Self-configuring data mining for ubiquitous computing

Aysegul Cayci[a,*], Ernestina Menasalvas[b], Yucel Saygin[a], Santiago Eibe[b]

[a]*Faculty of Engineering and Natural Sciences, Sabanci University, Orhanli, Tuzla, 34956 Istanbul, Turkey*
[b]*Facultad de Informatica, Universidad Politecnica de Madrid, Madrid, Spain*

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 since algorithm executes in an environment in which the context often changes and computing resources are often severely limited. We propose to analyze the execution behavior of the data mining algorithm by mining its past executions. By doing so, we discover the effects of resource and context states as well as parameter settings on the data mining quality. We argue that a classification model is appropriate for predicting the behavior of an algorithm's execution and we concentrate on decision tree classifier. We also define taxonomy 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. Behavior model constituents and class label transformations are formally defined and experimental validation of the proposed approach is also performed.

Keywords: Data mining, ubiquitous computing, decision trees.

*Corresponding author.

Email addresses: `aysegulcayci@su.sabanciuniv.edu` (Aysegul Cayci), `emanasalvas@fi.upm.es` (Ernestina Menasalvas), `ysaygin@sabanciuniv.edu` (Yucel Saygin), `seibe@fi.upm.es` (Santiago Eibe)

1. Introduction

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.

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. In this paper:

- 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 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 to propose a general-purpose solution for configuring ubiquitous data mining and we impose no restrictions on the types of the algorithm parameters that we configure. 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.
- We propose to use taxonomy of quality in order to find the most appropriate behavior model which is balanced in terms of accuracy and specificity of classification.

The rest of the paper is organized as follows: Section 2, is related work. In Section 3 we formulate the problem and present the proposed approach whereas Section 4 elaborates on solution by decision tree. In Section 5, we explain the experiment that we performed in order to validate of the proposed approach and finally, Section 6 presents the conclusion.

2. Related Work

Ubiquitous data mining has several application areas today. Examples include health-care, transportation, assisted living and commerce to name a few. We do not focus on a specific application area but on the contrary we contemplate on how to mine data on a ubiquitous computing environment in general. Our perspective on ubiquitous data mining is as follows:

1. Resource-awareness: awareness of the resources that will be demanded by data mining, knowledge on how to measure the availability of the resources and how to optimize the use of the resources.
2. Context-awareness: exploiting the variability of the context to achieve better mining results.

3. Autonomous behavior: taking the decisions related to self execution independently.

4. Adaptability: adapting the decisions to the changing conditions.

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 resource-aware data mining presented in [8] is for data streams where output granularity is adapted to the data rate of the stream, available memory and time constraints. In a later study ([9]), the idea of adapting output granularity is defined within a generic framework for resource aware stream mining where input granularity and processing settings of the algorithm are also adapted in a resource aware manner. A quality aware data stream mining specific to frequent itemset mining algorithm in [7] is able to adapt according to output quality as well as the resource consumption patterns. At a recent work, a general model of resource and quality aware data stream mining is proposed in [13] 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. In [14], a frequent itemset stream mining algorithm is presented that utilizes an adaptive memory scheme to maximize the mining accuracy for confined memory space. In [21], k-means algorithm is proposed for data stream clustering that is able to adapt to variations in memory availability.

Another line of research focuses on adjusting parameters of stream mining by considering context. In [11], context aware mining of streams is proposed where parameters that control input and output as well as the process of the algorithms are adjusted dynamically and autonomously according to the changes in context and situations. The demonstration and evaluation of the framework for a health monitoring application also exists in the same study. A domain specific context-aware ubiquitous stream mining model for intersection safety can be found in [20].

Resource-aware and/or context-aware adjustments of parameters that are mentioned above are proposed for mining data streams where data arrive continuously in a rapid speed. Hence, proposed solutions are specific to data stream mining and some of them are applicable only to specific data stream mining algorithms. 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 ([16][6][17]). Similarly, there are other prospective ubiquitous computing application areas such as user profiling ([10]), activity planning ([15]), personal health monitoring ([4]) 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 data mining algorithm running on a ubiquitous computing environment without imposing any restrictions on the type of data mining algorithm or parameters.

There are a few number of resource aware algorithms for mining non-stream data where the proposed algorithms adapt depending on the availability of a specific resource during its execution([3] [18] [19]). This approach which changes the algorithm's execution to optimize resource usage rather than configuring it has two drawbacks: solution is through a specific algorithm and general use with other data mining algorithms is not possible, and does not handle the situations where more than one resource is constrained.

The approach which we use for determining the configuration of data mining is quite different from the work given above such that we employ data mining to discover the appropriate parameter settings from the history of execution results whereas the proposed resource/context aware mining techniques do not use data mining methods. The reason we use a data mining technique for generating configuration decisions is 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.

In a previous work([2]) we used Bayesian Networks to represent the probabilistic relationships among context, algorithm parameter settings and the performance of data mining. In this work, we use a more flexible data mining model, and extend our previous work by formalizing the behavioral modeling and provide a comprehensive analysis of the data mining quality through taxonomies.

3. Modeling the Behavior of a Data Mining Algorithm

3.1. Problem Formulation and the Proposed Approach

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.

***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:

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

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. 1 illustrates an overall view of the approach which consists of the following basic steps: 1. Collect relevant information during the execution of the algorithm, 2. Maintain a collection of past execution data, 3. Learn a behavior model from the past execution data, and 4. Use behavior model for automatic configuration of data mining.

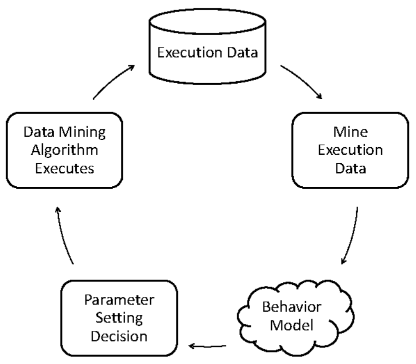


Figure 1: Overall View of Automatic Parameter Setting.

3.2. Execution Data

Formal definition of execution data is given below:

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: $Q_I = \{< q_1 : d_1, ..., q_n : d_n > | d_1 \in dom_1, ..., d_n \in dom_n\}$*

Table 1: Sample C, P and Q.

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

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 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 paper 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 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*) 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.

3.3. Classification Models to Represent Behavior

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 type 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.

3.4. 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 5. *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_D given in Definition 5 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 6. *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 7. *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:*
$Q_{A_I} = \{< q_A : d_A > | d_A \in dom_A\}$

4. 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 7):

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

On the other hand, Q_A which is a composite attribute formed by aggregation of a number of attributes may result in high number of classes preventing accurate classification. For this reason, we consider different abstraction levels of data mining quality as possible class labels.

4.1. Abstractions over the Class Label

A hierarchical structure that shows the taxonomy of data mining quality attributes in Q_D is used to abstract the data mining quality:

Definition 8. *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 $Q_G = \{g_1, g_2, ...\} = Q_T - Q_D$ 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 $g_i \in Q_G$.*

Table 2: 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:***
$Q_{Processor} = \{prc\_cycles, \%\_prc\}$	$Memorydom = \{VeryLow, Low, Average,$
$cdom \times pdom \rightarrow Processordom$	$High, VeryHigh\}$
$Q_{Memory} = \{avg\_mem, max\_mem\}$	$Processordom = \{VeryLow, Low,$
$adom \times mdom \rightarrow Memorydom$	$Average, High, VeryHigh\}$
$Q_{Resource} = \{Processor, Memory, battery\_usg\}$	$Resourcedom = \{Low, Average,$
$Processordom \times Memorydom \times bdom \rightarrow Resourcedom$	$High, VeryHigh\}$
$Q_{Model} = \{support, confidence\}$	$Modeldom = \{Low, Average, High\}$
$sdom \times cdom \rightarrow Modeldom$	$Overalldom = \{Good, Bad\}$
$Q_{Overall} = \{Model, Resource\}$	
$Modeldom \times Resourcedom \rightarrow Overalldom$	

- *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:*
    $f_{g_i} : q_1dom \times q_2dom \times .... \times q_{|Q_{g_i}|}dom \rightarrow g_idom$
    where q_idom *is the domain set of* $i'th$ *member of* Q_{g_i}*.*

Data mining quality abstraction given in Definition 8 is explained by the following example. Discretized data mining quality schema, Q_D in Table 2 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 $confidence$ and $support$. Figure 2 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_idom) of abstract data mining quality features which are the generalizations of the *Memory*, *Processor* and *Resource* usage as well as

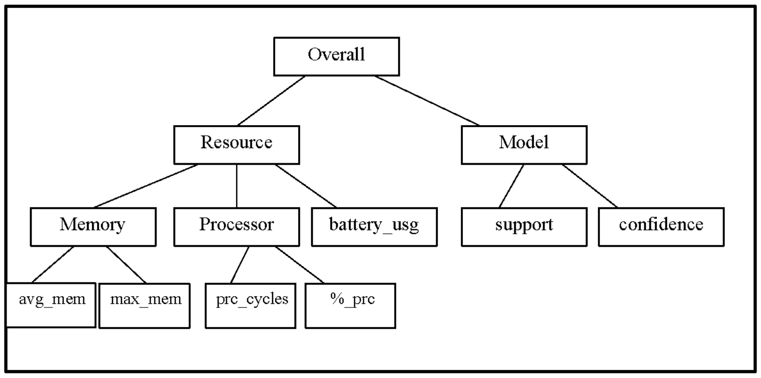


Figure 2: Data Mining Quality Taxonomy Specific to Association Rule Mining

the data mining *Model* and *Overall* quality are shown in Table 2. Successor sets of abstract data mining quality features ($Q_{Processor}$, Q_{Memory} and so on) are derived from the taxonomy T according to Definition 8. 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 2 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.

4.2. The AS/BM Strategy

We propose to abstract data mining quality using Definition 8 into several class labels resulting in more than one candidate behavior model for predicting algorithm configurations. In that respect, we propose a strategy that we call AS/BM (Accuracy Specificity Balanced Behavior Model Selection) in which 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. 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 9. *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 $O_{set} = \{Q_D\}$.*
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 3 is enumerated according to Definition 9 from the data mining taxonomy given in Figure 2. 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 9 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.

Table 3: L_{set}: Set of possible class label attribute sets.

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

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 10. *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 8 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 9).
  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}.

We consider 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 $S_{set}$, $Q_{AS_I}$ }
 1: $S_{set} \leftarrow \{\}$ {Screened set of class label attributes sets}
 2: $S_{attr} \leftarrow \{\}$ {Screened attributes from $Q_{AL}$}
 3: $recs \leftarrow \mathrm{g}_{count}(Q_{AL_I})$ {Total number of instances}
 4: $t\sharp \leftarrow threshold$ {Number of instances in a class}
 5: $t\% \leftarrow threshold\_percent$ {Number of instances in a class as % of $recs$}
 6: $n \leftarrow numberof\_classes\_below\_threshold$
 7: $t \leftarrow smaller\_of(t\sharp, recs * t\%)$
 8: for $k = 1$ to $|L_{set}|$ do
 9:    $\downarrow thresh \leftarrow 0$ {Number of classes below threshold}
10:    for $i = 1$ to $|dom_{al_k}|$ do
11:       $al_{k_i} \leftarrow member(dom_{al_k}, i)$
          {Returns the $i^{th}$ class in the domain}
12:       $c \leftarrow \mathrm{g}_{count}(\sigma_{q_{al_k} = al_{k_i}}(Q_{AL_I}))$
13:       if $c < t$ then
14:          $\downarrow thresh + +$
15:       end if
16:    end for
17:    if $\downarrow thresh > n$ then
18:       $S_{set} \leftarrow S_{set} \cup member(L_{set}, k)$
19:       $S_{attr} \leftarrow S_{attr} \cup q_{al_k}$
20:    end if
21: end for
22: $Q_{AS_I} \leftarrow \Pi_{S_{attr}}(Q_{AL_I})$
```

SCRN returns S_{set} and Q_{AS_I} which are the set of 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. Accuracy of M_i (acc_i) is estimated by k-fold cross-validation method where training and testing are repeated for k times.

Algorithm 2 MSEL

```
Require: Enumerated class labels are screened and $Q_{AS_I}$ is produced.
    {Input is $T$, $S_{set}$ , $E_I$, $Q_{AS_I}, coefficient$}
    {Output is $M$ }
 1: $max\_score \leftarrow 0$
 2: $top\_s \leftarrow return\_finestspecificitydegree(T)$
 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:         $l \leftarrow taxonomy\_level(T, qual\_attr)$
            {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: $M \leftarrow BUILD(E_I, s_{choose}, Q_{AS_I})$
```

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 predictive accuracy, 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 that will be used for extraction of data mining algorithm's configuration, is built using the class label which is scored highest in terms of accuracy and specificity of classification.

5. Experimental Evaluation

This section explains the experiments that we have performed in order to show the applicability of the proposed approach for obtaining a behavior model that can be used for recommending data mining configuration. 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, iii) assess the configuration decisions derived from the behavior model.

5.1. Experiment Setup

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.

5.1.1. Execution Data Generator Architecture

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

We have chosen well known association rule mining algorithm, Apriori ([1]) as the sample algorithm that is run by EDG for the experiment. Data generator software consists of JAVA programs (Figure 3) except the bottleneck creator modules which are C++ programs. Apriori is run by calling Weka ([12]) 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

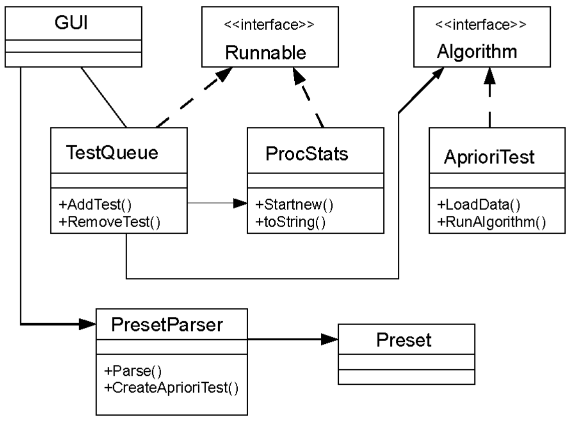


Figure 3: Class descriptions of EDG.

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 3. 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

Table 4: 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	

system information and statistics.

5.1.2. Experiment Data

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 4. We chose five of the parameters Weka receives for Apriori API's as configurable parameters (line 2 in Table 4) 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 4.

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 through 6 in Table 4). 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 as-

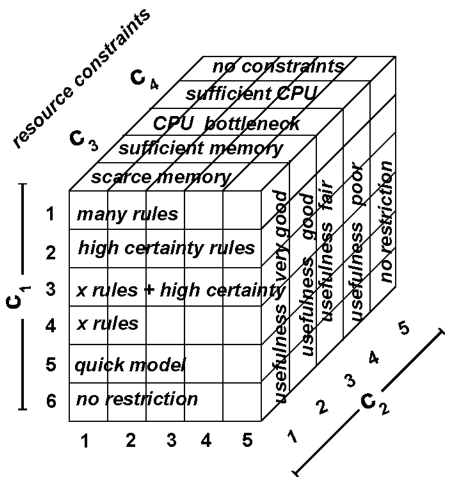


Figure 4: Cube of circumstances.

sociated 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 4).

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 9,10 in Table 4).

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 4 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 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 4 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.

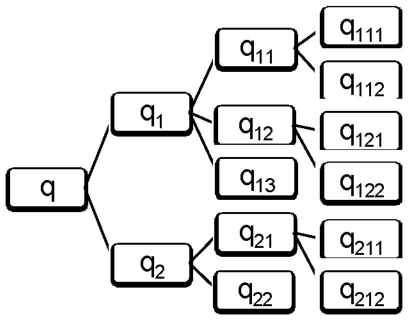
We run Apriori under every resource state given in Figure 4 ten times with each configuration generated from every configuration template of c_1's and c_2's state combinations. Hence, we produced 1500 execution records.

Finally, c_3 and c_4's (resources') usage measures by Apriori and quality indicators from the data mining model generated by Apriori were collected by EDG to constitute the base for the class label formation (line 11 in Table 4). In the next subsection, we explain in detail the transformations made on the data mining quality and the taxonomy used in the experiment.

5.1.3. 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 min-



q_{111}	average memory usage
q_{112}	max. memory usage
q_{11}	memory usage
q_{121}	total processor time
q_{122}	number of proc. 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 degree
q_{22}	number of rules in the model
q_{2}	model quality
q	overall quality

Figure 5: Data Mining Quality Taxonomy Used in the Experiment

imum 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 5.

Secondly, we aggregated the attributes in Q_D to generate aggregated data mining quality which is defined by Q_A (Definition 7). The aggregation function that we used consists of three simple steps: i) encode bins in the associated domain of every Q_D's attribute with ordinal values, ii) find the ordinal value for every tuple's every attribute in Q_{D_I}, iii) 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 5(a) 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 Figure 5(b). As can be seen in Figure 5(a) 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 successor set to its domain were defined for each element of Q_G. For

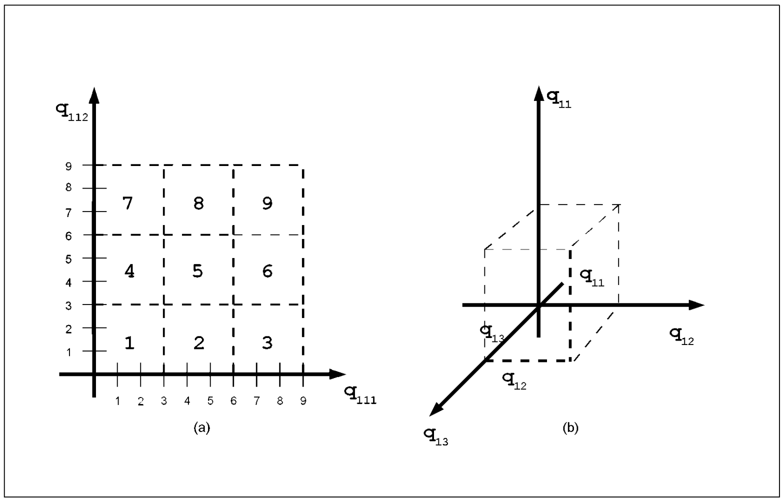
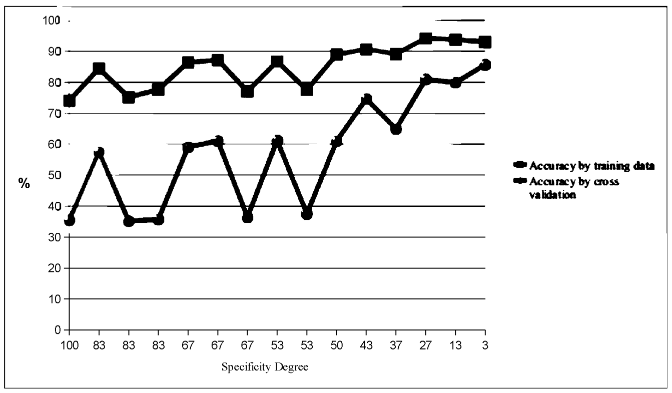


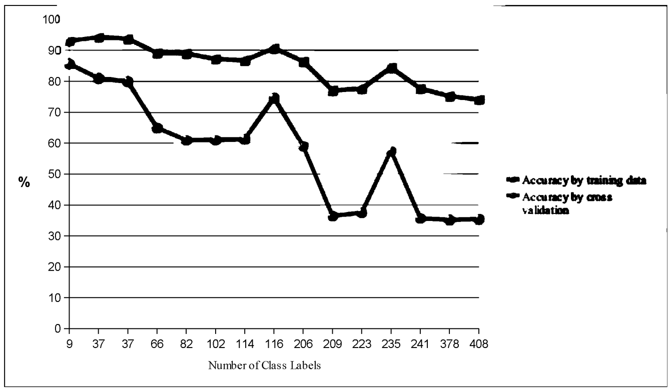
Figure 6: Mappings from Successor Set Domains to Abstract Domains

these mappings, we used either a two or three dimensional coordinate system depending on the number of attributes in the successor set of the abstract data mining quality attribute (Figure 6). The axes of each coordinate system were labeled by the ordinal values assigned to the bins in the domains of the attributes in the successor 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 6(a) 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 6(b) shows how three domains are mapped. Afterwards, we used the mappings to generate the abstract data mining quality (G in Definition 10) for *execution* file.

Finally, fifteen class label attribute sets were enumerated in L_{set} from the taxonomy (Definition 9). 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 10).



(a) Accuracy vs Specificity



(b) Accuracy vs Number of Classes

Figure 7: Analysis of Decision Tree Models

5.2. 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.

5.2.1. 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(a). 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(a) were normalized by dividing to the specificity degree of the decision tree that had the highest specificity. In Figure 7(a), training accuracy as well as the accuracy computed by using ten-fold cross validation were plotted. As usual, training data accuracy is higher than generalization accuracy estimated by cross validation.

General trend observed in both of the plots 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. 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.

5.2.2. 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(b), we plotted the decision tree models' accuracy figures derived from training data and computed by ten-fold cross validation respectively against the number of classes each decision tree possess. According to the results, accuracy generally deteriorates as the number of classified class labels increases which complies with the presumption.

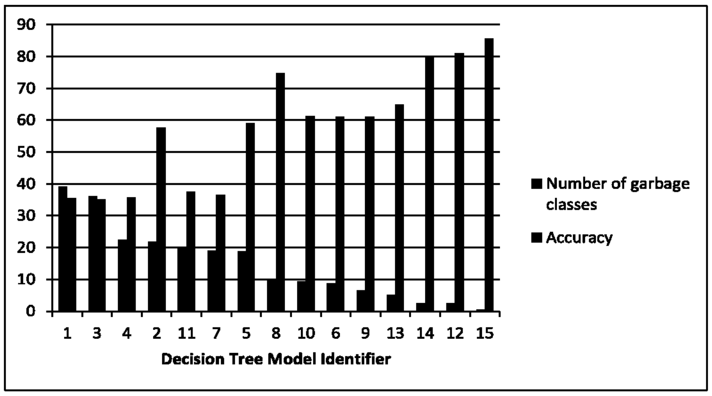


Figure 8: Effect of Garbage Classes on the Model's Accuracy

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 8, 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:

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

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 8) 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.

5.2.3. Assessment of Configuration Decisions

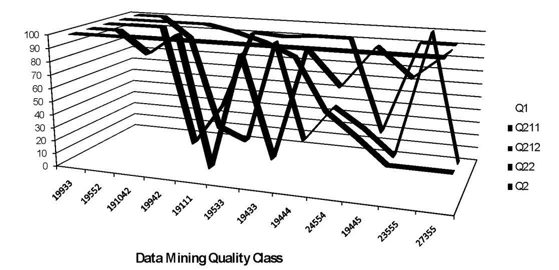
In this part of the experiment, we derived configuration decisions from the selected decision tree model and subsequently we used the derived configurations to configure Apriori. The purpose of this experiment is to compare the quality attained by Apriori executions which were run by a derived configuration 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:

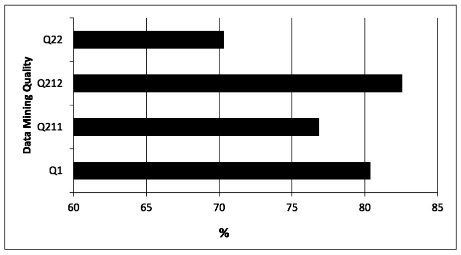
$$P4 <= 0.668\ AND\ P2 > 0.879\ AND\ P5 > 0.324\ AND$$
$$P5 <= 0.429\ AND\ P4 > 0.526\ AND\ P2 <= 0.976 :\ 19552$$

Note that, reverses of the data mining quality class abstraction and aggregation functions (Section 5.1.3) 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 4 because we executed Apriori for the circumstances in Figure 4. 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 4. 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.



(a) Successful Recommendations by Class



(b) Successful Recommendations in Overall

Figure 9: Recommendation Assessment Charts

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

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 5.1.3 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 9(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 9(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 9(a), we considered all recommendations were successful in terms of resource usage (q_1).

Percentages of successful recommendations in overall are given in Figure 9(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 9(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.

5.2.4. 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. 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.

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 $55 Mbytes$ whereas average peak memory usage was $49 Mbytes$. 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.

6. Conclusion

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.

We proposed a cost-effective solution aiming a reasonable accuracy without either restricting the number of quality features or the measurement variety of any quality feature since data mining quality has a significant importance on generating appropriate configuration decisions.

We currently work on the realization of the proposed approach within the framework of a ubiquitous computing application. We designed an application that downloads movie ratings from a social network site so that associations among the movie lists are mined on the mobile devices of interested users to recommend them movies. We apply the proposed approach to configure mining of social network data on an Android device for personal recommendations generation using collaborative filtering.

References

[1] R. Agrawal, R. Srikant, Fast Algorithms for Mining Association Rules, In : Proc. Int. Conf. on Very Large Data Bases, VLDB'94, Morgan Kaufmann, San Francisco, 1994, pp. 487–499.

[2] A. Cayci, S. Eibe, E. Menasalvas, Y. Saygin, Bayesian Networks to Predict Data Mining Algorithm Behavior in Ubiquitous Computing Environments, In: M. Atzmueller, A. Hotho, M. Strohmaier, A. Chin (Eds.), Proc. 2010 Int. Conf. on Analysis of Social Media and Ubiquitous Data, MSM'10/MUSE'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 119-141.

[3] K. Chuang, J. Huang, M. Chen, Mining Top-k Frequent Patterns in the Presence of the Memory Constraint, The VLDB Journal 17, 5 (Aug. 2008), 1321-1344.

[4] S. Consolvo, D. W. McDonald, T. Toscos, M. Y. Chen, J. Froehlich, B. Harrison, P. Klasnja, A. LaMarca, L. LeGrand, R. Libby, I. Smith, J. A. Landay, Activity Sensing in the Wild: a Field Trial of Ubifit Garden, In: Proc. 2008 SIGCHI Conference on Human Factors in Computing Systems, CHI '08. ACM, New York, NY, USA, 2008, pp. 1797-1806.

[5] C. Cumby, A. Fano, R. Ghani, M. Krema, Building Intelligent Shopping Assistants Using Individual Consumer Models, In : Proc. 10th Int. Conf. on Intelligent User Interfaces, IUI '05, ACM, New York, NY, USA, 2005, pp. 323-325.

[6] O. Flasch, A. Kaspari, K. Morik, M. Wurst, Nemoz A Distributed Framework for Collaborative Media Organization, In: Proc. 3rd Int. Workshop on Distributed Frameworks for Multimedia Applications, 2007.

[7] C. Franke, M. Karnstedt, K. Sattler, Mining Data Streams under Dynamicly Changing Resource Constraints, In : Knowledge Discovery, Data Mining, and Machine Learning, KDML, 2006, pp. 262-269.

[8] M.M. Gaber, S. Krishnaswamy and A. Zaslavsky, Resource-Aware Mining of Data Streams, Journal of Universal Computer Science 11, 8 (2005), 1440-1453.

[9] M.M. Gaber, P.S. Yu, A Holistic Approach for Resource-aware Adaptive Data Stream Mining, New Gen. Comput. 25, 1 (January 2007), 95-115.

[10] H. Haddadi, P. Hui, I. Brown, MobiAd: Private and Scalable Mobile Advertising, In: Proc. Fifth ACM International Workshop on Mobility in the Evolving Internet Architecture, MobiArch '10. ACM, New York, NY, USA, 2010, pp. 33-38.

[11] P.D. Haghighi, M.M. Gaber, S. Krishnaswamy , A. Zaslavsky, S.W. Loke, Context-aware Adaptive Data Stream Mining, Intell. Data Anal. 13, 3 (August 2009), 423-434.

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA Data Mining Software: An Update. SIGKDD Explorations, 11, 2009.

[13] C. Junghans, M. Karnstedt, M. Gertz, Quality-driven Resource-adaptive Data Stream Mining, SIGKDD Explor. Newsl. 13, 1 (August 2011), 72-82.

[14] D. Lee, W. Lee, Finding Maximal Frequent Itemsets over Online Data Streams Adaptively. In: Proc. Fifth IEEE International Conference on Data Mining, ICDM '05. IEEE Computer Society, Washington, DC, USA, 2005, pp. 266-273.

[15] L. Lin, D.J. Patterson, D. Fox, H. Kautz, Learning and Inferring Transportation Routines, Artif. Intell. 171, 5-6 (April 2007), pp. 311-331.

[16] I. Mierswa, K. Morik, M. Wurst, Collaborative Use of Features in a Distributed System for the Organization of Music Collections, In: Shen S., Cui L. (Eds.), Intelligent Music Information Systems: Tools and Methodologies, Idea Group Publishing, USA, 2007, pp.147-176.

[17] K. Morik, Nemoz: a Distributed Framework for Collaborative Media Organization, In: M. May and L. Saitta (Eds.) Ubiquitous Knowledge Discovery. Springer-Verlag, Berlin, Heidelberg, 2010, pp. 199-215.

[18] A. Nanopoulos, Y. Manolopoulos, Memory-adaptive Association Rules Mining, Information Systems 29, 5 (Jul. 2004), 365-384.

[19] S. Orlando P. Palmerini, R. Perego, F. Silvestri, Adaptive and Resource-aware Mining of Frequent Sets, In: Proc. 2002 IEEE international Conference on Data Mining (December 09 - 12, 2002). ICDM. IEEE Computer Society, Washington, DC, pp. 338.

[20] F. D. Salim, S. Krishnaswamy, S. W. Loke, A. Rakotonirainy, Context-aware Ubiquitous Data Mining Based Agent Model for Intersection Safety, In: T. Enokido, L. Yan, B. Xiao, D. Kim, D. Dai, L.T. Yang (Eds.), In: Proc. EUC 2005 Workshops: UISW, NCUS, SecUbiq, USN, and TAUES, Nagasaki, Japan, (December 6-9, 2005). Lecture Notes In Computer Science, vol. 3823. Springer-Berlin, Heidelberg, 2005, pp.61-70.

[21] R. Shah, S. Krishnaswamy, M.M. Gaber, Resource-Aware Very Fast K-Means for Ubiquitous Data Stream Mining, In: Proceedings of Second International Workshop on Knowledge Discovery in Data Streams, ECML 2005 and PKDD, 2005.