that there is an aggravating effect of garbage classes on the accuracy.

We also computed the score of each decision tree model by using Algorithm 2. The following list ranks the decision tree models by their score:

$$(8, 12, 2, 6, 15, 5, 14, 10, 9, 13, 1, 4, 3, 7, 11)$$

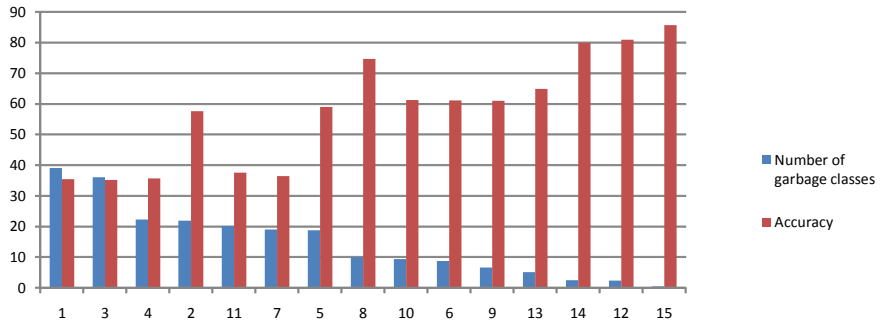


Figure 7.11: Effect of garbage classes on the model's accuracy

Final observation supporting pre-screening presumption is that, five out of six class label attributes that are most likely to be eliminated by pre-screening (first six bars in Figure 7.11) are among the class label attributes of six worst scored decision tree models. Hence, it is possible to say that pre-screening eliminates the decision trees that are very unlikely to be selected as the appropriate model for configuration decisions by Algorithm 2.

Assessment of Configuration Decisions

In this part of the experiment, we derived configuration decisions from the selected decision tree model and subsequently we used the derived configurations to configure Apriori. The purpose of this experiment is to compare the quality attained by Apriori executions which were run by a derived configuration against the quality that is predicted from the decision tree model for the derived configuration. We accomplished this experiment in three main steps:

Extract Configuration For configuration extraction, the decision tree model that classifies by the aggregation of the attributes in the set $\{q_1, q_{211}, q_{212}, q_{22}\}$ was used since it was found to be the highest scored model. We obtained decision rules from the decision tree model (that will be referred as dt_8 thereafter) so that data mining quality class memberships of configurations are logically represented. An example decision rule which consists of parameter setting predicates and the corresponding aggregated data

mining quality class, is as follows:

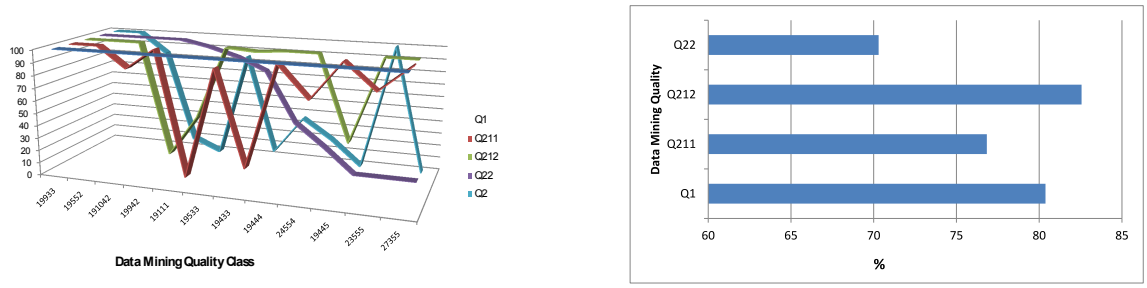
$$P4 <= 0.668\ AND\ P2 > 0.879\ AND\ P5 > 0.324\ AND$$

$$P5 <= 0.429\ AND\ P4 > 0.526\ AND\ P2 <= 0.976 :\ 19552$$

Note that, reverses of the data mining quality class abstraction and aggregation functions (Section 7.3) applied respectively to the data mining quality class give the individual quality predictions by the decision rule. For instance, the predicted data mining quality (19552) for the configuration in the example decision rule indicates high support, high confidence model having number of rules below average obtained by average memory and high CPU usage within a short execution time. In fact, data mining quality predictions are associated to the cube of circumstances given in Figure 7.7 because we executed Apriori for the circumstances in Figure 7.7. For example, data mining quality (19552) must be attained under the circumstance where high certainty rules ($c_1 = 2$) having highest degree of usefulness ($c_2 = 1$) are needed in spite of the CPU bottleneck ($c_4 = 1$) and barely sufficient memory ($c_3 = 2$) conditions in the device. The number of decision rules formed from dt_8 is 144 bearing 116 different classes.

In order to use for Apriori configuration in the next step, we formed a configuration template from each decision rule related to a circumstance in Figure 7.7. Parameter settings in a configuration template are ranges of values where boundaries are constituted by either the existing predicates in the decision rule or the highest/lowest possible settings of the parameters whenever predicate for the boundary is nonexistent. Although resource usage was abstracted in dt_8, we obtained fine usage figures for memory and processor as well as the duration of the data mining process after decoding q_1 so that we generated recommendations for specific resource usages rather than overall resource usage. When multiple decision rules were obtained for the same circumstance, we eliminated the ones other than the decision rule that has the highest number of classified instances.

In short, we extracted configuration templates that each one is predicted to achieve a specific data mining quality in this step.



(a) Successful recommendations by class

(b) Successful recommendations in overall

Figure 7.12: Assessment of recommendations derived from decision tree

Execute Apriori with Derived Configurations The configuration templates extracted in the previous step were used to configure Apriori while running it via EDG. During the verification runs of Apriori, if the corresponding decision rule indicated a circumstance, that circumstance was simulated while executing Apriori. In this step, Apriori was run 724 times until sufficient number of executions resulting in designated data mining quality were collected.

Verify the Configuration Decisions In the final step, we assessed the appropriateness of configuration decision rules. For this purpose, we made use of the quality measurement figures collected during the Apriori runs in the previous step. As we did when forming the class labels for the decision tree model, we abstracted and aggregated the data mining quality attributes in these execution records using the functions given in Section 7.3 to form the "*realized*" data mining quality. Afterwards, we compared the "*realized*" data mining quality of each Apriori that ran with a configuration derived from a decision rule against the data mining quality class of the same decision rule.

Percentages of successful recommendations for a sample set of data mining quality classes are given in Figures 7.12(a). We selected a representative sample of classes to illustrate different levels of data mining quality objectives achieved. Percentages are plotted for each individual data mining quality attribute in the set $\{q_1, q_{211}, q_{212}, q_{22}\}$ as well as the combined model quality q_2 which is the aggregation of attributes in the

set $\{q_{211}, q_{212}, q_{22}\}$. We tested the equality of "*realized*" data mining quality and its class while calculating the percentages. On the other hand, "*realized*" resource usages (q_1) of the classes given in Figure 7.12(a) always indicated lesser consumption than their respective classes from which the recommendations were formed. Therefore, it is reasonable to accept that the resource usage objectives of the recommendations are satisfied. For this reason when plotting the percentages of successful recommendations in Figure 7.12(a), we considered all recommendations were successful in terms of resource usage (q_1).

Percentages of successful recommendations in overall are given in Figure 7.12(b) in which the percentage of the Apriori executions which achieve the objective of the parameter settings are grouped by the relevant quality measurement. In Figure 7.12(b), when calculating the successful recommendation percentages, we looked for an exact match between the "*realized*" data mining quality and the data mining quality class of its configuration decision rule. Although the percentage of executions that do not satisfy resource usage objective is around 19%, only 2% of the recommendations results in higher resource consumption (q_1) than the designated objective which means that better resource usage were achieved.

We proposed a mechanism to automatize data mining configuration based on the argument that a specific circumstance requires a specific data mining quality. As the final step of verification, we compared the experiment results to a baseline where there is no automatization but default values were used for parameter settings. For this purpose, we ran Apriori with the default settings of Weka and collected resource usage and resulting data mining model quality indicators to form a baseline. When compared to the baseline, Apriori executions that had been configured in the experiment (using dt_8) to optimize the related resource had 20% less memory usage and 88% less cpu usage. Also, when run with a dt_8 derived configuration with the objective to minimize the runtime of data mining, the elapsed time of Apriori had been 90% less compared to the baseline. Minimum support and minimum confidence of the data mining model generated by Apriori with default configurations were 0.4 and 0.91 respectively. On the

other hand, if either highest support or highest confidence rules are required, configurations derived from dt_8 generated data mining models with minimum support value of 0.8 and minimum confidence value of 1 respectively. If the parameters of data mining are not tuned, it is a possibility that data mining could not produce any model. In our case, although the default settings of Apriori resulted in a model, the data mining quality obtained was far below the figures that we had obtained by running Apriori with the configurations derived to optimize a specific resource usage or data mining quality indicator.

Impact of the Proposed Approach on Android Device's Resources

In this section, we assess the overhead of behavior model generation and its deployment to the system. Behavior model generation and deployment are two independent processes as can be seen in Figure 7.13. Every configuration of data mining does not trigger the generation of a new behavior model, on the contrary, behavior model is generated once and is deployed repeatedly until it decays. The decay of the model can be assessed by comparing the data mining quality realized against the data mining quality predicted after each mining of data with the recommended configuration. The only case which requires the behavior model to be re-built is when the percentage of successful recommendations for a data mining quality class drops below a threshold value (ξ).

In the experimental evaluation after configuring and running Apriori with extracted recommendations from the behavior model, we detected predictions of varying accuracies for different data mining quality classes (Figure 7.12). Experiment results indicate that there is a need to increase the predictive accuracy for the classes which have successful recommendations below ξ by supplying more training data. It is reasonable to transfer merely the execution records pertaining to data mining quality classes where percentage of successful recommendations are below ξ so that the accuracy of predictions are improved while the growth of the input for behavior model generation process is kept minimal.

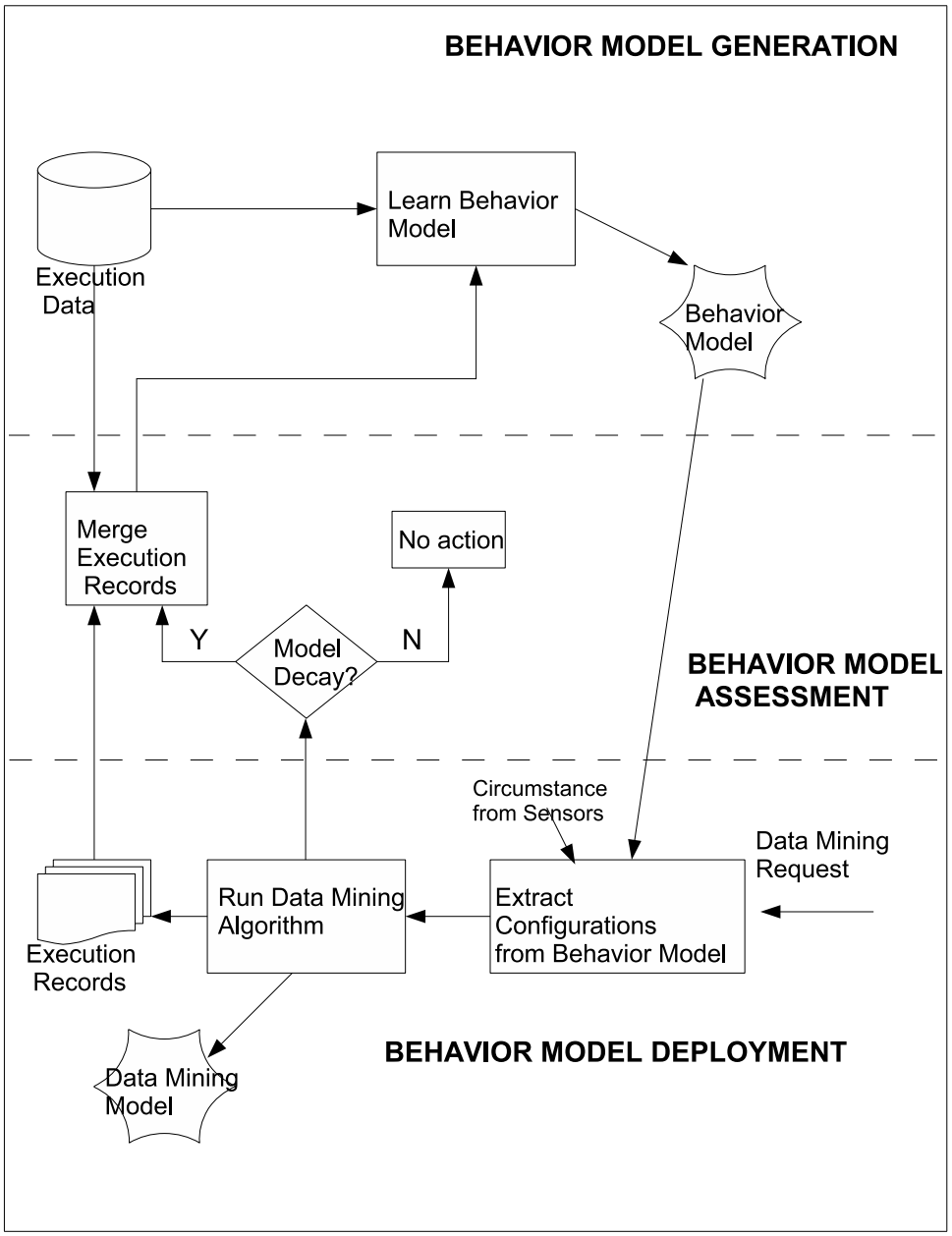


Figure 7.13: Processes for self-configuring data mining

The overhead of behavior model deployment is minimal since the worst case complexity of classifying by data mining quality from a behavior model at hand is $O(d)$, where d is the depth of decision tree. The depth of the decision tree that we used in the experiment (dt_8) was 16 which implies 16 accesses at most for each configuration recommendation.

On the other hand, since behavior model generation is much more computationally intensive, we evaluate the feasibility our approach by measuring the behavior model generation although it is expected to run much less frequently. For this reason, we constructed the decision tree models on an Android device which runs one of the prominent mobile operating systems. The Android device that we used for this purpose is Sony Xperia Tablet Computer, SGPT12 model. Operating system installed on the device is Android 4.0.3, kernel version 2.6.39.4. The tablet runs on a 1.4GHz Nvidia Tegra 3 CPU with 1 Gbyte of RAM. Device is equipped with 16 Gbytes of internal storage and 16 Gbytes of storage on SD CARD.

In order to find out the impact of our approach on Android operating system, Weka libraries ported to Android platform were used for decision tree construction. We measured the overhead of the same decision tree learning algorithm (J48) that we used in the experiments and we supplied the same training sets. We applied the pre-screening (Algorithm 1) and eliminated seven decision tree models by pre-screening. Eight out of fifteen possible decision tree models need to be constructed to estimate their predictive accuracies. On the Android device, the total elapsed time to construct eight decision tree models left after pre-screening was 5.44 minutes whereas longest and shortest run times of J48 were 57 and 17 seconds respectively. Since behavior model generation is independent of its deployment for configuring data mining, it runs as a background process but it must still end in a reasonable time range. The total elapsed time that we measured for behavior model generation on an Android device can be considered as acceptable in that respect.

We also analyzed the memory and CPU usage of J48 which learns behavior model from execution related data on an Android system. While constructing eight decision

tree models left after pre-screening, highest peak memory usage observed for J48 was $55Mbytes$ whereas average peak memory usage was $49Mbytes$. We observed that J48 is a cpu-intensive task since almost 90% of its runtime is accounted for CPU usage. Battery level of the device decreased by 2 percentage during entire executions of J48.

We conclude that, the overhead of deployment of an existing behavior model on the system is negligible. Behavior model generation takes some time but it does not require real-time computing and is expected to be much less frequently run. Furthermore, although behavior model generation is a cpu-intensive task, it does not cause a cpu bottleneck in the system since it runs in the background with low priority.

Chapter 8

MINING SOCIAL MEDIA DATA ON ANDROID DEVICE

As the final step of our study, we carried out the experiments on a device running Android operating system. We considered FESTweets, the movie recommendation application that we described in Section 6.2 as the example Android application. In order to use for experiment setup, execution data generator (EDG) software that is introduced in Section 7.1 was modified to run on Android platform. As mentioned before, EDG calls Apriori to collect execution data. Since the example application of this experiment is a movie recommender, we used a movie ratings data set to mine with Apriori. In the next section we provide details on the movie ratings data set.

8.1 Movie Ratings Data Set

Grouplens Research Project ([55]) which is a research group in the Department of Computer Science and Engineering at the University of Minnesota has collected movie ratings from the MovieLens web site ([50]). The purpose of the web site is to generate recommendations for the users as well as to collect research data. The data was collected during the seven-month period from September 19th, 1997 through April 22nd, 1998 and has been cleaned up - users who had less than 20 ratings or did not have complete

demographic information were removed from this data set ([55]).

After making the necessary transformations, we used the movie ratings data set that is made available by Grouplens Research Project as the mining data set of Apriori in this experiment. A brief description of data before any transformations, is as follows:

- There are $100,000$ ratings of 943 users in the data set.
- Users rated 1682 different movies.
- Movies are rated on a scale of $1 - 5$.
- Each record in the data set consists of user id, movie id, rating, timestamp fields.

We first converted the movie ratings data set to the input format of Apriori where attributes of the data set correspond to the movies and each record holds the movie list of a single user which he scored as liked. While performing the conversion, movies that are rated greater than 3 were accepted as liked, thus the corresponding attributes of these movies were marked as *true* while all others were marked with *?* indicating not liked.

We have done a simple research on the number of movies aired on the well-known domestic and international film festivals and we have found out that there are usually between 100 to 200 movies in the festival programs. Since we aim to generate recommendations for a film festival, the number of movies in the movie ratings data set is too high and should be reduced. Instead of eliminating the movies randomly, we eliminated the movies which are liked only by at most 12% of the users.

A brief description of data after the mentioned transformations is as follows:

- There are 670 records (correspond to transactions in frequent itemset terminology) in the data set where each user has one record.
- The number of attributes of the data set (correspond to items in frequent itemset terminology) is 135 where each attribute corresponds to a movie.

- Movies liked by the users are represented by the value *true* otherwise a *?* is coded so that Apriori searches associations among *true* values only.
- 68.5% of the users liked more than 20 movies.
- The movie which has the highest number of likings, is liked by 463 users. The movie which has the least likings is liked by 71 users.

We refer to transformed movie ratings data set as *data set of movie lists* from thereafter.

8.2 Frequent Itemset Mining with Apriori

As in the experiments performed in other platforms, EDG that we run on Android also executed Apriori through Weka API calls. Before giving specifics about the class weka.associations.Apriori, we provide a brief information on Apriori as introduced in [2].

8.2.1 Apriori Algorithm

Apriori is an algorithm for mining frequent itemsets for Boolean association rules. Market basket analysis which is used for understanding the buying habits of the customers is a typical example of frequent itemset mining. In this sense, discovering associations among the likings of the festival audience for generating recommendations is similar to market basket analysis.

Apriori is an iterative algorithm such that at each iteration k, frequent *k-itemsets* (L_k) are extracted where *k-itemsets* is a set of itemsets each having k items. Moreover, an itemset is frequent if it satisfies the *minimum support threshold*. Initially, frequent *1-itemsets* are determined by counting the items in the input data set. Afterwards, frequent *k-itemsets* are extracted by the following actions:

Table 8.1: Parameters of Weka implementation of Apriori

	Parameter	Option
1	Requested number of rules	-N
2	Minimum confidence of a rule	-C
3	Delta for minimum support	-D
4	Upper bound for minimum support	-U
5	Lower bound for minimum support	-M

- Generate candidate itemsets C_k by joining L_{k-1} with itself.
- Prune all itemsets in C_k that have some $(k-1) - subset$ not in L_{k-1}.
- Obtain frequent *k-itemsets* L_k from the itemsets in C_k that satisfy *minimum support threshold*.

Actions listed above are iterated until L_k is empty. Frequent itemsets obtained by Apriori satisfy the *minimum support threshold*. It is possible to generate strong association rules from the frequent itemsets extracted by Apriori. Strong association rules satisfy *minimum confidence threshold* as well as *minimum support threshold*. In-depth information about Apriori can be found in [36] and [60].

8.2.2 Weka Implementation of Apriori

Weka implementation of Apriori (JAVA class weka.associations.Apriori) accepts five principal parameters that affect the data mining model generated (Table 8.1).

Parameter settings of Apriori by Weka influence the resulting data mining model according to the following principals ([61]):

- It is aimed to generate the requested number of rules (-N). The number of rules in the resulting data mining model never exceeds -N but the strong association rules that satisfy the requested minimum support and minimum confidence might be less than -N.
- Algorithm starts searching frequent itemsets by making use of the minimum support given by -U.

- Algorithm repeatedly searches for frequent itemsets by decreasing an amount of -D from the minimum support in each step.
- When either of the following occurs, algorithm stops searching for frequent itemsets
  - requested number of rules (-N) with the required minimum confidence (-C) are found,
  - the support has reached the lower bound given by -M.

It is also possible to set lift, leverage or conviction as the type of metric instead of confidence to rank the rules. If a metric other than confidence is set, a minimum value for that metric (metric score) should be given and therefore minimum confidence is not the parameter of Apriori in such a configuration. The principals for generating a data mining model given above still apply except the first stop condition. Frequent itemset mining stops when requested number of rules (-N) with a score above the required metric score are found.

8.3 DM Model for Movie Recommendations

Movie recommendations are extracted from the association rules which are mined by Apriori from the *data set of movie lists*. A subset of the association rules generated by a particular Apriori execution is given in Figure 8.1 whereas complete data mining model can be found in Appendix C. We obtained the output given in Appendix C by running Apriori through Weka Explorer in order to explain the data mining model. Nevertheless, Weka API that is called by EDG also returns all the information that is present in the output of Weka Explorer.

The configuration of Apriori for this execution instance is marked by green in Appendix C. Note that, it is requested to rank to rules by lift hence the value provided by means of the parameter -C is the minimum lift score. As can be seen on the out-

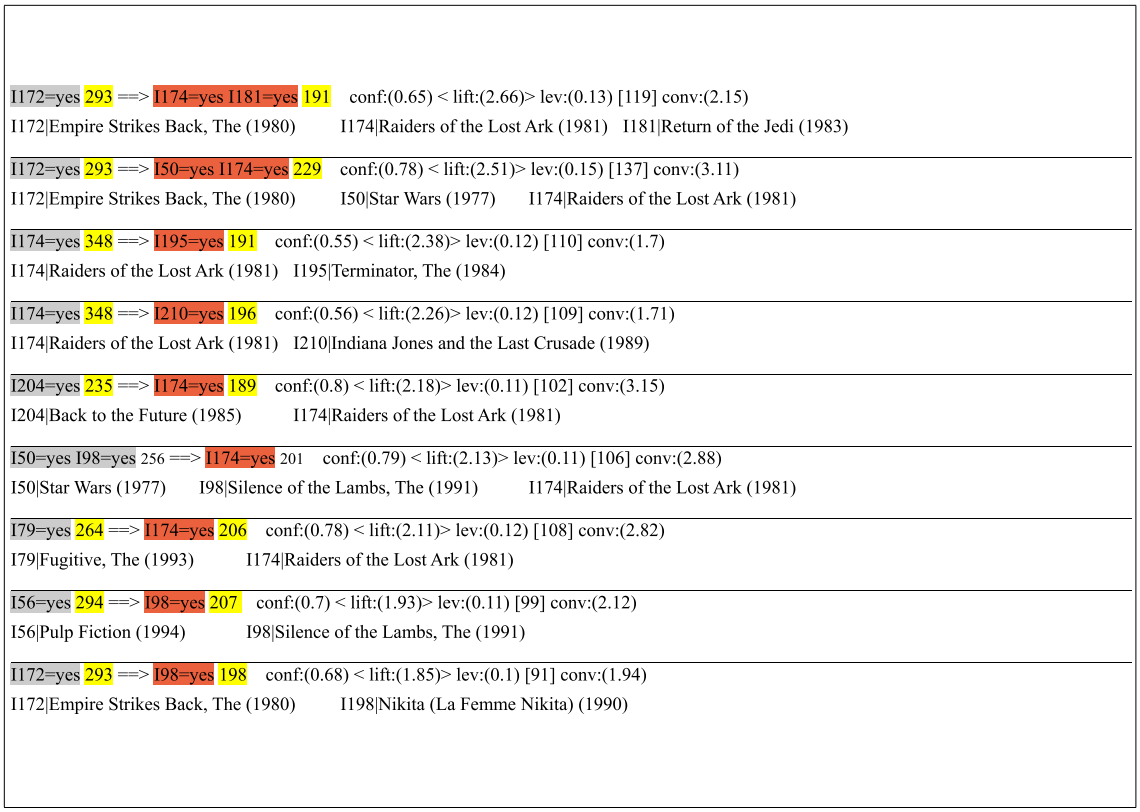


Figure 8.1: Movie recommendations.

put, minimum support was decreased to 0.2 (blue marked line in Appendix C) in 18 iterations (orange marked line in Appendix C) until a data model with 50 rules that satisfy the requested lift is found. This means that frequent itemsets were mined 18 times before the requested data mining model is acquired.

Since labels are used instead of the movie titles in the *data set of movie lists*, movie titles do not appear on the association rules. In order to improve understanding, the matching movie titles of the movies occurring in each association rule are copied underneath the rule in Figure 8.1. Each rule has a premise (marked grey in Figure 8.1) and a consequence (marked red in Figure 8.1) preceding and following the $\Rightarrow$ symbol respectively. The number that is marked yellow in the premise is the support of premise. First rule in the figure indicates that, the number of users in *data set of movie lists* who liked "The Empire Strikes Back" is 293. The number in the consequence (again marked yellow) is the support involving both premise and consequence. So that, 191

users liked all three films: “The Empire Strikes Back”, “Raiders of the Lost Ark” and “Return of the Jedi”.

Four metrics, confidence, lift, leverage and conviction, measure the interestingness of every rule in the model (Figure 8.1). Since the metrics are the means to individually evaluate the rules as well as the data mining model in overall, choosing which metric to optimize is important while configuring Apriori. Therefore, we discuss the metrics in detail. Association rule $A \Rightarrow B$ is assumed throughout this section. $A \cup B$ means that both A and B appear in the transaction. Both A (premise of the rule) or B (consequence of the rule) may represent a single item (movie) or a set of items (movies).

confidence : Confidence is the conditional probability that a transaction having A also contains B. It states an explicit percentage (that transactions having A also contains B) and is a measure of certainty for a rule. Confidence is given by:

$$conf(A \Rightarrow B) = \frac{supp(A \cup B)}{supp(A)} = \frac{P(A \cup B)}{P(A)} = P(B|A) \tag{8.1}$$

Thus, $conf(A \Rightarrow B) \neq conf(B \Rightarrow A)$.

A problem with confidence is that support of consequent is not taken into account in its computation. If the consequent of the rule has higher support than the confidence of the rule (i.e. $P(B) > \frac{P(A \cup B)}{P(A)}$), due to the formula used to calculate the confidence, a very high confidence for the rule could be computed although strong association among the items in the premise and consequent does not exist.

lift : Lift measures how likely for A and B to occur together than expected if they where statistically independent. It is a measure of dependent or correlated events. A positive lift value, implies that A and B are dependent and gives the degree of

dependence. Lift is denoted by the following equation:

$$lift(A \Rightarrow B) = lift(B \Rightarrow A) = \frac{conf(A \Rightarrow B)}{supp(B)} = \frac{conf(B \Rightarrow A)}{supp(A)} = \frac{P(A \cup B)}{P(A)P(B)} \quad (8.2)$$

Lift is symmetric and thus measures co-occurrence not the implication.

conviction : Conviction is the proportion of the probability that A occurs without B (if A, B are dependent) to frequency of A occurs without B. Conviction is given by the equation:

$$conv(A \Rightarrow B) = \frac{1 - supp(B)}{1 - conf(A \Rightarrow B)} = \frac{P(A)P(\neg B)}{P(A \cup \neg B)} \quad (8.3)$$

Conviction measures the implication adequately and takes into account both $P(A)$ and $P(B)$.

leverage : Leverage is a variation of lift such that leverage finds the difference between the frequency of A and B occurs in the data set and the probability that A and B occurs independently. The following equation is used for computing leverage.

$$leverage(A \Rightarrow B) = P(A \cup B) - P(A)P(B) \quad (8.4)$$

Further information on lift, conviction and leverage can be found in [7] and [54].

In conclusion, the data mining model metrics discussed in this section, support, confidence, lift, leverage and conviction of the model as well as the number of rules returned by the data mining model constitute the quality indicators of the data mining model.

8.4 Android Operating System

Android is an open source operating system for mobile devices such as smartphones and tablet computers. Android is a Linux-based operating system. In that respect, basic operating system tasks such as I/O management, memory management, process management, security and so on are handled by Linux kernel. Instead of JVM (Java Virtual Machine) which is a stack-based architecture, a register-based architecture DVM (Dalvik Virtual Machine) is introduced to run the applications on Android. Register-based VM has the advantage that the number of VM instructions is substantially reduced when compared to a stack-based VM.

Memory Management: DVM is designed to run the executables (dex files) which are generated from JAVA class files by the "dx" tool. Due to the reason that Android devices are memory constrained devices, DVM is designed to optimize the memory allocation of the applications at runtime. The file format of the executable is improved to obtain minimal memory footprint. JVM stores constants (such as string constants, field, variable, class, interface and method names) used in the code in the private heterogeneous constant pool of each class file whereas DVM stores the constants in a single shared constant pool. In this way, duplication of constants across class files is prevented. Debugging and monitoring of Android applications is possible by DDMS (Dalvik Debug Monitor Server) which is integrated into Eclipse ([25]). DDMS works with both the emulator and a connected device. Screen capture of heap information provided by DDMS for an application can be seen in Figure 8.2.

In the experiments, Android API library is used to collect memory usage data. Runtime class (http://developer.android.com/reference/java/lang/Runtime.html) is called by EDG during execution of Apriori to calculate its average and maximum memory usage.

Figure 8.2: Eclipse DDMS

Process Management: Android performs multitasking which is crucial for the mobile device users to switch among the opened applications instantly but at the same time multitasking in a mobile device where memory is constrained is challenging. Android does not allow to close the applications (unless force stopped) so that users have a wide range of applications at their disposal all the time. On the other hand, so many open applications consume memory which should be avoided in mobile devices. Process management of Android is designed to allow multitasking while avoiding out of memory conditions. Android keeps the application's process in "running" state although the application is sent to background as a result of user switching to another application. If the background application has more work to do, Android allows it to continue working. Conversely, if the background application has no more work to do, it is still kept in the "running" state. In either case, application appears in the foreground instantly when the user switches to that application. In order to avoid out of memory condition, Android may force-kill the applications by considering their priority. However, Android keeps the last-state of the application that is force-killed so that if a user later switches to a force-killed application, its state is resumed. Application priority is determinant on choosing the process to kill. Application states which imply the priority of the process are: *active, visible, started service, background, empty.*

Substantial information on Android operating system can be acquired from the web sites [24] and [62].

8.5 Configuring Apriori for Movie Recommendations

It is aimed by the the movie recommendation application (FESTweets) to generate a list of recommendations for the movies in the festival program so that users can sift through the list before buying tickets. We suggest to use Apriori to extract a data mining model that reveals the common likes of the users. It is important to note that, data

mining model extraction for recommendation gathering is not a one-time process, on the contrary, repeated mining of *data set of movie lists* will produce new recommendations since more users will enter their movie lists over time. Prior to mining *data set of movie lists* each time, two questions need to be answered: what are the expectations from the data model (how many recommendations are enough, what certainty is expected from the recommendations and so on) and what are the processing constraints. Therefore, considering that movie recommendations are generated on a mobile device, we discuss in the next section what could be the requirements of data mining model and processing under different circumstances.

8.5.1 Circumstance/Quality Mapping

Date: FESTweets recommendations would be needed throughout the ticket sales period which always begins days before the festival start date and continue during the festival period. Before the festival period, users have plenty of time to decide so it is reasonable to provide them more options during this period whereas during the festival period users are tight on time for movie selection so offering them only significant movie recommendations would be appropriate. We conclude that generating a data model with lower minimum support before the festival period will provide users the chance to examine also the rare movie lists. On the other hand, when there is a time constraint for deciding, it is better to provide movie lists which are supported by higher percentage of users. Therefore, we determined **minimum support** of the data model as the effecting data mining quality for models generated at different **dates**.

Time: Another determining factor of the data mining model is the time of the day when the movie recommendations are asked for. In general, people are more busy during working hours but they may still want to glance at the movie lists. Few recommendations of high certainty is convenient during the expected busy hours whereas during off hours there is time for sifting through longer recommendation lists involving recommendations of lower certainty. As we mentioned in Section 8.3, there are four metrics

that measure certainty. Lift suits best for our purpose because confidence is not reliable all the time due to the drawback that in its computation support of the consequence is not considered. Moreover, lift is a measure of co-occurrence which is sufficient for measuring the certainty of associations among favorite movies of users. Hence, **lift** is the measure for the appropriateness of the model generated during certain **time**s of the day.

Location: If a user needs a movie list recommendation in or in the vicinity of a cinema complex where festival movies are aired, a recommendation list should be produced as quickly as possible since the user is most probably about to buy tickets. This is the situation in which the duration of data mining model creation is critical. Thus, when user's **location** is one of the festival cinema complex, then success criteria of the model is the **duration** of data mining process.

Device's Resources: New recommendations may be needed when there is scarcity of device resources that are needed by the data mining process. A data mining model can still be build by sparingly using the scare resource. When there is scarcity of **available memory**, mining data with least **average memory** allocation is the best strategy while when there is **CPU bottleneck** mining data with least **CPU time** is aimed.

Note that the designated requirements for data mining model and processing do not specify how to configure Apriori. At the same time, it is not possible to tell with certainty how to configure data mining so that the designated requirements are fulfilled. For example, one can not tell what must be set for the Apriori parameters given in Table 8.1 so that a data mining model is extracted from *data set of movie lists* in shortest duration. For this reason, in this thesis, we proposed to learn the behavior model of the data mining algorithm (Apriori in this case) so that configurations which will generate data mining models satisfying the designated requirements can be discovered.

From this stage on, we explain the steps to extract behavior model of Apriori for mining *data set of movie lists* in order to determine the configurations that will most

likely fulfill the designated requirements for the data mining model and processing.

8.5.2 Training Data

Information collected during several executions of Apriori mining *data set of movie lists* was needed to construct its behavior model. We ran Apriori with *data set of movie lists* through EDG to collect training data for the behavior model. In each run, we configured Apriori with a different set of settings. We first determined the possible set of values for each parameter, afterwards we ran Apriori with every combination of the settings. Next, we explain in brief how we determined the possible set of values for each Apriori parameter given in Table 8.1.

- Upper bound minimum support (-U): If U is set higher than the support of the large itemsets in the data set, Weka at each iteration of frequent itemset mining, decreases the value given in U by the amount given in D until the real support of large itemsets in the data set is found. If U is very much higher than the support of large itemsets in the data set, high number of void iterations would increase the overall processing cost. In order to set a practical U value for the experiments, we ran Apriori once with the configuration given below to find the support of large itemsets in the *data set of movie lists*.

  ```
  -N 1 -C 0.01 -D 0.01 -U 1.0 -M 0.01
  ```

  For this run, we set U and M to the highest and lowest possible values respectively so that the highest support figure within the widest range can be detected. We also set a very low minimum confidence on purpose so that no rules extracted from the itemsets are eliminated. As a result, the obtained highest support of frequent itemset in the data set is 0.48. Therefore, we set U to 0.5 for all configurations.

- Requested number of rules (-N): We used a constant value for N (= 50) for all executions. The value we picked for N is large enough to generate sufficient

number of recommendations. By using a large value for N, we let the settings of C and M to determine the number rules in the data mining model since N is the upper bound for the number of rules generated but not an absolute value.

- Minimum confidence of a rule (-C): We ran Apriori with five minimum confidence settings given in the set $\{1, 0.9, 0.8, 0.7, 0.6\}$.

- Lower bound for minimum support (-M): M should be less than U. The values in the set $\{0.4, 0.3, 0.2, 0.1\}$ are used for the experiments.

- Delta for minimum support (-D): D determines how many iterations may exist between $U - M$. If D is small, the number of iterations increases which negatively impacts the processing cost of data mining. On the other hand a large D prevents fine setting of support. D settings used in the experiment are $\{0.1, 0.05, 0.02\}$.

The experiments were performed on Sony Xperia Tablet Computer, SGPT12 model. Operating system installed on the device is Android 4.0.3, kernel version 2.6.39.4. The tablet runs on a 1.4GHz Nvidia Tegra 3 CPU (quad-core CPU, and includes a fifth "companion" core) with 1 Gbyte of RAM. Device is equipped with 16 Gbytes of internal storage and 16 Gbytes of storage on SD CARD.

List of executions containing the subset of the fields from the output of EDG is given in Appendix D. Each line in the report given in Appendix D corresponds to an execution record. Note that EDG returns all of the information presented in Appendix C but in the report given in Appendix D we included only the fields that are used for behavior model construction. Support given in each line of the report is the minimum support of the data mining model. Confidence is the minimum of the confidences calculated for the rules in the data mining model whereas lift is the average of the rule lifts. We specified confidence as the type of the metric to rank the rules in the data mining model so that we guaranteed that all the rules have higher confidence than the minimum confidence requested. On the other hand, since we determined that average lift of the rules (that satisfy the minimum confidence requested) is effective

Table 8.2: Circumstance/quality mappings for movie lists mining

	CIRCUMSTANCE/ DM QUALITY	INTERVALS/ RATING SCALES					
C	DATE	(0,15]	(15,21]	(21,31]	(31,40)		
	(Remaining days to the festival end date)						
Q	SUPPORT	highest	average	below average	lowest		
C	TIME	(0,6]	(6,9]	(9,12]	(12,14]	(14,18]	(18,24)
	(Time of day)						
Q	LIFT	lowest	average	highest	average	highest	below average
C	LOCATION	(0,4]	(4,7]	(7,10]	(10,13)		
	(Proximity to film festival cinemas)						
Q	DURATION	maximum	average	below average	minimum		
C	AVAIL. MEMORY	(0,5]	(5,9]	(9,13]	(13,17]	(17,21]	
	(Amount of free memory on device)						
Q	AVGMEM	maximum	above average	average	below average	minimum	
C	AVAIL. CPU	(1,4]	(4,6]	(6,8]	(8,10]	(10,12]	(12,14]
	(Amount of free processor time in the device)						
Q	CPUTIME	maximum	above average	average	below average	far below average	minimum

for configuration decisions (discussed in Section 8.5.1), lift is also included in behavior model construction.

8.5.3 Behavior Model

Behavior model of movie lists mining was generated in the form of Bayesian network. Bayesian network was constructed using the mechanism outlined in Section 4.1. Circumstance attributes that are mentioned in Subsection 8.5.1, *date, time, location, available memory* and *available cpu* were added to the execution records given in Appendix D. We derived the contents of circumstance attributes that are added to each execution record from the quality attributes in the same record such that the quality attained by each execution was accepted as the indication of the related circumstance. Circumstance attributes were populated by using the mappings given in Table 8.2.

In every *C* labeled line of Table 8.2, a circumstance attribute and the possible intervals of values that we determined for that circumstance attribute are given. For instance, we divided the time of day into six intervals considering the level of busyness of

a person during a day and anticipated that recommending movie lists of high certainty is appropriate during too busy hours of a person. Likewise, we assumed that the festival period is fifteen days and we anticipated that the support of recommended movie lists should be higher during the festival days whereas necessary support of recommended movie lists gradually decrease as the number of remaining days to the festival end date increase. We rated the attained values of quality attributes which are shown in Q labeled lines of Table 8.2. In pairs of C, Q labeled lines of Table 8.2, quality attribute rating and the associated interval of values for the circumstance are given one under the other. In each execution record, after finding the associated interval for the circumstance, we assigned a randomly selected value from that interval. For instance, in an execution record, if the support of the data model is rated highest, date is assigned a value in the range $(0, 15]$ in that execution record.

After populating the circumstance attributes of the training data, Bayesian network structure was learnt from the training data using K2 algorithm. We used the K2 implementation of Weka ([35] which we modified as discussed in Section 4.1. We discretized *avgmem, cputime, duration* beforehand since K2 assumes variables are discrete. The Bayesian network (Figure 8.3) that is extracted by including a subset of circumstance attributes, Apriori parameters and a subset of data mining quality attributes is used to show how configurations of movie lists mining are inferred from a behavior model. It can be seen in Figure 8.3 that no relationships for the parameters N and U were discovered since we set constant values for them during the experiments.

8.5.4 Configuration Recommendations

We extracted movie list mining configuration recommendations from the behavior model given in Figure 8.3. Domain sets of circumstance and data mining quality attributes as well as circumstance/quality mappings that are used for this purpose are given in Table 8.2. Each parameter setting of Apriori that mines movie lists was inferred from the behavior model shown in Figure 8.3 by applying the Definition 5. We explain the behavior model inferences by referring to the constructs used in Definition 5. C_I

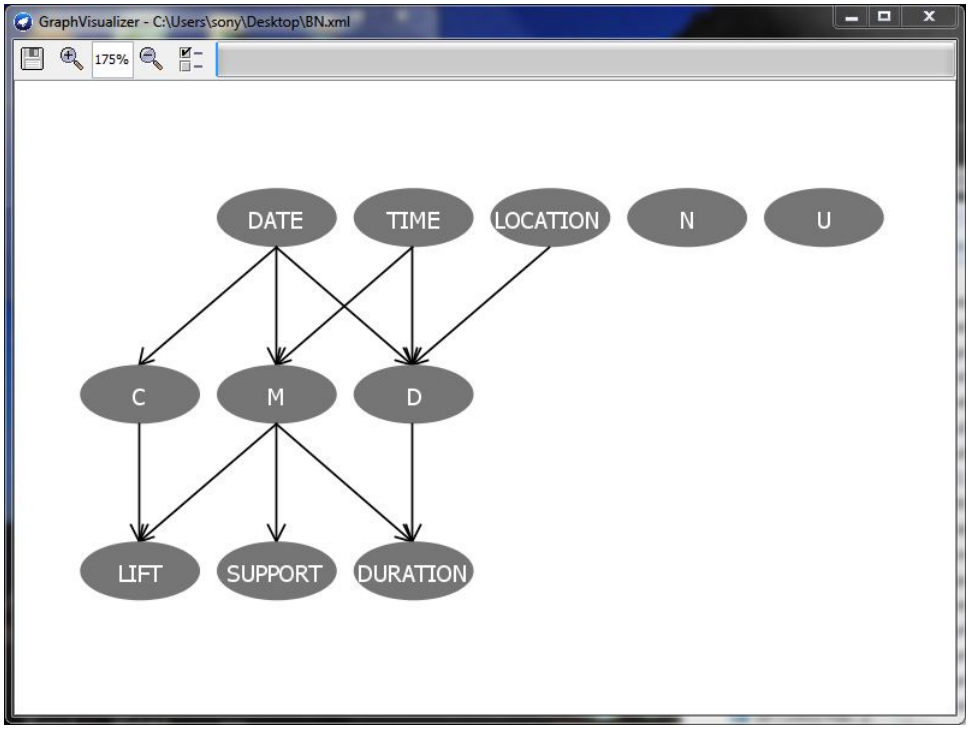


Figure 8.3: Behavior model of movie lists mining

given below exemplifies the possible circumstance tuples that can be derived from Table 8.2. The associated data mining quality tuple of each circumstance tuple is given on the same row under Q_I. Note that intervals such as $(0, 6]$ and rating labels such as minimum, maximum and so on are used instead of the exact values for legibility. For example, first tuple of C_I defines the circumstance where movie list mining is requested during morning hours of the festival period at a place close to one of the festival cinemas.

$C_I =$

$\{< date : (0, 15], time : (0, 6], location : (0, 4] >,$

$< date : (0, 15], time : (0, 6], location : (4, 7] >,$

$< date : (0, 15], time : (0, 6], location : (7, 10] >,$

$< date : (0, 15], time : (0, 6], location : (10, 13) >,$

$< date : (0, 15], time : (6, 9], location : (0, 4] >,$

$< date : (0, 15], time : (6, 9], location : (4, 7] >,$

$Q_I =$

$\{< support : highest, lift : lowest, duration : maximum >,$

$< support : highest, lift : lowest, duration : average >,$

$< support : highest, lift : lowest, duration : belowaverage >,$

$< support : highest, lift : lowest, duration : minimum >,$

$< support : highest, lift : average, duration : maximum >,$

$< support : highest, lift : average, duration : average >,$

$< date : (0, 15], time : (6, 9], location : (7, 10] >$, $< support : highest, lift : average, duration : belowaverage >$,

$< date : (0, 15], time : (6, 9], location : (10, 13) >$, $< support : highest, lift : average, duration : minimum >$,

....}

Parameter schema of Apriori is as follows:

$P(C : cdom,\ D : ddom,\ M : mdom)$ where

$cdom_i = \{c_1, c_2, ...\}$ $ddom_i = \{d_1, d_2, ...\}$ and $mdom_i = \{m_1, m_2, ...\}$

Then, the settings of the parameters C (minimum confidence) , D (delta) and M (lower bound minimum support) for the given circumstance (c^{tuple}), quality (q^{tuple}) are highest of the calculated φ_{C_k}, φ_{D_k} and φ_{M_k} respectively.

$\varphi_{C_k} = Probability(C = c_k \mid c^{tuple}, q^{tuple})\ where\ c_k \in cdom$

$\varphi_{D_k} = Probability(D = d_k \mid c^{tuple}, q^{tuple})\ where\ d_k \in ddom$

$\varphi_{M_k} = Probability(M = m_k \mid c^{tuple}, q^{tuple})\ where\ m_k \in mdom$

The conditional probabilities given above are calculated using the Bayesian network tool: JavaBayes ([22]). JavaBayes accepts Bayesian network representations in BIF (Bayesian Interchange Format) file format which is XML-based. BIF ([23]) is supported by Weka so that a Bayesian network constructed by Weka using an algorithm such as K2 can be saved as an XML BIF file.

In Appendix E, we publish the movie list mining configuration recommendations for all possible combinations of date, time and location given in Table 8.2.

Chapter 9

SUMMARY AND CONCLUSION

After Web 2.0, there has been an extreme increase in the number of people that actively contribute to the creation of Web content. Sharing information in the form of text, audio, video and image through social network/media sites that run Web 2.0 applications, became so indispensable habit of the great majority of the people in the world that it turned out to be the normal way of communication. As a matter of course, user-generated web content that grows enormously every moment, is processed for different purposes such as marketing, recommendation generation or personalization by using different methods. Data mining is among the preferred methods to discover knowledge from user-generated web content.

Other two technological developments in the last decade that have an huge impact on the habits of individuals were the dissemination of mobile phones succeeded by smartphones in affordable prices as well as the increase in the coverage and bandwidth of wireless networks. Consequently, the usage of mobile phones became pervasive and smartphone usage boosted in time resulting in substantial number of people being able to be online ubiquitously. So, nowadays it is quite customary to access the Web 2.0 applications, especially to the ones residing on the social sites via smartphones. comScore's report published in February 2012 ([18]) verifies the pervasive use of mobiles (234 million in the U.S.A) as well as the increasing trend in smartphone ownership (104 million in the U.S.A). Mobile content usage statistics in comScore's recently published

report also indicate a 3.1 percent increase (from 33 percent to 36.1 percent) on *social network or blog access* from the mobile devices within a three month period.

Since smartphones are trendy and are known to be preferred for social network/media access, it is possible and reasonable to download and mine relevant social network/media data on smartphones. Although it is possible to mine social data centrally on the servers, there can be several reasons and cases for favoring ubiquitous data mining to centralized approach. Among others, one of the reasons may be privacy whereas lacking of centralized computing power may be another.

We tackled the problem of automatically configuring an algorithm, in particular a data mining algorithm and we searched a solution to this problem for ubiquitous computing because not only autonomous behavior is essential for this dominant computing model of today but also data mining is indispensable for enriching ubiquitous computing applications with intelligence.

A number of challenges lie in the design of a general solution for ubiquitous computing. Since ubiquitous computing defines a broad range of applications and device types, configuration decisions should be dynamically given rather than applying a logic that is statically coded. Circumstantial factors are effective on ubiquitous computing and configuring an algorithm's execution by considering the circumstantial factors is important. Furthermore, assessing the success of the configuration decisions is essential.

In order to meet the challenges of the problem, we proposed an approach based on machine learning so that the behavior of the data mining algorithm in varying circumstances is modeled to be used for the configuration of the algorithm. By our approach, data mining quality that is realized is part of the behavior model so that whether the configuration quality goals are attained or not is assessed. Most importantly, adapting to the changing conditions by generating a new behavior model of data mining is possible whenever the existing behavior model lacks in attaining the configuration quality goals.

In summary;

- we specified the factors relevant to self-configuration of ubiquitous data mining.
- we have shown how self-configuration of ubiquitous data mining is possible by learning its behavior using Bayesian networks.
- we formally defined the classification of the executions of ubiquitous data mining by quality using decision trees so that data mining quality for self-configuration of data mining can be predicted.
- we performed experimental evaluations of the proposed self-configuration methods individually.
- we proposed a method to assess the efficiency of the behavior model.
- we have contemplated on possible application areas of data mining on mobile devices and social media and we elaborated on self-configuring social media data mining on mobile device.

Appendix A

K2: A Bayesian Method for Learning Structure of Bayesian Network from Data

A Bayesian method for constructing Bayesian network from data is explained in this section. The method is presented in the papers, [20] and [21]. We used the method in the first part of our study for constructing the behavior model of an algorithm's execution.

In the proposed method, the probabilistic dependencies among the domain variables from a database of cases are searched in order to form possible Bayesian network structure(s) whereas the most probable Bayesian network structure of data is determined consequently. In order to rank the possible Bayesian network structures so that the most probable one is found, the method computes $P(B_{S_i} \mid D)/P(B_{S_j} \mid D)$ where D is the database of cases whereas B_{S_i} and B_{S_j} are pairs of Bayesian network structures that can be extracted from D.

Most probable Bayesian network structure is learned from database of cases D by proposing:

- A formula for computing $P(B_{S_i}, D)$ given that

$$\frac{P(B_{S_i}|D)}{P(B_{S_j}|D)} = \frac{P(B_{S_i},D)}{P(B_{S_j},D)}$$

- A heuristic search procedure (K2) that attempts to find the B_S that maximizes (or nearly maximizes) $P(B_S \mid D)$ since exhaustive search procedure is exponential in the number of domain variables.

Solution is based on the following assumptions:

- Domain variables are discrete.
- Cases in the database are independent.
- There are no cases with missing values.
- Assignment of a value to the conditional probability $P(x \mid y)$ is independent of the assignment of a value to the conditional probability $P(x^{'} \mid y^{'})$ when two probabilities are components of different conditional probability distributions.
- The conditional distribution function $f(B_P \mid B_S)$ used in $P(B_S, D)$is uniform, where B_P is a vector whose values denote the conditional probability assignments associated with Bayesian network structure B_S.

Formula proposed in [20] and [21] for computing $P(B_S, D)$ (provided that assumptions listed above hold) is given in A.0.1:

$$P(B_S, D) = c \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{A.0.1}$$

Symbols used in the formula are as follows.

D is a database of m cases.Z is a set of n discrete variables, where a variable x_i in Z has r_i possible value assignments: $\{v_{i1}, ..., v_{ir_i}\}$. B_S denote the Bayesian network structure containing just the variables in Z. Each variable x_i in B_S has a set of parents π_i. Let $\Phi_i[j]$ denote the jth unique instantiation of π_i relative to D. q_i is the number of unique instantiations of π_i. N_{ijk} is the number of cases in D in which variable x_i is instantiated as v_{ik} and π_i is instantiated as $\Phi_i[j]$. N_{ij} is computed as: $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

Equation that maximizes $P(B_S, D)$ is denoted by:

$$\max_{B_S}[P(B_S, D)] = c \prod_{i=1}^{n} \max_{\pi_i}[\prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}!] \tag{A.0.2}$$

K2 is a heuristic search method for maximizing $P(B_S, D)$ with the assumption that there is an ordering on the domain variables. In K2, a greedy-search method is employed to find the parent set of each variable that maximizes the inner product in A.0.2. In particular, each node which is assumed to have no parents initially, is incrementally added parents so that each added parent increases the probability of the resulting structure most. The parent set of a node increases until no node can increase the probability of the resulting structure.

The time complexity of K2 is $O(mn^4r)$ where m is the number of cases in the database, n is the number of variables and r represent the number of possible assignments of variables. It has been assumed that factorials in A.0.2 are pre-computed and are stored in arrays.

Appendix B

Twitter: A Microblogging Site

Twitter([43]) is a microblogging service that allows users to share 140 character status updates ("tweets"). A "tweet" may contain picture, link and video as well as the text message. Millions of "tweets" are posted per day through Twitter. User determines what is interesting for him so that the messages from the users he chooses to follow ("followee") appear on his home page in "timeline" (in real time order they are posted). User can allow his "tweets" to be available publicly or protect his "tweets" and let only the confirmed users to be his "followers". Users of Twitter are not only individuals but also the organizations that prefer Twitter as a platform to reach customer base, advertise their products and so on. It is possible to download publicly available "tweets" resulting in abundance of third-party applications that either mine "tweets" or mash up with data from a distinct source.

As of today Twitter is not only the most popular microblogging service having a huge number of users but also it is the media where people like to share their status all the time due to its practical use. It is also possible and customary to use Twitter with mobile devices (smartphones and tablet PC's) so that users can interact with the application everywhere and anytime. Twitter also offers access or download to its corpus of data through API's to be used by the applications.

Appendix C

Data Mining Model for Movie Recommendations

An example data mining model of movie list mining produced by Weka Explorer.

=== Run information ===

Scheme: weka.associations.Apriori -N 50 -T 1 -C 1.1 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation: movies_itemset_clean
Instances: 942
Attributes: 194
[list of attributes omitted]
=== Associator model (full training set) ===

Apriori
=======

Minimum support: 0.2 (188 instances)
Minimum metric <lift>: 1.1
Number of cycles performed: 16

Generated sets of large itemsets:

Size of set of large itemsets L(1): 51

Size of set of large itemsets L(2): 37

Size of set of large itemsets L(3): 5

Best rules found:

1. I172=yes 293 ==> I174=yes I181=yes 191 conf:(0.65) < lift:(2.66)> lev:(0.13) [119] conv:(2.15)
2. I174=yes I181=yes 231 ==> I172=yes 191 conf:(0.83) < lift:(2.66)> lev:(0.13) [119] conv:(3.88)
3. I172=yes 293 ==> I50=yes I174=yes 229 conf:(0.78) < lift:(2.51)> lev:(0.15) [137] conv:(3.11)
4. I50=yes I174=yes 293 ==> I172=yes 229 conf:(0.78) < lift:(2.51)> lev:(0.15) [137] conv:(3.11)
5. I174=yes 348 ==> I195=yes 191 conf:(0.55) < lift:(2.38)> lev:(0.12) [110] conv:(1.7)
6. I195=yes 217 ==> I174=yes 191 conf:(0.88) < lift:(2.38)> lev:(0.12) [110] conv:(5.07)
7. I174=yes 348 ==> I172=yes I181=yes 191 conf:(0.55) < lift:(2.29)> lev:(0.11) [107] conv:(1.67)
8. I172=yes I181=yes 226 ==> I174=yes 191 conf:(0.85) < lift:(2.29)> lev:(0.11) [107] conv:(3.96)
9. I50=yes I172=yes 274 ==> I174=yes 229 conf:(0.84) < lift:(2.26)> lev:(0.14) [127] conv:(3.76)
10. I174=yes 348 ==> I50=yes I172=yes 229 conf:(0.66) < lift:(2.26)> lev:(0.14) [127] conv:(2.06)
11. I174=yes 348 ==> I210=yes 196 conf:(0.56) < lift:(2.26)> lev:(0.12) [109] conv:(1.71)
12. I210=yes 235 ==> I174=yes 196 conf:(0.83) < lift:(2.26)> lev:(0.12) [109] conv:(3.7)
13. I172=yes 293 ==> I174=yes 240 conf:(0.82) < lift:(2.22)> lev:(0.14) [131] conv:(3.42)
14. I174=yes 348 ==> I172=yes 240 conf:(0.69) < lift:(2.22)> lev:(0.14) [131] conv:(2.2)
15. I174=yes 348 ==> I204=yes 189 conf:(0.54) < lift:(2.18)> lev:(0.11) [102] conv:(1.63)
16. I204=yes 235 ==> I174=yes 189 conf:(0.8) < lift:(2.18)> lev:(0.11) [102] conv:(3.15)
17. I50=yes I98=yes 256 ==> I174=yes 201 conf:(0.79) < lift:(2.13)> lev:(0.11) [106] conv:(2.88)
18. I174=yes 348 ==> I50=yes I98=yes 201 conf:(0.58) < lift:(2.13)> lev:(0.11) [106] conv:(1.71)
19. I79=yes 264 ==> I174=yes 206 conf:(0.78) < lift:(2.11)> lev:(0.12) [108] conv:(2.82)
20. I174=yes 348 ==> I79=yes 206 conf:(0.59) < lift:(2.11)> lev:(0.12) [108] conv:(1.75)
21. I173=yes 248 ==> I174=yes 189 conf:(0.76) < lift:(2.06)> lev:(0.1) [97] conv:(2.61)
22. I174=yes 348 ==> I173=yes 189 conf:(0.54) < lift:(2.06)> lev:(0.1) [97] conv:(1.6)
23. I172=yes 293 ==> I50=yes I181=yes 220 conf:(0.75) < lift:(2.03)> lev:(0.12) [111] conv:(2.49)
24. I50=yes I181=yes 349 ==> I172=yes 220 conf:(0.63) < lift:(2.03)> lev:(0.12) [111] conv:(1.85)
25. I50=yes I172=yes 274 ==> I181=yes 220 conf:(0.8) < lift:(2)> lev:(0.12) [109] conv:(2.98)

26. I181=yes 379 ==> I50=yes I172=yes 220 conf:(0.58) < lift:(2)> lev:(0.12) [109] conv:(1.68)
27. I172=yes I174=yes 240 ==> I181=yes 191 conf:(0.8) < lift:(1.98)> lev:(0.1) [94] conv:(2.87)
28. I181=yes 379 ==> I172=yes I174=yes 191 conf:(0.5) < lift:(1.98)> lev:(0.1) [94] conv:(1.49)
29. I56=yes 294 ==> I98=yes 207 conf:(0.7) < lift:(1.93)> lev:(0.11) [99] conv:(2.12)
30. I98=yes 344 ==> I56=yes 207 conf:(0.6) < lift:(1.93)> lev:(0.11) [99] conv:(1.71)
31. I172=yes 293 ==> I181=yes 226 conf:(0.77) < lift:(1.92)> lev:(0.11) [108] conv:(2.58)
32. I181=yes 379 ==> I172=yes 226 conf:(0.6) < lift:(1.92)> lev:(0.11) [108] conv:(1.7)
33. I98=yes 344 ==> I50=yes I174=yes 201 conf:(0.58) < lift:(1.88)> lev:(0.1) [94] conv:(1.65)
34. I50=yes I174=yes 293 ==> I98=yes 201 conf:(0.69) < lift:(1.88)> lev:(0.1) [94] conv:(2)
35. I56=yes 294 ==> I174=yes 203 conf:(0.69) < lift:(1.87)> lev:(0.1) [94] conv:(2.02)
36. I174=yes 348 ==> I56=yes 203 conf:(0.58) < lift:(1.87)> lev:(0.1) [94] conv:(1.64)
37. I50=yes I174=yes 293 ==> I181=yes 219 conf:(0.75) < lift:(1.86)> lev:(0.11) [101] conv:(2.33)
38. I181=yes 379 ==> I50=yes I174=yes 219 conf:(0.58) < lift:(1.86)> lev:(0.11) [101] conv:(1.62)
39. I98=yes 344 ==> I172=yes 198 conf:(0.58) < lift:(1.85)> lev:(0.1) [91] conv:(1.61)
40. I172=yes 293 ==> I98=yes 198 conf:(0.68) < lift:(1.85)> lev:(0.1) [91] conv:(1.94)
41. I50=yes 501 ==> I172=yes I181=yes 220 conf:(0.44) < lift:(1.83)> lev:(0.11) [99] conv:(1.35)
42. I172=yes I181=yes 226 ==> I50=yes 220 conf:(0.97) < lift:(1.83)> lev:(0.11) [99] conv:(15.11)
43. I98=yes 344 ==> I174=yes 231 conf:(0.67) < lift:(1.82)> lev:(0.11) [103] conv:(1.9)
44. I174=yes 348 ==> I98=yes 231 conf:(0.66) < lift:(1.82)> lev:(0.11) [103] conv:(1.87)
45. I50=yes 501 ==> I172=yes I174=yes 229 conf:(0.46) < lift:(1.79)> lev:(0.11) [101] conv:(1.37)
46. I172=yes I174=yes 240 ==> I50=yes 229 conf:(0.95) < lift:(1.79)> lev:(0.11) [101] conv:(9.36)
47. I50=yes 501 ==> I174=yes I181=yes 219 conf:(0.44) < lift:(1.78)> lev:(0.1) [96] conv:(1.34)
48. I174=yes I181=yes 231 ==> I50=yes 219 conf:(0.95) < lift:(1.78)> lev:(0.1) [96] conv:(8.32)
49. I50=yes 501 ==> I172=yes 274 conf:(0.55) < lift:(1.76)> lev:(0.13) [118] conv:(1.51)
50. I172=yes 293 ==> I50=yes 274 conf:(0.94) < lift:(1.76)> lev:(0.13) [118] conv:(6.86)

Appendix D

Training Data for Behavior Model Construction

Execution data of movie list mining collected through EDG consisting of the attributes:

- AVGMEM: Average memory usage in bytes.
- CPUTIME: Total CPU time in msecs.
- DURATION: Duration in msecs.
- N: Requested number of association rules.
- C: Minimum confidence requested.
- D: Delta for minimum support.
- U: Upper bound minimum support.
- M: Lower bound minimum support.
- SUPPORT: Minimum support of the data mining model.
- CONF: Minimum confidence of the data mining model.
- LIFT: Minimum lift of the data mining model.
- RULES: Number of rules in the data mining model.

AVGMEM	CPUTIME	DURATION	N	C	D	U	M	SUPPORT	CONF	LIFT	RULES
476800000	165190	165305	50	1	0.1	0.5	0.12	0.1	1	1.56	50
477543424	230200	230919	50	1	0.05	0.5	0.12	0.1	1	1.56	50
477486080	363450	363368	50	1	0.02	0.5	0.1	0.12	1	1.56	50
476205056	29300	29539	50	0.9	0.1	0.5	0.1	0.2	0.95	1.42	50
476235776	29470	29471	50	0.9	0.1	0.5	0.2	0.3	0.95	1.42	50
475051008	11190	11204	50	0.9	0.1	0.5	0.3	0.3	0.92	1.37	5
476123136	6510	6543	50	0.9	0.1	0.5	0.4	0.4	0.92	1.35	2
476214272	41690	41791	50	0.9	0.05	0.5	0.1	0.2	0.95	1.42	50
477288448	41370	41412	50	0.9	0.05	0.5	0.2	0.2	0.95	1.42	50
477177856	13330	13359	50	0.9	0.05	0.5	0.3	0.3	0.92	1.37	5
476118016	9500	9563	50	0.9	0.05	0.5	0.4	0.4	0.92	1.35	2
476163072	62240	62148	50	0.9	0.02	0.5	0.1	0.22	0.91	1.47	50
476168192	62410	62251	50	0.9	0.02	0.5	0.2	0.22	0.91	1.47	50
476891136	28890	29049	50	0.9	0.02	0.5	0.3	0.3	0.92	1.37	5
475256832	20650	21210	50	0.9	0.02	0.5	0.4	0.4	0.92	1.35	2
477320192	29250	29293	50	0.8	0.1	0.5	0.1	0.2	0.95	1.42	50
477314048	29470	29607	50	0.8	0.1	0.5	0.2	0.3	0.95	1.42	50
475063296	10780	11390	50	0.8	0.1	0.5	0.3	0.3	0.8	1.36	14
475066368	6450	6492	50	0.8	0.1	0.5	0.4	0.4	0.85	1.3	3
476136448	20800	20956	50	0.8	0.05	0.5	0.1	0.25	0.82	1.46	50
476137472	20730	20630	50	0.8	0.05	0.5	0.2	0.25	0.82	1.46	50
477206528	13320	13514	50	0.8	0.05	0.5	0.3	0.3	0.8	1.36	14
483871744	9620	9486	50	0.8	0.05	0.5	0.4	0.4	0.85	1.3	3
476151808	47860	47739	50	0.8	0.02	0.5	0.1	0.24	0.84	1.47	50
477078528	48280	48425	50	0.8	0.02	0.5	0.2	0.24	0.84	1.47	50
476141568	28890	28771	50	0.8	0.02	0.5	0.3	0.3	0.8	1.36	14
477200384	20770	20544	50	0.8	0.02	0.5	0.4	0.4	0.85	1.3	3
476223488	29660	29876	50	0.7	0.1	0.5	0.1	0.2	0.95	1.42	50
476243968	30010	30107	50	0.7	0.1	0.5	0.2	0.3	0.95	1.42	50
476158976	11240	11108	50	0.7	0.1	0.5	0.3	0.3	0.7	1.4	26
476148736	6510	6381	50	0.7	0.1	0.5	0.4	0.4	0.85	1.3	3
476154880	20800	20772	50	0.7	0.05	0.5	0.1	0.25	0.82	1.46	50
475086848	21200	21468	50	0.7	0.05	0.5	0.2	0.25	0.82	1.46	50
476150784	13270	13171	50	0.7	0.05	0.5	0.3	0.3	0.7	1.4	26
476146688	9510	9504	50	0.7	0.05	0.5	0.4	0.4	0.85	1.3	3
476158976	38450	38625	50	0.7	0.02	0.5	0.1	0.26	0.79	1.46	50
476274688	38690	38482	50	0.7	0.02	0.5	0.2	0.26	0.79	1.46	50
477205504	28860	29037	50	0.7	0.02	0.5	0.3	0.3	0.7	1.4	26
476139520	24400	24444	50	0.7	0.02	0.5	0.4	0.4	0.85	1.3	3
477339648	29910	29915	50	0.6	0.1	0.5	0.1	0.2	0.95	1.42	50
476230656	29270	29267	50	0.6	0.1	0.5	0.2	0.3	0.95	1.42	50
477220864	12550	12462	50	0.6	0.1	0.5	0.3	0.3	0.6	1.38	45
477233152	6550	6484	50	0.6	0.1	0.5	0.4	0.4	0.62	1.29	5
476172288	20760	20648	50	0.6	0.05	0.5	0.1	0.25	0.82	1.46	50
476158976	20630	20647	50	0.6	0.05	0.5	0.2	0.25	0.82	1.46	50
476153856	13260	13324	50	0.6	0.05	0.5	0.3	0.3	0.6	1.38	45
476169216	9850	9914	50	0.6	0.05	0.5	0.4	0.4	0.62	1.29	5
476167168	32010	32052	50	0.6	0.02	0.5	0.1	0.28	0.67	1.4	50
476154880	32270	32806	50	0.6	0.02	0.5	0.2	0.28	0.67	1.4	50
476155904	28800	28808	50	0.6	0.02	0.5	0.3	0.3	0.6	1.38	45
476166144	20990	20936	50	0.6	0.02	0.5	0.4	0.4	0.62	1.29	5

Figure D.1: Subset of data collected by EDG

Appendix E

Configuration Recommendations for Movie List Mining

Each line of the list given below shows a circumstance tuple followed by the recommended configuration tuple for that circumstance.
Circumstance tuples consist of the attributes: date, time and location. For convenience, ranges of values are used instead of exact values since the recommended configurations are valid for all the circumstance states within the given range.
Configuration tuples consist of the attributes: C (minimum confidence requested), M (lower bound minimum support), D (delta for minimum support).

CIRCUMSTANCE	CONFIGURATION
< date:(0,15], time:(0,6], location:(0,4] >	< C:0.8, M:0.3, D:0.05 >
< date:(0,15], time:(0,6], location:(4,7] >	< C:0.8, M:0.4, D:0.02 >
< date:(0,15], time:(0,6], location:(7,10] >	< C:0.8, M:0.4, D:0.02 >
< date:(0,15], time:(0,6], location:(10,13) >	< C:0.8, M:0.4, D:0.05 >
< date:(0,15], time:(6,9], location:(0,4] >	< C:0.7, M:0.3, D:0.1 >
< date:(0,15], time:(6,9], location:(4,7] >	< C:0.7, M:0.3, D:0.02 >
< date:(0,15], time:(6,9], location:(7,10] >	< C:0.7, M:0.3, D:0.05 >
< date:(0,15], time:(6,9], location:(10,13) >	< C:0.7, M:0.3, D:0.1 >
< date:(0,15], time:(9,12], location:(0,4] >	< C:0.8, M:0.2, D:0.05 >
< date:(0,15], time:(9,12], location:(4,7] >	< C:0.8, M:0.3, D:0.02 >
< date:(0,15], time:(9,12], location:(7,10] >	< C:0.8, M:0.3, D:0.05 >
< date:(0,15], time:(9,12], location:(10,13) >	< C:0.8, M:0.3, D:0.05 >
< date:(0,15], time:(18,24], location:(0,4] >	< C:0.9, M:0.3, D:0.05 >
< date:(0,15], time:(18,24], location:(4,7] >	< C:0.9, M:0.3, D:0.02 >
< date:(0,15], time:(18,24], location:(7,10] >	< C:0.9, M:0.3, D:0.05 >
< date:(0,15], time:(18,24], location:(10,13) >	< C:0.9, M:0.3, D:0.1 >
< date:(15,21], time:(0,6], location:(0,4] >	< C:0.7, M:0.2, D:0.02 >
< date:(15,21], time:(0,6], location:(4,7] >	< C:0.7, M:0.2, D:0.02 >
< date:(15,21], time:(0,6], location:(7,10] >	< C:0.7, M:0.4, D:0.05 >
< date:(15,21], time:(0,6], location:(10,13) >	< C:0.7, M:0.4, D:0.05 >
< date:(15,21], time:(6,9], location:(0,4] >	< C:0.7, M:0.2, D:0.05 >
< date:(15,21], time:(6,9], location:(4,7] >	< C:0.7, M:0.2, D:0.05 >
< date:(15,21], time:(6,9], location:(7,10] >	< C:0.7, M:0.3, D:0.05 >
< date:(15,21], time:(6,9], location:(10,13) >	< C:0.7, M:0.2, D:0.05 >
< date:(15,21], time:(9,12], location:(0,4] >	< C:0.8, M:0.2, D:0.02 >
< date:(15,21], time:(9,12], location:(4,7] >	< C:0.8, M:0.2, D:0.05 >
< date:(15,21], time:(9,12], location:(7,10] >	< C:0.8, M:0.2, D:0.05 >
< date:(15,21], time:(9,12], location:(10,13) >	< C:0.8, M:0.2, D:0.05 >
< date:(15,21], time:(18,24], location:(0,4] >	< C:0.7, M:0.2, D:0.02 >
< date:(15,21], time:(18,24], location:(4,7] >	< C:0.7, M:0.2, D:0.02 >
< date:(15,21], time:(18,24], location:(7,10] >	< C:0.7, M:0.3, D:0.05 >
< date:(15,21], time:(18,24], location:(10,13) >	< C:0.7, M:0.2, D:0.05 >

Figure E.1: Configuration recommendations under possible circumstances

CIRCUMSTANCE	CONFIGURATION
< date:(21,31], time:(0,6], location:(0,4] >	< C:0.9, M:0.2, D:0.02 >
< date:(21,31], time:(0,6], location:(4,7] >	< C:0.8, M:0.4, D:0.02 >
< date:(21,31], time:(0,6], location:(7,10] >	< C:0.8, M:0.3, D:0.05 >
< date:(21,31], time:(0,6], location:(10,13) >	< C:0.8, M:0.4, D:0.05 >
< date:(21,31], time:(6,9], location:(0,4] >	< C:0.9, M:0.1, D:0.05 >
< date:(21,31], time:(6,9], location:(4,7] >	< C:0.9, M:0.2, D:0.1 >
< date:(21,31], time:(6,9], location:(7,10] >	< C:0.9, M:0.1, D:0.05 >
< date:(21,31], time:(6,9], location:(10,13) >	< C:0.9, M:0.1, D:0.05 >
< date:(21,31], time:(9,12], location:(0,4] >	< C:0.9, M:0.2, D:0.02 >
< date:(21,31], time:(9,12], location:(4,7] >	< C:0.9, M:0.2, D:0.05 >
< date:(21,31], time:(9,12], location:(7,10] >	< C:0.9, M:0.2, D:0.05 >
< date:(21,31], time:(9,12], location:(10,13) >	< C:0.9, M:0.2, D:0.05 >
< date:(21,31], time:(18,24], location:(0,4] >	< C:0.9, M:0.2, D:0.02 >
< date:(21,31], time:(18,24], location:(4,7] >	< C:0.9, M:0.3, D:0.02 >
< date:(21,31], time:(18,24], location:(7,10] >	< C:0.9, M:0.3, D:0.05 >
< date:(21,31], time:(18,24], location:(10,13) >	< C:0.9, M:0.4, D:0.05 >
< date:(31,40), time:(0,6], location:(0,4] >	< C:1, M:0.1, D:0.02 >
< date:(31,40), time:(0,6], location:(4,7] >	< C:1, M:0.1, D:0.02 >
< date:(31,40), time:(0,6], location:(7,10] >	< C:1, M:0.1, D:0.05 >
< date:(31,40), time:(0,6], location:(10,13) >	< C:1, M:0.4, D:0.05 >
< date:(31,40), time:(6,9], location:(0,4] >	< C:1, M:0.1, D:0.02 >
< date:(31,40), time:(6,9], location:(4,7] >	< C:1, M:0.1, D:0.02 >
< date:(31,40), time:(6,9], location:(7,10] >	< C:1, M:0.3, D:0.05 >
< date:(31,40), time:(6,9], location:(10,13) >	< C:1, M:0.1, D:0.05 >
< date:(31,40), time:(9,12], location:(0,4] >	< C:1, M:0.1, D:0.02 >
< date:(31,40), time:(9,12], location:(4,7] >	< C:1, M:0.1, D:0.02 >
< date:(31,40), time:(9,12], location:(7,10] >	< C:1, M:0.1, D:0.05 >
< date:(31,40), time:(9,12], location:(10,13) >	< C:1, M:0.1, D:0.05 >
< date:(31,40), time:(18,24], location:(0,4] >	< C:1, M:0.1, D:0.02 >
< date:(31,40), time:(18,24], location:(4,7] >	< C:1, M:0.1, D:0.02 >
< date:(31,40), time:(18,24], location:(7,10] >	< C:1, M:0.3, D:0.05 >
< date:(31,40), time:(18,24], location:(10,13) >	< C:1, M:0.1, D:0.05 >

Figure E.2: Configuration recommendations under possible circumstances (cont.)

Bibliography

[1] Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Oper. Res.*, 54(1):99–114, January 2006.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] Steven C. Amstrup, Hal Caswell, Eric DeWeaver, Ian Stirling, David C. Douglas, Bruce G. Marcot, and Christine M. Hunter. Rebuttal of "polar bear population forecasts: A public-policy forecasting audit". *Interfaces*, 39(4):353–369, July 2009.

[4] I. A. Beinlich, Henri Jacques Suermondt, R. Martin Chavez, and Gregory F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine*, pages 247–256. Springer-Verlag, 1989.

[5] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[6] Leo Breiman, Jerome Friedman, R. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[7] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 255–264, Tucson, Arizona, USA, May 1997.

[8] Wray L. Buntine. A guide to the literature on learning probabilistic networks from data. *Knowledge and Data Engineering, IEEE Transactions on*, 8(2):195 –210, apr 1996.

[9] Longbing Cao, Vladimir Gorodetsky, and Pericles A. Mitkas. Agent mining: The synergy of agents and data mining. *Intelligent Systems, IEEE*, 24(3):64 –72, May-June 2009.

[10] Aysegul Cayci, Santiago Eibe, Ernestina Menasalvas, and Yücel Saygın. Bayesian networks to predict data mining algorithm behavior in ubiquitous computing environments. In Martin Atzmüller, Andreas Hotho, Markus Strohmaier, and Alvin Chin, editors, *MSM/MUSE*, volume 6904 of *Lecture Notes in Computer Science*, pages 119–141. Springer, 2010.

[11] Aysegul Cayci, João Bartolo Gomes, Andrea Zanda, Ernestina Menasalvas, and Santiago Eibe. Situation-aware data mining service for ubiquitous environments. In *Proceedings of the 2009 Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, UBICOMM '09, pages 135–140, Washington, DC, USA, 2009. IEEE Computer Society.

[12] Aysegul Cayci, João Bartolo Gomes, Andrea Zanda, Ernestina Menasalvas, and Santiago Eibe. Research challenge of locally computed ubiquitous data mining. In Maria Manuela Cruz-Cunha and Fernando Moreira, editors, *Handbook of Research on Mobility and Computing: Evolving Technologies and Ubiquitous Impacts*, pages 576–594. IGI Global, 2011.

[13] Aysegul Cayci, Ernestina Menasalvas, Yücel Saygın, and Santiago Eibe. Self-configuring data mining for ubiquitous computing. *Inf. Sci.*, Submitted for review, 2012.

[14] Aysegul Cayci, Yücel Saygın, and Ernestina Menasalvas. Twitter-based recommendation system through ubiquitous data mining. *Expert Syst. Appl.*, Submitted for review, 2013.

[15] Joong Hyuk Chang and Won Suk Lee. Finding frequent itemsets over online data streams. *Information & Software Technology*, 48(7):606–618, 2006.

[16] Ching-Ming Chao and Guan-Lin Chao. Resource-aware high quality clustering in ubiquitous data streams. In Runtong Zhang, José Cordeiro, Xuewei Li, Zhenji Zhang, and Juliang Zhang, editors, *ICEIS (1)*, pages 64–73. SciTePress, 2011.

[17] Eugene Charniak and Robert Goldman. A semantics for probabilistic quantifier-free first-order languages, with particular application to story understanding. In *Proceedings of the 11th international joint conference on Artificial intelligence - Volume 2*, IJCAI'89, pages 1074–1079, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[18] comScore. "`http://www.comscore.com/Press_Events/Press_Releases/2012/4/comScore_Reports_February_2012_U.S._Mobile_Subscriber_Market_Share`".

[19] Sunny Consolvo, David W. McDonald, Tammy Toscos, Mike Y. Chen, Jon Froehlich, Beverly Harrison, Predrag Klasnja, Anthony LaMarca, Louis LeGrand, Ryan Libby, Ian Smith, and James A. Landay. Activity sensing in the wild: a field trial of ubifit garden. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1797–1806, New York, NY, USA, 2008. ACM.

[20] Gregory F. Cooper and Edward Herskovits. A bayesian method for constructing bayesian belief networks from databases. In Bruce D'Ambrosio and Philippe Smets, editors, *UAI*, pages 86–94. Morgan Kaufmann, 1991.

[21] Gregory F. Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Mach. Learn.*, 9(4):309–347, October 1992.

[22] Fabio Gagliardi Cozman. "`http://www.cs.cmu.edu/~javabayes/`".

[23] Fabio Gagliardi Cozman. "`http://www.cs.cmu.edu/~fgcozman/Research/InterchangeFormat/`".

[24] Android Developers. "`http://developer.android.com/`".

[25] The Eclipse Foundation. "`http://www.eclipse.org/`".

[26] Conny Franke, Marcel Karnstedt, and Kai-Uwe Sattler. Mining data streams under dynamicly changing resource constraints. In Klaus-Dieter Althoff and Martin Schaaf, editors, *LWA*, volume 1/2006 of *Hildesheimer Informatik-Berichte*, pages 262–269. University of Hildesheim, Institute of Computer Science, 2006.

[27] Mohamed Medhat Gaber, Shonali Krishnaswamy, and Arkady Zaslavsky. Adaptive mining techniques for data streams using algorithm output granularity. In *Workshop (AusDM 2003), Held in conjunction with the 2003 Congress on Evolutionary Computation (CEC 2003)*. Springer Verlag, 2003.

[28] Mohamed Medhat Gaber, Shonali Krishnaswamy, and Arkady B. Zaslavsky. Resource-aware mining of data streams. *J. UCS*, 11(8):1440–1453, 2005.

[29] Mohamed Medhat Gaber, Shonali Krishnaswamy, and Arkady B. Zaslavsky. Resource-aware mining of data streams. *J. UCS*, 11(8):1440–1453, 2005.

[30] Mohamed Medhat Gaber and Philip S. Yu. A framework for resource-aware knowledge discovery in data streams: a holistic approach with its application to clustering. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 649–656, New York, NY, USA, 2006. ACM.

[31] Matteo Gagliolo and Jürgen Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, August 2006.

[32] Hamed Haddadi, Pan Hui, and Ian Brown. Mobiad: private and scalable mobile advertising. In *Proceedings of the fifth ACM international workshop on Mobility in the evolving internet architecture*, MobiArch '10, pages 33–38, New York, NY, USA, 2010. ACM.

[33] Pari Delir Haghighi, Arkady Zaslavsky, Shonali Krishnaswamy, Mohamed Medhat Gaber, and Seng Loke. Context-aware adaptive data stream mining. *Intell. Data Anal.*, 13(3):423–434, August 2009.

[34] Pari Delir Haghighi, Arkady B. Zaslavsky, Shonali Krishnaswamy, and Mohamed Medhat Gaber. Mobile data mining for intelligent healthcare support. In *HICSS*, pages 1–10. IEEE Computer Society, 2009.

[35] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[36] Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[37] David Heckerman. A tutorial on learning with bayesian networks. In Michael I. Jordan, editor, *Learning in graphical models*, pages 301–354. MIT Press, Cambridge, MA, USA, 1999.

[38] Cynthia S. Hood and Chuanyi Ji. Proactive network fault detection. *IEEE Computer and Communications Societies, Annual Joint Conference of the*, 0:1147, 1997.

[39] Earl B. Hunt, Philip J. Stone, and Janet Marin. *Experiments in induction / Earl B. Hunt, Janet Marin, Philip J. Stone*. Academic Press, New York :, 1966.

[40] Frank Hutter and Youssef Hamadi. Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Technical report, In: Technical Report: MSR-TR-2005125, Microsoft Research, 2005.

[41] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, pages 1152–1157. AAAI Press, 2007.

[42] Minitab Inc. "`http://www.minitab.com/us-EN/`".

[43] Twitter Inc. "`http://www.twitter.com/`".

[44] Conny Junghans, Marcel Karnstedt, and Michael Gertz. Quality-driven resource-adaptive data stream mining. *SIGKDD Explorations*, 13(1):72–82, 2011.

[45] Marcel Karnstedt, Conny Franke, and Mohamed Medhat Gaber. A model for quality guaranteed resource-aware stream mining. In *Fifth International Workshop on Knowledge Discovery from Ubiquitous Data Streams icw ECML/PKDD'07*, pages 72–82, 2007.

[46] Lin Liao, Donald J. Patterson, Dieter Fox, and Henry Kautz. Learning and inferring transportation routines. *Artif. Intell.*, 171(5-6):311–331, April 2007.

[47] Ingo Mierswa, Katharina Morik, and Michael Wurst. Collaborative use of features in a distributed system for the organization of music collections. In Shephard Shen and Liu Cui, editors, *Intelligent Music Information Systems: Tools and Methodologies*, pages 147–176. Idea Group Publishing, 2007.

[48] Douglas C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 2006.

[49] Katharina Morik. Nemoz: a distributed framework for collaborative media organization. In Michael May and Lorenza Saitta, editors, *Ubiquitous knowledge discovery*, pages 199–215. Springer-Verlag, Berlin, Heidelberg, 2010.

[50] Movielens. "`http://movielens.umn.edu/`".

[51] Reyes Pavón, Fernando Díaz, Rosalía Laza, and Victoria Luzón. Automatic parameter tuning with a bayesian case-based reasoning system. a case of study. *Expert Syst. Appl.*, 36(2):3407–3420, March 2009.

[52] Reyes Pavón, Fernando Díaz, and Victoria Luzón. A model for parameter setting based on bayesian networks. *Eng. Appl. Artif. Intell.*, 21(1):14–25, February 2008.

[53] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[54] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W.J. Frawley, editors, *Knowledge Discovery in Databases*. AAAI/MIT Press, Cambridge, MA, 1991.

[55] GroupLens Research Project. "`http://www.grouplens.org/`".

[56] J. Ross Quinlan. Induction of decision trees. *Mach. Learn*, pages 81–106, 1986.

[57] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[58] Flora Dilys Salim, Shonali Krishnaswamy, Seng Wai Loke, and Andry Rakotonirainy. Context-aware ubiquitous data mining based agent model for intersection safety. In Tomoya Enokido, Lu Yan, Bin Xiao, Daeyoung Kim, Yuan-Shun Dai, and Laurence Tianruo Yang, editors, *EUC Workshops*, volume 3823 of *Lecture Notes in Computer Science*, pages 61–70. Springer, 2005.

[59] Biplav Srivastava and Anupam Mediratta. Domain-dependent parameter selection of search-based algorithms compatible with user performance criteria. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 3*, AAAI'05, pages 1386–1391. AAAI Press, 2005.

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

[61] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[62] Android Developers Youtube. "http://www.youtube.com/user/androiddevelopers/".