BABEŞ BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

SCIENTIFIC REPORT

October 2015- December 2015

MACHINE LEARNING FOR SOLVING SOFTWARE MAINTENANCE AND EVOLUTION PROBLEMS

–Învăţare automată în probleme privind evoluţia şi întreţinerea sistemelor informatice–

Project leader: Assoc. prof. CZIBULA István-Gergely

Project code: **PN-II-RU-TE-2014-4-0082**
Contract no.: **263/01.10.2015**

2015

Contents

1 Introduction **2**

2 Software defect detection **4**
2.1 Literature review 4

3 Unsupervised software defect detection **6**
3.1 Self-organizing maps 6
3.2 Methodology 7
3.2.1 Data pre-processing 7
3.2.2 The SOM model 7
3.3 Experimental evaluation 9
3.3.1 Data sets 9
3.3.2 Data pre-processing 9
3.3.3 Results 10
3.4 Discussion and comparison to related work 12
3.5 Design of the AMEL system 14

4 Adaptive association rule mining **17**
4.1 Background on relational association rule mining 18
4.1.1 Example 19
4.2 Methodology 21
4.3 Experimental evaluation 26
4.3.1 Experiments 26
4.3.2 Synthetic data 27
4.4 Discussion 31
4.4.1 Analysis of the *ARARM* method 31
4.4.2 Comparison to related work 34

5 Conclusions **36**

Chapter 1

Introduction

The present project proposes several research directions towards using machine learning techniques in the resolution of practical problems in software engineering. Since maintenance and software evolution problems are of utmost importance to the modern programmer, there exists a grown interest in the automation of as many processes in the lifecycle of software as possible, as well as the development of adequate mathematical models.

The major objective of our project is to contribute to improving the development process of software systems and their quality by obtaining innovative results in the search-based software engineering domain. An integrated software system (AMEL) will be also developed, with the purpose of assisting software developers in the maintenance and evolution stages of the software lifecycle, in activities such as: *defect prediction*, *package-level software refactoring* and the *identification of hidden dependencies in software systems*.

The problems tackled have a major practical importance since software developers face them daily, therefore the development of automated techniques which offer solutions to these problems would lead to more correct and error-free software. Since it is very hard to identify direct solutions, classification methods based on machine learning are extremely useful in solving the aforementioned problems.

In this report we will present the original scientific results which were obtained for achieving the objectives proposed in the project's work plan for the year 2015. The first scientific objective is related to the *development of new classification algorithms (SOM, fuzzy SOM, fuzzy RAR) for identifying entities with defects in software systems*. The second objective is connected to the *design of the AMEL integrated software system* as well as the development of the module related to defect detection.

The report is organized as follows. In Chapter 2 we present the problem of *software defect detection*, emphasizing the relevance of the problem, as well as existing related approaches for solving the problem. Our original approach for identifying the defects in software systems is introduced in Chapter 3. The last section from Chapter 3 presents the design of the AMEL system, as well as details regarding its development process. An original approach for adaptive relational association rule mining which can be used for defective software entities detection is introduced in Chapter 4.

The scientific results we have obtained during the period October 2015 - December 2015 are:

- An unsupervised learning based approach using *self-organizing maps* (SOM) for software defect detection.
- An approach for adaptive relational association rule mining which is useful for detecting defective software entities.
- **2** scientific papers: **1** ISI paper (in a SCI-E journal) [16] and **1** BDI paper [40].

We mention that the 2014 *impact factor* of our ISI publication is 2.810.

Chapter 2

Software defect detection

In order to increase the efficiency of quality assurance, *defect detection* tries to identify those modules of a software where errors are present. In many cases there is no time to thoroughly test each module of the software system, and in these cases defect detection methods can help by suggesting which modules should be focused on during testing.

Identifying the software entities (classes, modules, methods, functions, etc.) that are defective is of major importance as it facilitates further software evolution and maintenance. In order to deliver high quality software on time, software project managers, quality managers and software developers need to continuously monitor, detect and correct software defects at all stages of the development process. Software defect prediction helps in detecting, tracking and solving software anomalies that might have an effect on human safety and lives, particularly in safety critical systems. Defect prediction also allows changes to be made earlier in the software lifecycle, leading to a lower software cost and improving customer satisfaction. Recent results show that researchers should concentrate on improving the quality of the data in order to overcome the limits of the existing software prediction models [30].

Although many methods for software defect prediction do exist within the software engineering literature, recent researches are still carried out for proposing more accurate software defect predictors and for overcoming the drawbacks and limitations of the existing models.

2.1 Literature review

A review on unsupervised learning-based approaches existing in the defect prediction literature will be provided in the following.

Abaei et al. proposed in [1] a fault prediction method by utilizing self-organizing maps and thresholds. Two experiments are conducted in this paper: the first one considers the removal of the modules' labels and re-computing them afterwards by taking threshold values into account for some selected attributes (the ones for which threshold values are known). In the second experiment, a SOM is used for both clustering and evaluating the input data. Threshold values are used as well for labeling the units from the trained SOM. For this second experiment they report a good Overall Error, and in most cases their proposed solution improves classification of unlabeled program modules in terms of FPR (False Positive Rate) and FNR (False Negative Rate). One drawback to their approach is that they do not obtain good results when the data set is very small.

Another approach is presented in [4] that predicts software fault using a Quad Tree-based K-Means algorithm. The difference to the original K-Means algorithm is that the cluster centers are found by using Quad Trees. They evaluate their approach on various data sets, and compute the FPR, FNR and Overall Error for them. These values indicate that their approach is slightly better than other cluster center initialization techniques and they achieve slightly better results from fewer number of iterations.

The method presented in [54] uses the K-Means clustering algorithm as well, but it uses Hyper Quad Trees for determining the cluster centers. They present that Hyper Quad Trees are more efficient than simple Quad Trees because they produce more accurate centroids. After the K-Means algorithm is run, a threshold value is used to determine which cluster represents the defective and which represents the non-defective entities. The results for some public data sets confirm obtaining better outcomes in terms of FPR and Overall Error when comparing this approach to simple Quad Tree approach.

Tosun et al. used in [53] network measures to identify defective modules in software systems. Their approach uses the Naive Bayes classifier, together with a Call Graph Based Ranking (CGBR) framework. The experimental evaluation was performed on both small and large data sets and for three cases: complexity metrics only, network metrics only and a combination between them. The results show a great performance of applying network metrics for large data sets, but they do not provide significant improvement for small projects.

A clustering-based approach is presented in [10], where the Xmeans algorithm is used, an algorithm which is similar to K-means, but it can automatically determine the optimal number of clusters. The authors use the implementation of this algorithm from WEKA [25], and when the clusters are created, software metric threshold values are applied to the mean vector of each cluster in order to decide whether it represents the defective or the non-defective entities. They claim that this method proved better results than a simple threshold-based approach, fuzzy c-means and k-means.

Another unsupervised software fault prediction model is given by Park and Hong in [43] where clustering algorithms that determine the number of clusters automatically are used. They have a pre-processing step, where attribute selection is performed, using the CfsSubsetEval method from WEKA. Results achieved with the Xmeans and EM models from WEKA (which can automatically determined the optimal number of clusters) were compared to other results produced with Xmeans (in [10]) and Quad Tree based K-Means algorithm. They conclude that both Xmeans and EM have good results if attribute selection is not performed, results that are better than the existing ones in most of the considered cases.

Chapter 3

An approach for software defect detection using self-organizing maps

In the original paper [40] we have addressed the problem of software *defect detection*, an important problem which helps to improve the software systems' maintainability and evolution. In order to detect defective entities within a software system, a self-organizing feature map was proposed. The trained map was able to identify, using unsupervised learning, if a software module is defective or not. We experimentally evaluated our approach on three open-source case studies, also providing a comparison with similar existing approaches. The obtained results emphasized the effectiveness of using self-organizing maps for software defect detection and confirmed the potential of our proposal.

We are proposing in this chapter an unsupervised machine learning method based on self-organizing maps for detecting defective entities within software systems [40]. The self-organizing map architecture was previously applied in the literature for defect detection, but using a kind of hybrid approach, where different threshold values for some software metrics were also used [1]. To our knowledge, there is no approach in the search-based software engineering literature similar to ours. The unsupervised model introduced in this paper proved to outperform most of the similar existing approaches, considering the data sets used for evaluation.

The rest of the chapter is structured as follows. Section 3.2.2 presents the fundamentals of self-organizing maps. Our proposal for identifying software defects using self-organizing feature maps is introduced in Section 3.2. Section 3.3 provides an experimental evaluation of our approach, while an analysis of the obtained results and comparison with existing similar work is given in Section 3.4.

3.1 Self-organizing maps

A *self-organizing map* (SOM) [49] is a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional (usually two-dimensional) representation of the training samples, called a *map* [22]. Self-organizing maps use a neighborhood function to preserve the topological relationships in the input space and are related to the category of *competitive learning* networks. Self-organizing maps are considered in the neural networks literature as the most innovative form of unsupervised learning.

The SOM provides a topology preserving mapping from the multidimensional input space to the map neurons (units). Each neuron from the input layer of a SOM is connected to each neuron from the output layer and each connection has an associated weight. The topology preservation property means that a SOM groups similar input instances on neurons that are close on the SOM [32]. The map is usually trained using the Kohonen algorithm [49].

The trained self-organizing map is able to provide clusters of similar data items [36]. This

particular characteristic of SOMs makes them appropriate for data mining tasks that involve classification and clustering of data items [36]. The SOM can be used as an effective tool for clustering as well as a tool for visualizing high-dimensional data.

3.2 Methodology

In this section we introduce our unsupervised neural network model for defect identification in software systems.

The main idea of this approach is to represent an entity (class, module, method, function) of a software system as a multidimensional vector, whose elements are the values of different software metrics applied to the given entity. We consider that a software system S is a set of components (called *entities*) $\mathcal{S} = \{s_1, s_2, ..., s_n\}$. We are considering a feature set of software metrics $\mathcal{SM} = \{sm_1, sm_2, ..., sm_k\}$ and thus each entity $s_i \in \mathcal{S}$ from the software system can be represented as a k-dimensional vector, having as components the values of the software metrics from $\mathcal{SM}$, $s_i = (s_{i1}, s_{i2}, \ldots, s_{ik})$ (s_{ij} represents the value of the software metric sm_j applied to the software entity s_i).

For each software entity, the label of the instance (defect or non-defect) is known. We mention that the labels will be used only in the pre-processing step and for evaluating the performance of the model.

3.2.1 Data pre-processing

The first step before applying the SOM approach is the *data pre-processing* step. During this step, the input data is scaled to [0,1] using the *Min-Max* normalization method, and then a feature selection step will be applied. Details about the feature selection step will be given in the experimental part of the paper (Section 3.3). After applying the feature selection step, m software metrics (features) are selected to be further used for building the SOM.

Regarding the normalization method, we have to mention that in our approach the *minimum* and *maximum* values for the software metrics (features) from the training data are used for the *Min-Max* normalization step. We have focused in this paper only on the unsupervised scenario of grouping the existing entities from a software system into *defective* or *non-defective*. In a supervised learning scenario, in which new testing data is used, it is not useful to use for normalization the *minimum* and *maximum* values for the features from the training data. Instead, it would be a good idea to use, for each software metric, the *minimum* and *maximum* values for that software metric. Further extensions of our approach will investigate this situation.

3.2.2 The SOM model

Before designing the SOM model, the input data set is pre-processed. For the training step of the SOM, a distance function between the input instances is required. We are considering the *distance* between the high-dimensional representation of two software entities s_i and s_j as the *Euclidean Distance* between their corresponding vectors of software metrics values. We have chosen the *Euclidean distance* because it is the most often used distance measure for SOM-based approaches and because, intuitively, considering the m-dimensional instances (preprocessed as mentioned in Section 3.2.1), the *Euclidean distance* will assign low distances to similar entities that are very likely to have the same output class (defect or not). Nevertheless, in the future we intend to extend our approach to use other distance measures as well.

The set of pre-processed software entities from the data set $\mathcal{S}$ are grouped into clusters using a SOM. For the self-organizing map, the *torus* topology is used (Figure 3.1). In geometry, a torus is a surface of revolution generated by revolving a circle in the three dimensional

space about an axis coplanar with the circle. It is shown in the literature that this topology provides better neighborhood than the conventional one [33].

Figure 3.1: A torus

The goal of this step is to obtain two clusters corresponding to the two classes of instances: *defects* and *non-defects*. For grouping the software entities the following steps are performed.

- **Map Construction**. For a given number of *epochs* (training episodes), perform the following. Each m-dimensional training instance (software entity) is fed to the map. For each instance s_i the following steps are performed:

  1. **Matching**. The neuron having its weight vector closest (considering the *Euclidean distance*) to instance s_i is declared the "winning" neuron. This is a competition phase, in which the output units from the map compete to match the input instance.
  2. **Updating**. After the "winning neuron" was identified, the connection weights of the winning unit and its neighbors are updated, such that are moved in the direction of the input instance by a factor determined by a learning rate.

- **Visualization**. After the training phase (the steps described above) was completed, in order to visualize the obtained map, the U-Matrix method [31] is used. The U-Matrix value of a particular node from the map is computed as the average Euclidean distance between the node and its closest 4 or 8 neighbors. These distances can be then be viewed as heights giving a U-Matrix landscape. The U-Matrix may be interpreted as follows [31]: high places on the U-Matrix encode data that are dissimilar while the data falling in the same valleys represent input instances that are similar. Thus, instances within the same valley can be grouped together to represent a cluster. Each cluster visualized on the map identifies a class of instances.

Once the map was built, it may also be used in a supervised learning scenario for classifying a new software entity. First, the "winning neuron" corresponding to this instance is determined (as indicated at the **Matching** step above). The cluster (class) to which the winning neuron belongs will indicate the class membership of the given entity.

3.2.2.1 Testing

For evaluating the performance of the SOM model, we are using several evaluation measures from the supervised classification literature. Since the training instances were labeled, the labels are used to compute the confusion matrix for the two possible outcomes (*non-defect* and *defect*). Considering the *defective* class as the *positive* one and the *non-defective* class as the *negative* one, the confusion matrix [51] for the defect detection task consists of: the number of *true positives* (TP), *false positives* (FP), *true negatives* (TN) and *false negatives* (FN).

Considering the values computed from the confusion matrix, the following evaluation measures will be used in this paper:

1. False Positive Rate (FPR), computed as $\frac{FP}{FP+TN}$.
2. False Negative Rate (FNR), computed as $\frac{FN}{FN+TP}$.
3. Overall Error (OE), computed as $\frac{FN+FP}{FN+FP+TN+TP}$.

We have decided to use these measures, because they are used in papers presenting similar approaches, so a direct comparison of the results is possible. But the data sets used for the experiments are imbalanced, so we have decided to compute the value of a fourth performance measure as well: the *Area Under the ROC Curve (AUC)* [23]. The *ROC* curve is a two-dimensional plot of *sensitivity* vs. *(1-specificity)*, which in our case contains one single point, linked to the $(0, 0)$ and $(1, 1)$ points.

3.3 Experimental evaluation

In this section we provide an experimental evaluation of the SOM model (described in Section 3.2) on three case studies which were conducted on open source data sets. We mention that we have used our own implementation for SOM, without using any third party libraries.

3.3.1 Data sets

We have used three openly available data sets for the experimental evaluation of our model, called *Ar3*, *Ar4* and *Ar5*, which can be downloaded from [19]. All three data sets come from a Turkish white-goods manufacturer embedded software implemented in C. They all contain the value of 29 different McCabe and Halstead software metrics, computed for the functions and methods from the software systems, and one class label, denoting whether the entity is defective or not. The *Ar3* data set contains metric values for 63 entities, out of which 8 are defective. The *Ar4* data set contains 107 entities, out of which 20 defective, while the *Ar5* data set has 36 entities, out of which 8 are defective.

For the SOM used in the experiments, the following parameter setting was used: the *number of training epochs* was set to 100000, the *learning coefficient* was set to 0.7, the *radius* was computed as half of the maximum distance between the neurons and the neighborhood function. We have tried out different parameter settings and we have achieved the best results with these values. However, we will perform in the future a thorough study to investigate the effect of different parameter settings.

3.3.2 Data pre-processing

In order to analyze the importance of the features, we are using the *information gain* measure. The *information gain* (IG) of a feature expresses the expected reduction in entropy determined by partitioning the instances according to the considered feature [42]. More exactly, the IG measure indicates the relevance of a feature in the defect classification task. Since the software metrics values (features values) are real numbers, in order to compute the information gain of the attributes we first discretize their values by dividing their interval of variation into ten sub-intervals.

For a better data analysis, we have computed the information gain of the features from the data set obtained using all three data sets (*Ar3*, *Ar4* and *Ar5*) together. The information gain values for the features are shown in Figure 3.2.

Starting from the IG values of the software metrics, we have chosen a threshold value τ and considered only the attributes whose IG was higher than this threshold. Out of these attributes, we selected those that measure different characteristics of the software system. For the threshold τ we have selected the value 0.163, because we have achieved the best

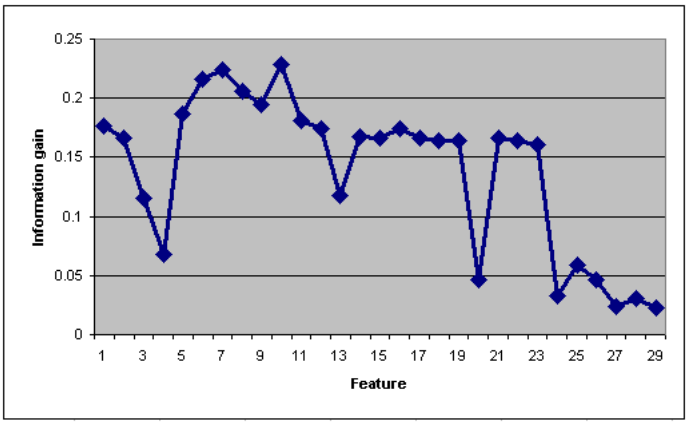

Figure 3.2: Information gain for the features.

results with this value. Out of the 18 software metrics whose value was higher than τ, we have selected the following 9 metrics: *halstead_vocabulary*, *total_operands*, *total_operators*, *executable_loc*, *halstead_length*, *total_loc*, *condition_count*, *branch_count*, *decision_count*. These selected attributes were used in the experimental evaluation on all three considered data sets. We mention that a different, possibly automatic, attribute selection method can also be implemented considering the IG values and will be further investigated.

3.3.3 Results

We are presenting in this section the results we have obtained by applying the SOM model on the *Ar3*, *Ar4* and *Ar5* data sets. For each data set considered for evaluation, the experiments are conducted as follows. First, the data pre-processing step is applied and then the methodology indicated in Section 3.2 is used for an unsupervised construction of a torus SOM. The U-Matrix corresponding to the trained SOM will be visualized (the red labels on the U-Matrix represent the defective entities and the yellow labels represent the non-defective ones). Then, the evaluation measures presented in Section 3.2.2.1 will be computed for evaluating the performance of the obtained results.

3.3.3.1 The *Ar3* data set

A torus SOM, consisting of 150x8 nodes, was trained on the set of software entities from the *Ar3* data set. Figure 3.3 depicts the U-Matrix visualization of the trained SOM.

Visualizing the U-Matrix for the resulting map, we have identified the two clusters, representing the defective and non-defective entities. The cluster with the defective entities contains 6 defective entities and 1 non-defective entity, thus we have 1 FP entity and 2 FN ones. All the other entities are placed in the correct cluster. The values of the performance measures from Section 3.2.2.1 are presented in the first three cells of the first row of Table 3.1.

3.3.3.2 The *Ar4* data set

A torus SOM, consisting of 150x8 nodes, was trained on the set of software entities from the *Ar4* data set. The U-Matrix visualization of the obtained SOM is illustrated in Figure 3.4.

Visualizing the U-matrix for the resulting map, we have identified the two clusters which represent the defective and non-defective entities. The cluster with the defective entities contains 10 defective entities and 2 non-defective entities. Consequently we have 10 FN

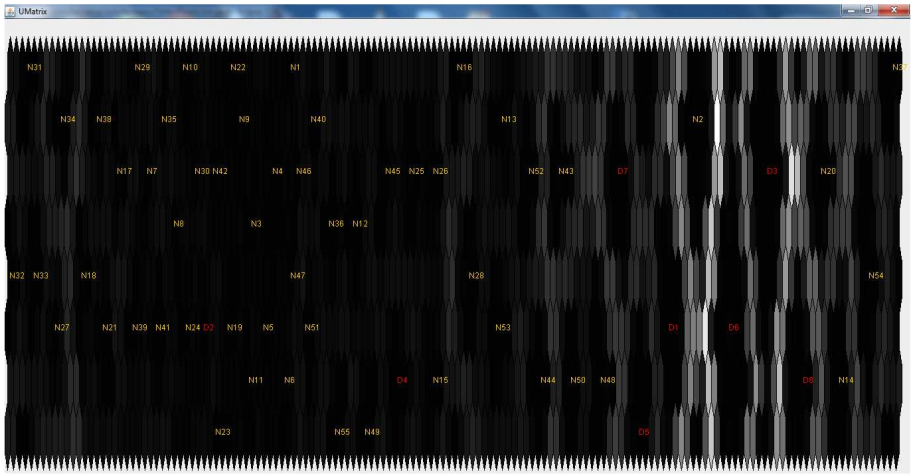

Figure 3.3: U-Matrix for the *Ar3* data set.

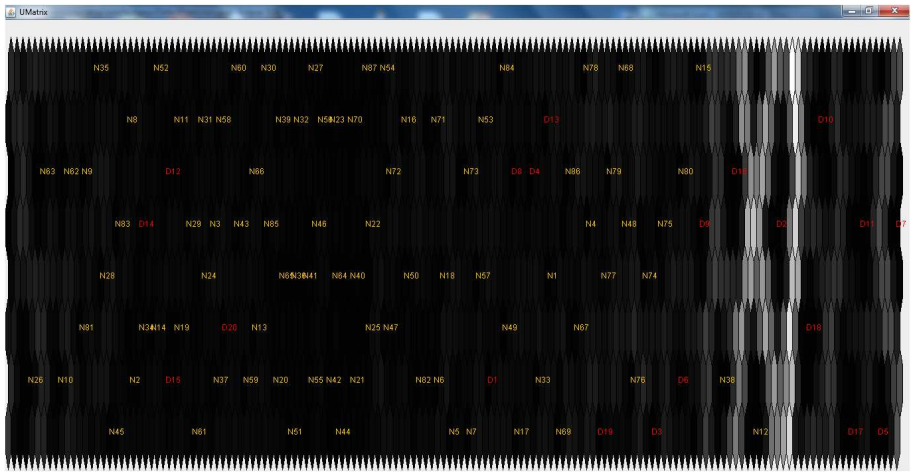

Figure 3.4: U-Matrix for the *Ar4* data set.

entities and 2 FP ones. The values of the performance measures are presented in the middle three cells of the first row of Table 3.1.

3.3.3.3 The *Ar5* data set

A torus SOM, consisting of 150x8 notes, was trained on the set of software entities from the *Ar5* data set. The U-Matrix visualization of the trained SOM is presented in Figure 3.5.

From Figure 3.5 we can observe that the obtained SOM indicates a good topological mapping of the input instances, and also identifies subclasses within the defective and non-defective classes. Most of the defective entities are grouped together (there is only one FN), but there is also one non-defective entity in this cluster, so we have 1 FP as well. The values of the performance measures for this data set are presented in the last three cells of the first row of Table 3.1.

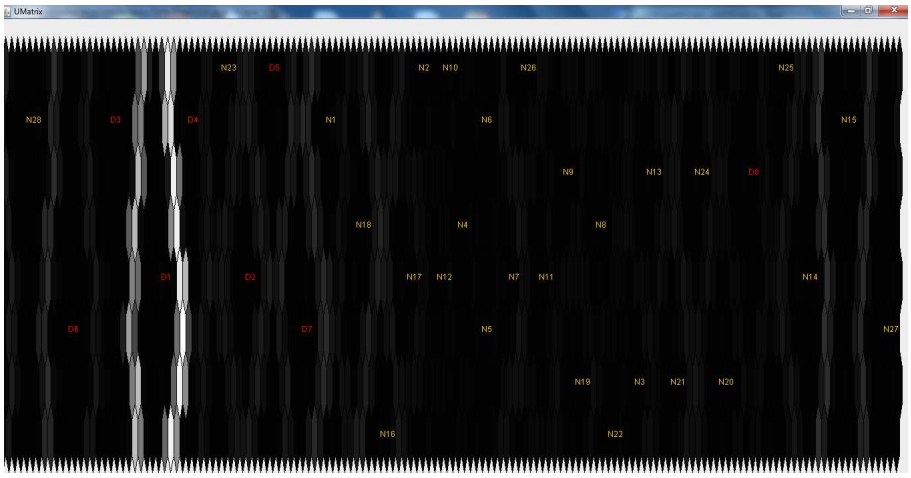

Figure 3.5: U-Matrix for the *Ar5* data set.

3.4 Discussion and comparison to related work

An analysis of the approach we have introduced in Section 3.2 for detecting the defective entities from software systems will be provided in the following. A discussion on the obtained experimental results, as well as a comparison of them with similar approaches from the literature is conducted.

As presented in the previous section, our approach was capable of separating the defective and non-defective entities in two clusters, obtaining a good topological mapping of the input instances. Even if the separation was not perfect, for all three data sets we had both false positive and false negative entities. A major advantage of the self-organizing map is that it does not require supervision and no assumption about the distribution of the input data is made. Thus, it may find unexpected hidden structures from the data being investigated. Moreover, as seen from our experiments (Section 3.3), it is interesting that the SOM is able to detect, within the defective/non-defective class, subclasses of instances. This would be very useful, from a data mining perspective, since it may provide useful knowledge for the software developers.

As the accuracy of the trained SOMs depends on the choice of some parameters (number of training epochs, learning coefficient, neighborhood function), we have to measure it. One method for evaluating the quality of the resulting map is to calculate the *average quantization error* over the input samples, defined as the Euclidean norm of the difference between the input vector and the best-matching model [34]. Figures 3.6, 3.7 and 3.8 give a graphical representation of the average quantization error during the training steps, for each case study considered for evaluation. It can be easily seen that while the error fluctuates at the beginning of the training phase, it decreases during it, and after the training is completed, a very small average quantization error of the trained maps is obtained (1.6×10^{-6} for *Ar3*, 1.7×10^{-4} for *Ar4* and 6.7×10^{-13} for *Ar5*), which shows the accuracy of the trained maps.

Considering the subclasses identified within the clusters for the defective and non-defective entities as further work we propose to analyze these subclasses to identify the characteristics of the software entities placed in them.

Table 3.1 presents the values of the FPR, FNR and OE performance measures computed for the results of our approach, but it also contains values reported in the literature for some existing approaches, presented in Section 2.1. The Hyper Quad Tree-based approach presented in [54] does not report FNR values, this is marked with "NR" in the table.

From Table 3.1 we can see, that even if our approach does not provide the best results in each case, it has better results than most of the approaches. Out of 51 cases in total, our

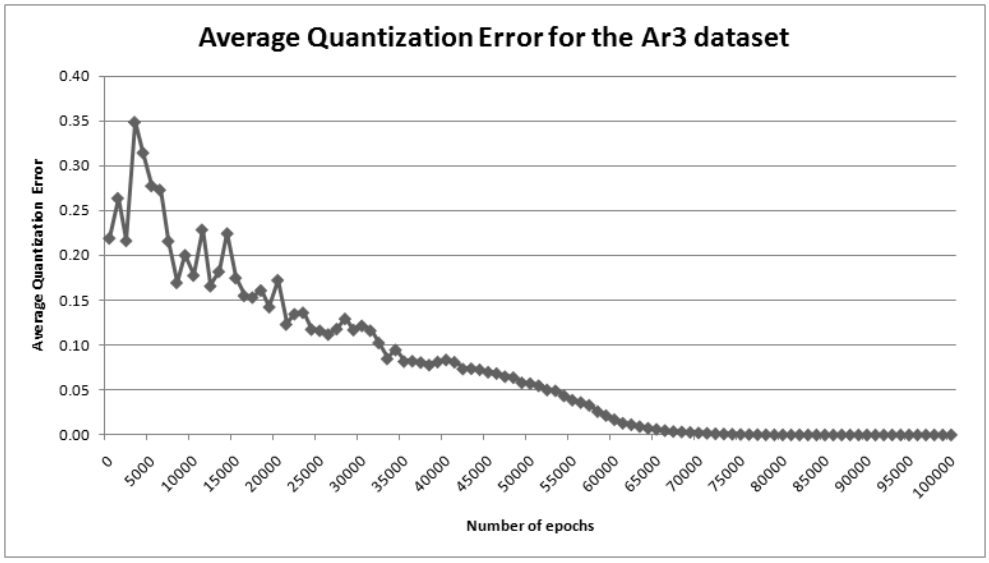

Figure 3.6: Average Quantization Error for the *Ar3* data set.

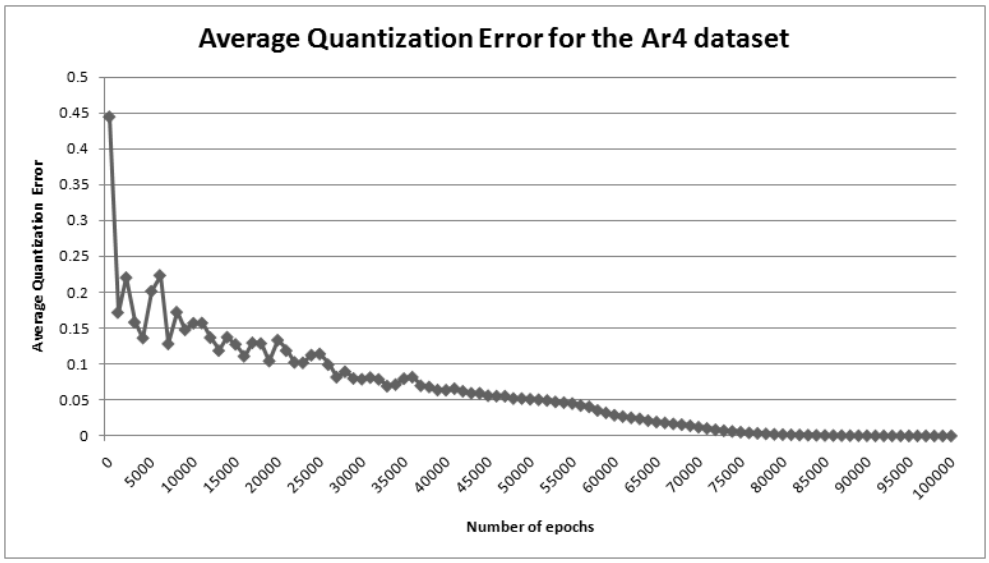

Figure 3.7: Average Quantization Error for the *Ar4* data set.

algorithm has a better or equal value for a performance measure in 43 cases, which represents **84.3%** of the cases.

The first line of Table 3.2 presents the value of the AUC measure computed for our approach. As presented in Section 3.2.2.1, for computing the value of the AUC measure we need to compute the *sensitivity* and *specificity* of the classification. Since *sensitivity* is equal to $1 - FNR$ and $(1 - specificity)$ is equal to FPR, we computed the value of the AUC measure for those approaches from the literature which report both of these values. They are presented in Table 3.2 as well, and for each data set the best value is marked with bold. Our approach has the highest AUC value for the *Ar5* data set, the second highest value for the *Ar3* data set and the fourth highest value for the *Ar4* data set. Interestingly, for the *Ar3* and *Ar4* data sets the best value is achieved by the other SOM-based approach presented in the literature, suggesting that SOMs are indeed suitable for this problem.

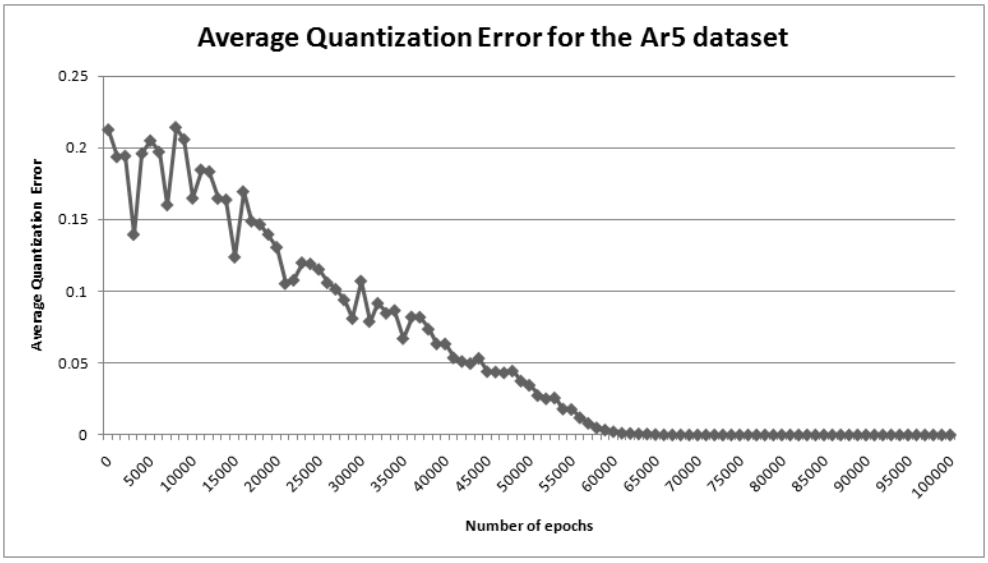

Figure 3.8: Average Quantization Error for the *Ar5* data set.

	Ar3			Ar4			Ar5		
Approach	**FPR**	**FNR**	**OE**	**FPR**	**FNR**	**OE**	**FPR**	**FNR**	**OE**
Our SOM	0.0182	0.25	0.0476	0.0230	0.5	0.1121	0.0357	0.125	0.0556
SOM and threshold [1]	0	0.25	0.0556	0.1034	0	0.0938	0.0714	0.25	0.1111
K-means - QT [4]	0.3454	0.25	0.3333	0.0459	0.45	0.1214	0.1428	0.125	0.1388
K-means - Hyper QT [54]	0.0263	NR	0.0263	0.1875	NR	0.1846	0.0246	NR	0.0246
XMeans [10]	0.3455	0.25	0.3333	0.4483	0.05	0.3738	0.1429	0.125	0.1389
XMeans [43]	0.0727	0.25	0.0952	0.023	0.6	0.1308	0.149	0.125	0.1389
EM [43]	0.1091	0.25	0.127	0.023	0.6	0.1308	0.149	0.25	0.1667

Table 3.1: Comparison of the performance of our method to existing approaches.

3.5 Design of the AMEL system

The AMEL system is designed to be highly scalable and versatile, as it can be regarded either as a framework or as a stand-alone software product. This is acheived by enforcing a 3-tier layered architecture to the software design (see Figure 3.9) which clearly separates the *presentation layer* (represented by any form of graphical user interface) from the *business layer* which manipulates data from the underlying *data layer* (currently storing data in designated .xml files). The system is developed using JDK 8 and the default GUI is designed and implemented using the Swing toolkit.

The development process follows the *Contract Driven Design* (CDD) approach, enabling efficient interface-based programming which facilitates the systems's evolution and maintainability.

The Graphical User Interface (GUI) of AMEL will allow users to easily work with all of AMEL's scientific modules (one for each of the problems we plan to research: *Software Defect Prediction, Package-level Software System Refactoring, Identifying Hidden Dependencies in Software Sysems*. The class diagram for our proposed design can be seen in Figure 3.9.

Since each problem is complex, the three modules will have to be separate subsystems. Each subsystem will offer user friendly ways to work with all of its features, starting from the input data and ending with the presentation of the results. Since the three models will

Approach	Ar3	Ar4	Ar5
Our SOM	0.866	0.739	**0.92**
SOM and threshold [1]	**0.875**	0.**948**	0.839
K-means - QT [4]	0.702	0.752	0.866
XMeans [10]	0.702	0.751	0.866
XMeans [43]	0.839	0.689	0.863
EM [43]	0.820	0.689	0.801

Table 3.2: Comparison of AUC values.

mostly be independent, it will be possible to use each one as a standalone application.

The GUI of the AMEL system, as well as the actual problem-solving modules it encompasses, will be written using cross-platform programming languages and libraries, ensuring that the system will be usable on most operating systems in use today. Moreover, we are also considering providing either a text-based interface or standalone libraries that expose our problem-solving functionalities, in order to make integrating our work into other systems possible.

As common helper functionalities, we plan to implement automatic data generation and generic cross-validation methods, which will be exposed through a main user interface.

Figure 3.10 presents the simplified class diagram for the implementation of the Self-Organizing Maps approach, which is a part of the *defect detection* module.

The module was developed to be general and easily extensible, and it is divided into five packages:

- The *som* package contains classes which represent some of the basic elements of a self-organizing map: a neuron of the map (represented by class *SOMNeuron*), the best matching unit (class *BMU*), a listener, which can be used to log information during the training process (class *SOMTrainingListener*), and, obviously, the self-organizing map itself (class *SOM*).

- The *topology* package contains the abstract representation of a self-organizing map topology and implementations for different existing topologies: rectangular, two-dimensional lattice and the torus that was used for the experimental evaluation of the above-presented approach.

- The *traindata* package contains representations of different training data: a randomly generated array, data from a file, filtered data and normalized data.

- The *trainsamplechooser* package contains the implementation of different stategies for choosing an input data during the training process.

- The *umatrix* package contains the classes needed for the creation and visualization of the U-Matrix for the trained map.

Figure 3.9: Package diagram

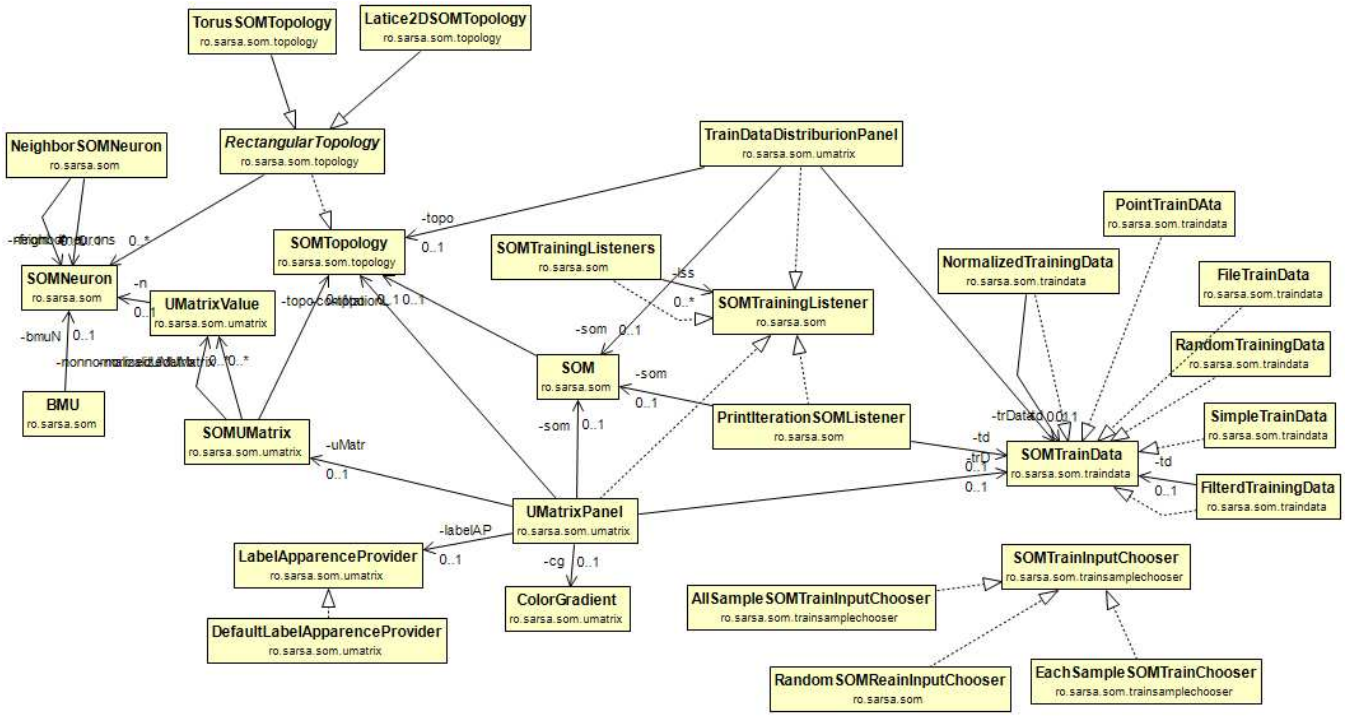

Figure 3.10: UML diagram for the defect detection module

Chapter 4

A novel approach to adaptive association rule mining

This chapter paper focuses on the adaptive relational association rule mining problem. Relational association rules represent a particular type of association rules which describe frequent relations that occur between the features characterizing the instances within a data set. We aim at re-mining an object set, previously mined, when the feature set characterizing the objects increases. An adaptive relational association rule method, based on the discovery of interesting relational association rules, is proposed. This method, called *ARARM* (*Adaptive Relational Association Rule Mining*) adapts the set of rules that was established by mining the data before the feature set changed, preserving the completeness . We aim to reach the result more efficiently than running the mining algorithm again from scratch on the feature-extended object set. Experiments testing the method's performance on several case studies are also reported. The obtained results highlight the efficiency of the *ARARM* method and confirm the potential of our proposal.

It is well known that mining different kinds of data is of great interest in various domains such as medicine, bioinformatics, bioarchaeology, as it can lead to the discovery of useful patterns and meaningful knowledge.

Association rule mining [7] means searching attribute-value conditions that occur frequently together in a data set [26], [52], [55]. Ordinal association rules [8] are a particular type of association rules. Given a set of records described by a set of characteristics (features or attributes), the ordinal association rules specify ordinal relationships between record features that hold for a certain percentage of the records. However, in real world data sets, features with different domains and relationships between them, other than ordinal, exist. In such situations, ordinal association rules are not powerful enough to describe data regularities. Consequently, *relational association rules* were introduced in [47] in order to be able to capture various kinds of relationships between record features.The *DRAR* method (*Discovery of Relational Association Rules*) was introduced for mining interesting relational association rules within data sets [47].

Relational association rule mining can be used in solving problems from a variety of domains, such as: data cleaning, natural language processing, databases, healthcare, bioinformatics, bioarchaeology, etc. We have previously applied, so far, relational association rule mining in different data mining tasks such as: medical diagnosis prediction [6], predicting if a DNA sequence contains a promoter region or not [15], software defect prediction [17], software design defect detection [18], data cleaning [8].

The method *DRAR* for relational association rule mining starts with a known set of objects, measured against a known set of features and discovers interesting relational association rules within the data set. But there are various applications where the object set is dynamic, or the feature set characterizing the objects evolves. Obviously, for obtaining the interesting

relational association rules within the object set in these conditions, the mining algorithm can be applied over and over again, beginning from scratch, every time when the objects or the features change. But this can be inefficient.

In paper [16], we proposed an adaptive relational association rule method, named *Adaptive Relational Association Rule Mining (ARARM)*, that is capable to efficiently mine relational association rules within the object set, when the feature set increases with one or more features. The *ARARM* method starts from the set of interesting rules that was established by applying *DRAR* before the feature set changed and adapts it considering the newly added features. The result is reached faster than running *DRAR* again from scratch on the feature-extended object set.

We have to mention that the adaptive relational association rule mining method, proposed in paper [16], is a novel approach. There exist in the data mining literature approaches which consider the adaptive association rule mining process for particular problems, but none of them deal with *relational* association rules as in our proposal.

The remaining of the chapter is organized as follows. A background on relational association rule mining is given in Section 4.1. The *Adaptive Relational Association Rule Mining* (*ARARM*) method is described in Section 4.2. Section 4.3 presents the experimental evaluation of our approach and shows the efficiency of the proposed method on several case studies. An analysis of the adaptive approach introduced in this paper, as well as a discussion on the obtained results and comparison to related work are given in Section 4.4.

4.1 Background on relational association rule mining

There is a continuous interest in applying association rule mining [50] in order to discover relevant patterns and rules in large volumes of data. Data mining methods [58], [3] are applied in various domains such as medicine, bioinformatics, bioarchaeology, software engineering.

In order to be able to capture various kinds of relationships between record attributes, the definition of ordinal association rules from [8], [9] was extended in [47] towards *relational association rules*.

In the following we will briefly review the concept of *relational association rules*, as well as the mechanism for identifying the relevant relational association rules that hold within a data set.

Let $R = \{r_1, r_2, \ldots, r_n\}$ be a set of *instances* (entities or records in the relational model), where each instance is characterized by a list of m attributes, $(a_1, \ldots, a_m)$. We denote by $\Phi(r_j, a_i)$ the value of attribute a_i for the instance r_j. Each attribute a_i takes values from a domain D_i, which contains the empty value denoted by ε. Between two domains D_i and D_j relations can be defined, such as: less ($<$), equal ($=$), greater or equal ($\geq$), etc. We denote by M the set of all possible relations that can be defined on D_i x D_j and by $\mathcal{A} = \{a_1, \ldots, a_m\}$ the attribute set.

Definition 1 [47] *A* relational association rule *is an expression* $(a_{i_1}\ \mu_1\ a_{i_2}\ \mu_2\ a_{i_3} \ldots \mu_{\ell-1}\ a_{i_\ell})$, *where* $\{a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}\} \subseteq \mathcal{A}$, $a_{i_j} \neq a_{i_k}$, $j, k \in \{1 \ldots \ell\}$, $j \neq k$ *and* $\mu_i \in M$ *is a relation over* $D_{i_j} \times D_{i_{j+1}}$, D_{i_j} *is the domain of the attribute* a_{i_j}. *If:*

a) $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ *occur together (are non-empty) in* $s\%$ *of the* n *instances, then we call* s *the* ***support*** *of the rule,*

and

b) we denote by $R' \subseteq R$ *the set of instances where* $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ *occur together and* $\Phi(r', a_{i_j})\ \mu_1\ \Phi(r', a_{i_{j+1}})$ *is true* $\forall 1 \leq j \leq \ell - 1$ *and for each instance* r' *from* R'*; then we call* $c = |R'|/|R|$ *the* ***confidence*** *of the rule.*

The *length* of a relational association rule is given by the number of attributes in the rule. Users usually need to uncover interesting relational association rules that hold within a data set; they are interested in relational rules which hold in a minimum number of instances, that are rules with support at least s_{min}, and confidence at least c_{min} (s_{min} and c_{min} are user-provided thresholds).

A relational association rule in R is called *interesting* [47] if its support s is greater than or equal to a user-specified minimum support, s_{min}, and its confidence c is greater than or equal to a user-specified minimum confidence, c_{min}.

In [9] an A-Priori [2] like algorithm, called *DOAR* (Discovery of Ordinal Association Rules), was introduced in order to efficiently find all ordinal association rules (i.e. relational association rules in which the relations are ordinal) of any length, that hold over a data set. The *DOAR* algorithm was proven to be correct and complete and it efficiently explores the search space of the possible rules [9].

The *DOAR* algorithm was further extended in [47, 15] towards the *DRAR* algorithm (*Discovery of Relational Association Rules*) for finding interesting relational association rules, i.e. association rules which are able to capture various kinds of relationships between record attributes. The *DRAR* algorithm provides two functionalities: (a) it finds all interesting relational association rules of any length; (b) it finds all maximal interesting relational association rules of any length, i.e. if an interesting rule r of a certain length l can be extended with one attribute and it remains interesting (its confidence is greater than the threshold), only the extended rule is kept.

So far, *relational association rules* were successful in different data mining tasks in domains like: *medicine* (for diagnosis prediction [6]), *bioinformatics* (for predicting if a DNA sequence contains a promoter region or not [15], *software engineering* [17, 18], as well as for *data cleaning* tasks [8].

4.1.1 Example

In order to better explain the concept of relational association rules and the extension of *DOAR* algorithm [9] that is used for discovering relational association rules, we give an example of a relational association rule mining task within a software system.

Let us consider the Java code example shown in Figure 4.1. The example is taken from [48] and was used by the authors in order to illustrate the *Move Method* refactoring.

The dataset considered in the mining process consists of a set of *software entities* (instances), each software entity being characterized by a set of *software metrics* (features characterizing the instances).

We consider in our example that a software entity can be either an application class, or a method from an application class. The software metrics considered in our experiment are:

1. Depth in Inheritance Tree (DIT) [14].
2. Number of Children (NOC) [14].
3. Fan-In (FI) [29] [38].
4. Fan-Out (FO) [29] [38].

We have previously used these software metrics in [39] for a clustering based automatic identification of refactorings that would improve the internal structure of a software system.

Using the above mentioned software metrics, each software entity from the system presented in Figure 4.1 can be represented as a 4-dimensional vector, having as components the values of the considered metrics. The corresponding dataset is given in Table 4.1.

As all attributes in our experiment have numerical values, we have defined two possible binary relations between the attributes: $<$ and $>$.

```
public class Class_A {
  public static int attributeA1;
  public static int attributeA2;

  public static void mA1(){
      attributeA1 = 0;
      mA2();
  }

  public static void mA2(){
          attributeA2 = 0;
          attributeA1 = 0;
    }

  public static void mA3(){
          attributeA2 = 0;
          attributeA1 = 0;
          mA1();
          mA2();
  }
}
```

```
public class Class_B {
  private static int attributeB1;
  private static int attributeB2;

  public static void mB1(){
          Class_A.attributeA1=0;
          Class_A.attributeA2=0;
          Class_A.mA1();
    }

  public static void mB2(){
          attributeB1=0;
          attributeB2=0;
    }

  public static void mB3(){
          attributeB1=0;
          mB1();
          m2();
  }
}
```

Figure 4.1: Java code example

Table 4.1: Dataset corresponding to the system from Figure 4.1

Entity	*DIT*	*NOC*	*FI*	*FO*
Class_A	1	0	3	1
Class_B	1	0	0	2
mA1	1	0	2	1
mA2	1	0	2	0
mA3	1	0	0	2
mB1	1	0	1	1
mB2	1	0	1	0
mB3	1	0	0	2

Table 4.2: Interesting relational association rules extracted from the dataset given in Table 4.1

Length	Rule	Confidence
2	DIT > NOC	1
2	NOC < FI	0.625
2	NOC < FO	0.75
2	FI > FO	0.5
3	DIT >NOC < FI	0.625
3	DIT > NOC < FO	0.75
3	NOC <FI >FO	0.5
4	DIT> NOC < FI > FO	0.5

Table 4.3: Maximal interesting relational association rules extracted from the dataset given in Table 4.1

Length	Rule	Confidence
3	DIT > NOC <FO	0.75
4	DIT > NOC < FI >FO	0.5

We executed the *DRAR* algorithm with minimum support threshold of 0.9 and minimum confidence threshold of 0.4. The discovered interesting relational rules are shown in Table 4.2 and the maximal interesting association rules are given in Table 4.3. For each discovered rule, its confidence is also provided.

As it can be seen in the results above, interesting relational association rules can be discovered within the set of software entities. Further analysis of these relational association rules may provide relevant information regarding the analyzed software system.

4.2 Methodology

In this section, we introduce the adaptive relational association rule mining approach, as well as an algorithm called *ARARM* (*Adaptive Relational Association Rule Mining*) that is capable to efficiently mine relational association rules within a data set, when the feature set increases with one or more features.

Let us consider a data set $R = \{r_1, r_2, \ldots, r_n\}$ consisting of high-dimensional *instances* (objects). Each instance is characterized by a list of m attributes (also called features), $(a_1, \ldots, a_m)$ and is therefore described by an m-dimensional vector $r_i = (r_{i1}, \ldots, r_{im})$. Different types of relations can be defined between the values of the features characterizing the instances from the data set. We denote by $\mathcal{R}el$ the set of all possible relations that can be defined between the features values. As presented in Section 4.1, interesting relational association rules that are able to express relations (from the set $\mathcal{R}el$) between the features values may be discovered using the *DRAR* method [15].

The measured set of features is afterwards extended with s ($s \geq 1$) new features, numbered as $m+1, m+2, \ldots, m+s$. After extension, the objects' feature vectors become $r_i^{ext} = (r_{i1}, \ldots, r_{im}, r_{i,m+1}, \ldots r_{i,m+s})$, $1 \leq i \leq n$. The set of extended instances is denoted by $R^{ext} = \{r_1^{ext}, r_2^{ext}, \ldots, r_n^{ext}\}$.

Considering certain minimum support and confidence thresholds (denoted by s_{min} and c_{min}), we want to analyze the problem of mining interesting relational association rules within

the data set R^{ext}, i.e. after object extension, and starting from the set of rules discovered in the data set R before the feature set extension. We aim at obtaining a better time performance with respect to the mining from scratch process. We denote in the following by $\mathcal{RAR}$ the set of interesting relational association rules having a minimum support and confidence within the data set R, and by $\mathcal{RAR}^{ext}$ the set of interesting relational association rules having a minimum support and confidence within the extended data set R^{ext}.

Certainly, the newly arrived features can generate new relational association rules. The new set of rules $\mathcal{RAR}^{ext}$ could be of course obtained by applying the *DRAR* method from scratch on the set of extended objects. But we try to avoid this process and replace it with one less expensive, but preserving the completeness of the rule generation process. More specifically, we will propose a method called *ARARM* (*Adaptive Relational Association Rule Mining*), which starts from the set $\mathcal{RAR}$ of rules mined from the data set before feature extension and adapts it (considering the newly added features) in order to obtain the set of interesting relational association rules within the set of extended objects R^{ext}. Definitely, through the adaptive process, we want to preserve the completeness of the *DRAR* method.

Let us denote by l the maximum length of the rules from the set RAR and by RAR_k $(1 \leq k \leq l)$ the set of interesting relational association rules of length k discovered in the data set R (before the feature set extension). Obviously, $\mathcal{RAR} = \bigcup_{k=1}^{l} RAR_k$.

In the following we give a brief description of the idea of discovering the set $\mathcal{RAR}^{ext}$ through adapting the set $\mathcal{RAR}$ of rules mined in the data set R before feature extension.

The *ARARM* algorithm identifies the interesting relational association rules using an iterative process that consists in length-level generation of rules, followed by the verification of the candidates for minimum support and confidence compliance. *ARARM* performs multiple passes over the data set R^{ext}. In the first pass, it calculates the support and confidence of the 2-length rules and determines which of them are interesting, i.e. verify the minimum support and confidence requirements. Every subsequent pass over the data consists of two phases. The k-length $(k \geq 2)$ rules from R^{ext} will certainly contain the k-length rules from $\mathcal{RAR}$ (the interesting rules discovered in the data set before extension) - if such rules exist. But, there is another possibility to obtain a k-length rule in the extended data set, through generating a candidate rule through joining two $k-1$-length rules from $\mathcal{RAR}^{ext}$ (generated at the previous iteration). During the second phase, a scan over R^{ext} is performed in order to compute the actual support and confidence of the candidate rules generated as described above. At the end of this step, the algorithm keeps the rules that are deemed interesting (have minimum support and satisfy the confidence requirements), which will be used in the next iteration. The process stops when no new interesting rules were found in the latest iteration.

At a certain iteration performed by the *ARARM* algorithm, the candidate generation process (denoted by *GenCandidates* in the algorithm from Figure 4.3) is essentially the same as the candidates generation process of the *DRAR* method [9] (the particularities of generating the candidates in the *ARARM* algorithm will be discussed afterwards). More specifically, for joining two $k-1$ length rules in order to obtain a k-length candidate rule, there are four possibilities which are illustrated in Figure 4.2. Similarly to the proof presented in [9] it may be proven that the candidate generation process ensures the correctness and completeness of the *ARARM* algorithm.

In the following we denote by μ^{-1} the *inverse* of the relation denoted by μ.

Regarding the binary relations that may be defined between the attributes domains, we mention that we do not assume any particular property (such that the transitivity property), both *DRAR* and *ARARM* are working with general relationships between the attributes domains. Supposing we have three attributes a_1, a_2 and a_3 and the set of relations $\{<, >, =\}$, the relational association rules $r_1 = (a_1 < a_2 < a_3)$ and $r_2 = (a_1 < a_3 > a_2)$ are viewed as

$$
\begin{aligned}
&rule_1 \equiv (a^1\mu^1 a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}),\\
&rule_2 \equiv (a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}\mu^2 a^2), \qquad (1)\\
&\Rightarrow c \equiv (a^1\mu^1 a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}\mu^2 a^2),\\
&\qquad\qquad \text{or}\\
&rule_1 \equiv (a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}\mu^1 a^1),\\
&rule_2 \equiv (a^2\mu^2 a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}), \qquad (2)\\
&\Rightarrow c \equiv (a^2\mu^2 a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}\mu^1 a^1),\\
&\qquad\qquad \text{or}\\
&rule_1 \equiv (a^1\mu^1 a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}),\\
&rule_2 \equiv (a^2\mu^2 a_{i_{k-2}}\mu_{k-3}^{-1}\ldots a_{i_2}\mu_1^{-1}a_{i_1}), \qquad (3)\\
&\Rightarrow c \equiv (a^1\mu^1 a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}(\mu^2)^{-1}a^2),\\
&\qquad\qquad \text{or}\\
&rule_1 \equiv (a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}\mu^1 a^1),\\
&rule_2 \equiv (a_{i_{k-2}}\mu_{k-3}^{-1}\ldots a_{i_2}{\mu_1}^{-1}a_{i_1}\mu^2 a^2), \qquad (4)\\
&\Rightarrow c \equiv (a^2(\mu^2)^{-1}a_{i_1}\mu_1 a_{i_2}\ldots\mu_{k-3}a_{i_{k-2}}\mu^1 a^1).
\end{aligned}
$$

Figure 4.2: The candidate generation process in the *ARARM* algorithm

distinct rules (otherwise, if the transitivity of the relation $<$ would be considered, then r_2 would be seen as a generalization of r_1). Furthermore, there are no constraints placed on the relationships. Thus, the rules $a_1 = a_2 < a_3$ and $a_2 = a_1 < a_3$ are considered by *ARARM* distinct rules (even if they are equivalent since $=$ is a symmetric relation). Obviously, if one would need to deal with relations having particular properties (e.g *transitive* or *symmetric*), after the set of relational association rules are discovered by *ARARM* (or *DRAR*), a filtering step may be added. This step may be used to remove equivalent rules, like those from the examples above.

Figure 4.3 gives the *ARARM* algorithm.

In the algorithm presented in Figure 4.3, by $\bigoplus$ we have denoted a special "add" operation. If r is a relational association rule and L is a set of relational association rules, by $r \bigoplus L$ we refer to the set of relational association rules obtained by adding r to L if L does not contain the "mirror" rule of r. If L contains the "mirror" of r, then $r \bigoplus L$ equals to L. We mention that by the "mirror" of a relational association rule $r \equiv (a_1\mu_1 a_2\mu_2\ldots\mu_{n-1}a_n)$ we refer to the rule $r^{-1} \equiv (a_n\mu_{n-1}^{-1}\ldots\mu_2^{-1}a_2\mu_1^{-1}a_1)$ (e.g the "mirror" of the rule $a_1 < a_2 > a_3$ is the rule $a_3 < a_2 > a_1$). The special operation $\bigoplus$ we have used in constructing the set of relational association rules assures that the resulting set does not contain duplicate rules (i.e rules together with their mirrors). For optimization reasons, when applying the $\bigoplus$ operation, the verification if the "mirror" of rule r is in L is skipped, when appropriate. For example, the candidate generation process may generate a "mirror" of a rule only if the set $\mathcal{R}el$ of relations used in the mining process contains at least a rule μ together with its inverse μ^{-1}. Only in this case, when adding a rule r with the $\bigoplus$ operation into a set L we must verify if the "mirror" of r is in L.

The most important step in the *ARARM* algorithm is the candidate generation process (denoted by the *GenCandidates* function), which is also computationally expensive. This function has as a parameter a set *Rules* of k-length relational association rules and returns a set of k+1 length relational association rules generated from *Rules* through the join operations depicted in Figure 4.2. The main idea of the candidate generation process is the following. All distinct combinations of two rules (r_1, r_2) from the set *Rules* are considered. If r_1 and r_2 match for join (in one of the four cases indicated in Figure 4.2, the rule r_{join} obtained by joining r_1 and r_2 is constructed and is added to the resulting set of rules. Obviously, since r_1 and r_2 are k-length rules, r_{join} will be a $k+1$-length rule. The *GenCandidates* function is described in Figure 4.4.

Algorithm *ARARM* **is:**
Input: - the set R of m-dimensional entities
- the set $\mathcal{R}el$ of relations used in the mining process
- the extended data set R^{ext} of $m+s$ dimensional entities
- the minimum confidence (c_{min}) and support (s_{min}) thresholds
- the set $\mathcal{RAR}$ of interesting relational association rules from the data set R
Output: - the set $\mathcal{RAR}^{ext}$ of all interesting relational association rules that hold over R^{ext}

$RAR_2 \leftarrow$ the set of 2 length rules from $\mathcal{RAR}$;
$C \leftarrow RAR_2 \cup \{ (a_{i_1}\mu_1 a_{i_2}) \mid a_{i_1}, a_{i_2} \in A,\ i_1 = 1 \ldots m+s, i_2 = m+1 \ldots m+s,\ i_1 < i_2\ \mu_1 \in \mathcal{R}el \}$;
Scan R^{ext} and compute the support and confidence of candidates in C;
Keep the interesting rules from $C \Rightarrow AdaptiveRules$;
$k \leftarrow 2$;
$termination \leftarrow false$;
$\mathcal{RAR}^{ext} \leftarrow AdaptiveRules$
While $(\neg termination)$ do
  $C \leftarrow GenCandidates(AdaptiveRules)$;
  Scan R^{ext} and compute the support and confidence of candidates in C;
  Keep the interesting rules from $C \Rightarrow L$;
  $RAR_k \leftarrow$ the set of k length rules from $\mathcal{RAR}$;
  $AdaptiveRules \leftarrow L \cup RAR_k$
  If $AdaptiveRules = \emptyset$ then
    $termination \leftarrow true$
  Else
    $k \leftarrow k+1$;
    For each $r \in AdaptiveRules$ do
      $\mathcal{RAR}^{ext} \leftarrow r \bigoplus \mathcal{RAR}^{ext}$
    EndFor
  EndIf
End;
End *ARARM*

Figure 4.3: The *ARARM* algorithm

It has to be stated that running the *ARARM* method with $m = 0$ provides the set of interesting relational association rules discovered in the input data set of s-dimensional entities. Thus, this running is equivalent with applying the *DRAR* method on the data set of s-dimensional entities. It can be seen in the *GenCandidates* function (Figure 4.4) that two rules are joined during the adaptive candidate generation process only if at least one rule has at least an attribute from the additional attributes, which are present only in the enlarged attribute set. Obviously, it is not necessary to join rules which contain only attributes from the original attribute set, since the joint rules are already known (these rule are in the set $\mathcal{RAR}$ of relational rules discovered in the set of m-dimensional entities). This way, when generating the k-length relational association rules from the extended data set of $m+s$ dimensional entities, the candidate generation process (expressed by the *GenCandidates* function) is not applied on the set of $k-1$ length rules from the set off interesting rules extracted from the data set of m-dimensional entities. Thus, unlike in the *DRAR* algorithm applied from scratch on the $m+s$ dimensional entities, the join operations between the $k-1$ length rules from the data set of entities before the attribute set extension are skipped.

Function *GenCandidates* (*Rules*) **is:**
Input: a set *Rules* of k-length relational association rules
Output: returns a set *NewRules* of $k + 1$ length relational association rules obtained by joining
the rules from *Rules*

NewRules $\leftarrow \emptyset$
$n \leftarrow$ number of rules from the set *Rules*
For $i \leftarrow 1$ to $n - 1$ do
  For $j \leftarrow i + 1$ to n do
    $ri \leftarrow$ the i-th rule from *Rules*
    $rj \leftarrow$ the j-th rule from *Rules*
      If ri or rj contain at least a newly added attribute (from the set $\{a_{m+1}, a_{m+2}, \ldots, a_{m+s}\}$)
        If ri matches for join with ri in one of the cases (1)-(4) from Figure 4.2 then
          $rjoin \leftarrow$ the rule obtained by joining ri and rj
          $NewRules \leftarrow rjoin \bigoplus NewRules$
        EndIf
      EndIf
  EndFor
EndFor
return *NewRules*
End*GenCandidates*

Figure 4.4: The *GenCandidates* function

The time savings in the *ARARM* execution time come from the time reduction of the candidate generation process as well as from an reduced number of support and confidence computations (a detailed analysis will be given in Section 4.3.2). Obviously, as seen from Figure 4.4, the step of computing the support and confidence for the rules from the set $\mathcal{RAR}$ is skipped, since for these rules we already have their support and confidence. Certainly, the reduction in the execution time of the *ARARM* increases with the increase of the set $\mathcal{RAR}$. This usually happens when decreasing the number s of added attributes. A smaller number of added attribute means a larger set of already known rules and this implies a smaller number of rules generated by *GenCandidates* function, as well as less time for support and confidence computations.

In the current implementation of the *ARARM* method, we did not deal with optimizing the rules matching step in the candidate generation process described in Figure 4.2. Since the same implementation for this step is used both in the *ARARM* and *DRAR* implementation, a more efficient implementation of it will lead to lower running times for both methods. Still, it will not significantly impact the improvement in running time (in %) of *ARARM* with respect to *DRAR* (what we are interested in). We will further investigate optimizations of the rules matching step in order to increase the efficiency of the candidate generation process (e.g using efficient data structures such as *hash tables* or *canonical forms*).

The current implementation of the *ARARM* algorithm provides the functionality of discovering all interesting relational association rules of any length, as well as the functionality of finding all maximal interesting relational association rules of any length. Moreover, we have parameterized the set of features used in the mining process, thus it is very easy to remove certain features from the mining process.

4.3 Experimental evaluation

In order to show the effectiveness of the adaptive relational association rule mining method that was introduced in Section 4.2, we consider in the following two case studies that will be further described. All the experiments presented in this section were carried out on a PC with an Intel Core i7 Processor at 1.87 GHz, with 8 GB of RAM.

4.3.1 Experiments

In the experiments we have performed for discovering the interesting relational association rules within the data sets, we have initially considered m attributes characterizing the instances within the data set and afterwards the set of features was extended with s attributes. The experiments were made considering different values for the minimum support and confidence thresholds (s_{min} and c_{min}) and different type of relational association rules, i.e *maximal* rules vs. *all* rules (as indicated in Section 4.1). For each performed experiment, the set of interesting relational association rules on the $m + s$ dimensional instances were obtained in two ways:

1. by applying the $DRAR$ method from scratch on the data set after the feature set extension (containing all $m + s$ features).
2. by adapting (through the $ARARM$ algorithm) the rules obtained on the data set before the feature set extension (containing m features).

We mention that the same set of interesting relational association rules is discovered in data, independent to the way the rules were generated (1. or 2.), but, obviously, we are expecting the running time of the adaptive algorithm to be lower than the running time of the $DRAR$ method applied from scratch.

For all experiments that will be presented in the following, we aim at emphasizing that $ARARM$ has a lower running time that $DRAR$ applied from scratch on the set of $m + s$ dimensional entities. The adaptive method we propose in this paper uses intermediary interesting relational association rules in order to obtain subsequent ones, thus avoiding the need to start the mining algorithm from the very beginning each time. Due to the temporal evolution of the attribute set, at a given time t when there are available the values of the first m attributes for the instances, the set $\mathcal{RAR}$ of rules discovered in the data set of m-dimensional entities are extracted. Assuming that at time $t + 1$ new values for the attributes are procured, the $ARARM$ method uses the existing rules from $\mathcal{RAR}$ and efficiently obtains new accurate relational association rules that include the latest available data. Thus, the running time of the $ARARM$ algorithm (Figure 4.3) is not influenced by the time needed to mine the rules on the data set before the attribute set extension.

For implementation, we have developed a Java API which allows the discovery, in an adaptive manner (using the $ARARM$ algorithm introduced in Section 4.2), of interesting relational association rules which occur within a data set. The API can be used to uncover (adaptive) relational association rules in various data sets, independent of the objects (instances) within the data set, the type of features characterizing the instances, as well as the relations that are defined between the features.

The experiments that will be presented in Section 4.3.2 were performed using the $ARARM$ API.

In all the data sets which will be further considered for evaluation, we are dealing with numerical attributes. Before applying the relational association mining process, we first normalize the data using the *min-max* normalization method. For all the experiments, two possible relations between the attributes values are considered in the relational association rule mining process: $\mathcal{R}el = \{\leq, >\}$. It has to be noted that for the first human skeletal

remains data set which contains missing attribute values, the minimum support threshold s_{min} used in the mining process is less than 1, for all the other data sets s_{min} is set to 1.

Regardless of the threshold c_{min} value, the *ARARM* method adaptively mines the set of interesting relational association rules having a minimum confidence of c_{min} within a data set, when new attributes are added to the data set. Even if the minimum confidence threshold value is irrelevant, in our experiments we have selected the value for c_{min} such that the number of discovered rules in data to be large enough. This way, with a reasonable number of rules, the time reduction of the *ARARM* algorithm with respect to *DRAR* may be better illustrated. Heuristics for selecting appropriate values for the threshold c_{min} would be useful when applying the *ARARM* algorithm in concrete data mining tasks.

4.3.2 Synthetic data

In the following we aim at testing the performance of the *ARARM* algorithm on larger data sets. We have to mention that the time performance of the *ARARM* algorithm for adaptive relational association rule mining depend not only on the dimensionality of the data set (number of instances and attributes), but mainly on the number of interesting relational associations rules which were discovered in data. Since not the data set is relevant in our study, but its dimensionality, we have considered two synthetic data sets which were obtained by merging publicly available data sets from the NASA repository [41] which are used in the software defect prediction literature.

Our goal is to investigate the effectiveness of the *ARARM* algorithm when (1) the number of attributes characterizing the instances within the data sets is large (the first synthetic data) and when (2) the number of instances within the data set is large. Obviously, in both cases, the number of relational association rules mined in data may be very large and this also depends on the minimum confidence threshold.

The first synthetic data set considered in our experiments consists of 125 instances characterized by 116 attributes. The experiments were performed by applying the adaptive *ARARM* algorithm and the *DRAR* algorithm applied from scratch after the feature set extension. Different values for the initial number of attributes (m) and the minimum confidence threshold c_{min} are considered for discovering all the relational association rules within the data set. The obtained results are presented in Tables 4.4 and 4.5 where the improvement achieved through the adaptive process is marked with bold. In these tables, n_m denotes the number of rules discovered in the data set before the attribute set extension (containing m attributes) and n_{m+s} represents the number of rules mined in the data set after the attribute set extension (containing $m + s$ attributes).

From Tables 4.4 and 4.5 one can observe that, for all the experiments, the *ARARM* method has a lower running time than the *DRAR* method applied from scratch. Even if the number of relational association rules mined in the data set is large, our adaptive method obtains an average improvement in running time of **21**%, which indicates the effectiveness of the *ARARM* method.

The second synthetic data set used for evaluating the *ARARM* method consists of 2366 instances characterized by 38 attributes. The experiments were performed by applying the adaptive *ARARM* algorithm and the *DRAR* algorithm applied from scratch after the feature set extension using a minimum confidence threshold of 0.82. A number of 38811 interesting relational association rules were discovered in data. Different values for the initial number of attributes (m) are considered in our experiments. Table 4.6 presents the performed experiments and the obtained results. For all the performed experiments, there is an improvement achieved through the adaptive process and it is marked with bold. In this table n_m denotes the number of interesting relational association rules discovered in the data set before the feature set extension (containing m features) and n_{m+s} represents the rules obtained in the extended data set (with $m + s$ features).

Experiment	c_{min}	**No. of attributes** (m)	**No. of added attributes** (s)	**No. of rules** n_m	**No. of rules** n_{m+s}	**Time from scratch (ms)**	**Time adaptive (ms)**	**Improvement in running time (%)**
1	0.96	26	90	23	13437	3350	3000	**10.45**
2	0.96	30	86	40	13437	3350	2908	**13.19**
3	0.96	34	82	64	13437	3350	2928	**12.6**
4	0.96	38	78	208	13437	3350	2918	**12.9**
5	0.96	42	74	310	13437	3350	2940	**12.24**
6	0.96	46	70	366	13437	3350	2932	**12.48**
7	0.96	50	66	366	13437	3350	2924	**12.72**
8	0.96	54	62	369	13437	3350	2897	**13.52**
9	0.96	58	58	399	13437	3350	2907	**13.22**
10	0.96	102	14	3688	13437	3350	2684	**19.88**
11	0.96	103	13	4050	13437	3350	2615	**21.94**
12	0.96	104	12	4642	13437	3350	2518	**24.84**
13	0.96	105	11	4646	13437	3350	2472	**26.21**
14	0.96	106	10	4815	13437	3350	2466	**26.39**
15	0.96	107	9	4920	13437	3350	2457	**26.66**
16	0.96	108	8	5140	13437	3350	2408	**28.12**
17	0.96	109	7	5150	13437	3350	2387	**28.75**
18	0.96	110	6	5412	13437	3350	2295	**31.49**
19	0.96	111	5	6772	13437	3350	2156	**35.64**
20	0.96	112	4	7124	13437	3350	2014	**39.88**
21	0.96	113	3	9218	13437	3350	1517	**54.72**
22	0.96	114	2	11280	13437	3350	884	**73.61**
23	0.96	115	1	11358	13437	3350	880	**73.73**
24	0.94	26	90	41	65887	57173	56195	**1.71**
25	0.94	30	86	72	65887	57173	55978	**2.09**
26	0.94	34	82	114	65887	57173	56484	**1.21**
27	0.94	38	78	389	65887	57173	56505	**1.17**
28	0.94	42	74	578	65887	57173	56469	**1.23**
29	0.94	46	70	719	65887	57173	56523	**1.14**
30	0.94	50	66	723	65887	57173	56595	**1.01**
31	0.94	54	62	808	65887	57173	56634	**0.94**
32	0.94	58	58	1077	65887	57173	56596	**1.01**
33	0.94	102	14	16501	65887	57173	51653	**9.65**
34	0.94	103	13	18226	65887	57173	50474	**11.72**
35	0.94	104	12	20835	65887	57173	49335	**13.71**
36	0.94	105	11	20936	65887	57173	49547	**13.34**
37	0.94	106	10	21917	65887	57173	48986	**14.32**
38	0.94	107	9	22652	65887	57173	48236	**15.63**
39	0.94	108	8	24010	65887	57173	47319	**17.24**
40	0.94	109	7	24082	65887	57173	47101	**17.62**
41	0.94	110	6	25067	65887	57173	46478	**18.71**
42	0.94	111	5	31230	65887	57173	41997	**26.54**
43	0.94	112	4	33094	65887	57173	40225	**29.64**
44	0.94	113	3	43893	65887	57173	29575	**48.27**
45	0.94	114	2	54613	65887	57173	18465	**67.7**
46	0.94	115	1	55106	65887	57173	18111	**68.32**

Table 4.4: Results for the first synthetic data set for $s_{min} = 1$

Experiment	c_{min}	**No. of attributes** (m)	**No. of added attributes** (s)	**No. of rules** n_m	**No. of rules** n_{m+s}	**Time from scratch (ms)**	**Time adaptive (ms)**	**Improvement in running time (%)**
47	0.935	26	90	59	147042	292652	291665	**0.34**
48	0.935	30	86	95	147042	292652	292531	**0.04**
49	0.935	34	82	159	147042	292652	292556	**0.03**
50	0.935	38	78	511	147042	292652	292021	**0.22**
51	0.935	42	74	757	147042	292652	288363	**1.47**
52	0.935	46	70	980	147042	292652	290428	**0.76**
53	0.935	50	66	1020	147042	292652	288040	**1.58**
54	0.935	54	62	1251	147042	292652	289367	**1.12**
55	0.935	58	58	1896	147042	292652	285637	**2.4**
56	0.935	102	14	38434	147042	292652	259216	**11.43**
57	0.935	103	13	42142	147042	292652	254846	**12.92**
58	0.935	104	12	47967	147042	292652	247048	**15.58**
59	0.935	105	11	48179	147042	292652	246894	**15.64**
60	0.935	106	10	50424	147042	292652	245175	**16.22**
61	0.935	107	9	52070	147042	292652	240410	**17.85**
62	0.935	108	8	55007	147042	292652	234799	**19.77**
63	0.935	109	7	55217	147042	292652	234950	**19.72**
64	0.935	110	6	57679	147042	292652	230088	**21.38**
65	0.935	111	5	70601	147042	292652	209675	**28.35**
66	0.935	112	4	75478	147042	292652	201583	**31.12**
67	0.935	113	3	99043	147042	292652	147498	**49.6**
68	0.935	114	2	122452	147042	292652	89480	**69.42**
69	0.935	115	1	123500	147042	292652	86725	**70.37**

Table 4.5: Results for the first synthetic data set for $s_{min} = 1$ and $c_{min} = 0.935$

Table 4.6 reveals that, for all the experiments performed on the second synthetic data, the *ARARM* method is more performant (from the running time point of view) than the *DRAR* method applied from scratch.

For a better analysis of the *ARARM* performance on the second synthetic data set, the running time (both for the *ARARM* and *DRAR* methods) was decomposed in: time for support and confidence computations and time for the candidate generation process. The obtained values are indicated in Table 4.7. $nsc_{scratch}$ and $nsc_{adaptive}$ represent the number of support and confidence computations performed by the *DRAR* method applied from scratch and by the *ARARM* method, respectively. By $tsc_{scratch}$ and $tsc_{adaptive}$ we denote the time needed for the support and confidence computations performed by the *DRAR* and *ARARM* methods. The time needed for the candidate generation process is represented by $tcg_{scratch}$ (for *DRAR*) and by $tcg_{adaptive}$ (for *ARARM*).

Analyzing the results indicated in Table 4.7 we observe that the time for support and confidence computations performed by *ARARM* decreases as the number rules found on the data set of m-dimensional instances is large enough and increases. The results reveal that the performance of *ARARM* (with respect to *DRAR*) is significant when the number n_m of rules is large. Figure 4.5 illustrates, for each experiment performed on the second synthetic data, the percentage of execution time used by *ARARM* and *DRAR* for confidence and support computations, as well as for the candidate generation process. We note that as the number n_m of rules increases, the reduction in execution time of the adaptive method given by the support and confidence computations becomes greater.

Experiment	No. of attributes (m)	No. of added attributes (s)	n_m	n_{m+s}	Time from scratch (ms)	Time adaptive (ms)	Improvement in running time (%)
1	15	23	76	38811	36314	34435	**5.17**
2	16	22	93	38811	36314	34361	**5.38**
3	17	21	102	38811	36314	34544	**4.87**
4	18	20	201	38811	36314	34635	**4.62**
5	19	19	290	38811	36314	34781	**4.22**
6	20	18	381	38811	36314	34887	**3.93**
7	21	17	580	38811	36314	34770	**4.25**
8	22	16	905	38811	36314	34527	**4.92**
9	23	15	1314	38811	36314	34266	**5.64**
10	24	14	1813	38811	36314	34167	**5.91**
11	25	13	2988	38811	36314	33772	**7**
12	26	12	4041	38811	36314	32982	**9.18**
13	27	11	4132	38811	36314	32891	**9.43**
14	28	10	4215	38811	36314	32993	**9.15**
15	29	9	6476	38811	36314	31701	**12.7**
16	30	8	7279	38811	36314	31101	**14.36**
17	31	7	10676	38811	36314	29154	**19.72**
18	32	6	14077	38811	36314	26928	**25.85**
19	33	5	16711	38811	36314	25071	**30.96**
20	34	4	25882	38811	36314	16888	**53.49**
21	35	3	32937	38811	36314	8740	**75.93**
22	36	2	32937	38811	36314	8740	**75.93**
23	37	1	32963	38811	36314	8702	**76.04**

Table 4.6: Results for the second synthetic data set for $s_{min} = 1$ and $c_{min} = 0.82$.

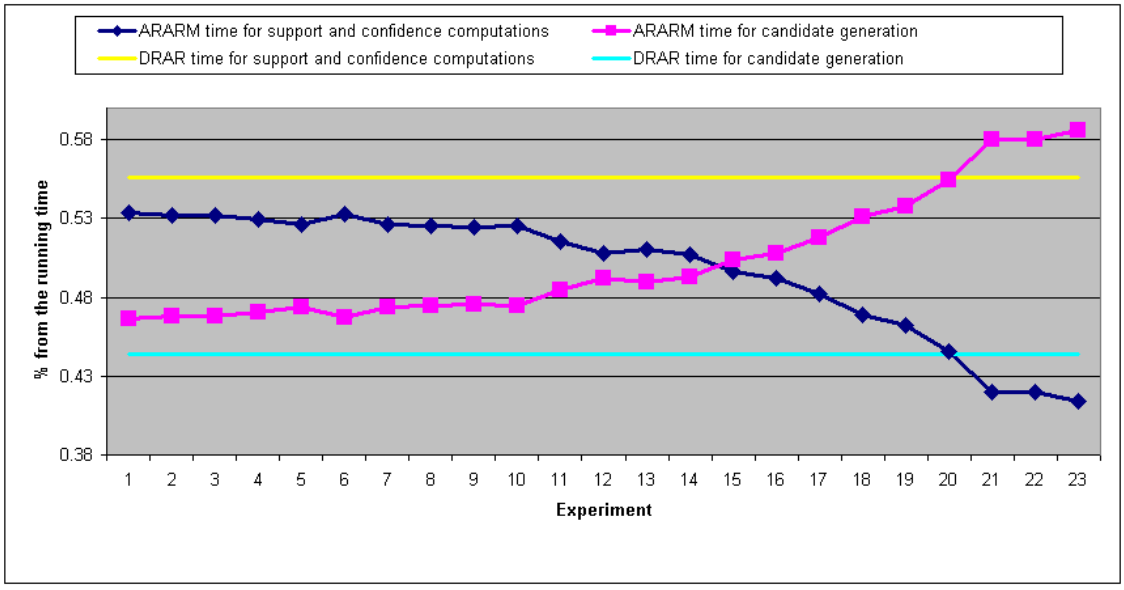

Figure 4.5: The candidate generation process influence on the *ARARM* running time for the second synthetic data set.

Experiment	$nsc_{scratch}$	$nsc_{adaptive}$	$tsc_{scratch}$	$tsc_{adaptive}$	$tcg_{scratch}$	$tcg_{adaptive}$
1	49744	49458	20204	18370	16110	16065
2	49744	49412	20204	18277	16110	16084
3	49744	49311	20204	18367	16110	16177
4	49744	49190	20204	18339	16110	16296
5	49744	49052	20204	18313	16110	16468
6	49744	48920	20204	18590	16110	16297
7	49744	48640	20204	18297	16110	16473
8	49744	48172	20204	18143	16110	16384
9	49744	47615	20204	17969	16110	16297
10	49744	46980	20204	17943	16110	16224
11	49744	45518	20204	17406	16110	16366
12	49744	44109	20204	16757	16110	16225
13	49744	43945	20204	16784	16110	16107
14	49744	43771	20204	16737	16110	16256
15	49744	41150	20204	15724	16110	15977
16	49744	40205	20204	15312	16110	15789
17	49744	36588	20204	14055	16110	15099
18	49744	32983	20204	12635	16110	14293
19	49744	30042	20204	11588	16110	13483
20	49744	19741	20204	7524	16110	9364
21	49744	9652	20204	3667	16110	5073
22	49744	9652	20204	3667	16110	5073
23	49744	9546	20204	3601	16110	5101

Table 4.7: Additional results for the second synthetic data set.

4.4 Discussion

In the following we aim to analyze the method proposed in this paper by emphasizing its advantages and drawbacks, as well as comparing *ARARM* method with other related approaches existing in the data mining literature.

4.4.1 Analysis of the *ARARM* method

Experiments were conducted in Section 4.3 in order to test the usefulness of *ARARM* method on five data sets: a gene expression data set, two human skeletal remains data sets and two synthetic data sets.

There are no existing approaches in the literature that deal with the adaptive *relational* association rule mining problem. Thus, we have conducted our evaluations towards comparing the running time of the adaptive method with the running time of the non-adaptive ones. We have focused on highlighting that the *ARARM* method provides the results faster than the *DRAR* method applied from scratch.

In order to conclude about the efficiency of the adaptive method against the non-adaptive one, we depict in Figures 4.6 and 4.7 (for each experiment we have performed on the case studies considered for evaluation in Section 4.3) the reduction (in percentage) of the running time of *ARARM* with respect to the running time of *DRAR*.

We notice that, for all the experiments represented in Figures 4.6 and 4.7, the running time for *ARARM* is less than the running time of *DRAR* applied from scratch. Larger the value on the y-axis is, greater is the reduction in the execution time of the *ARARM* method. Thus, the time needed to adaptively discover the interesting relation association rules is less

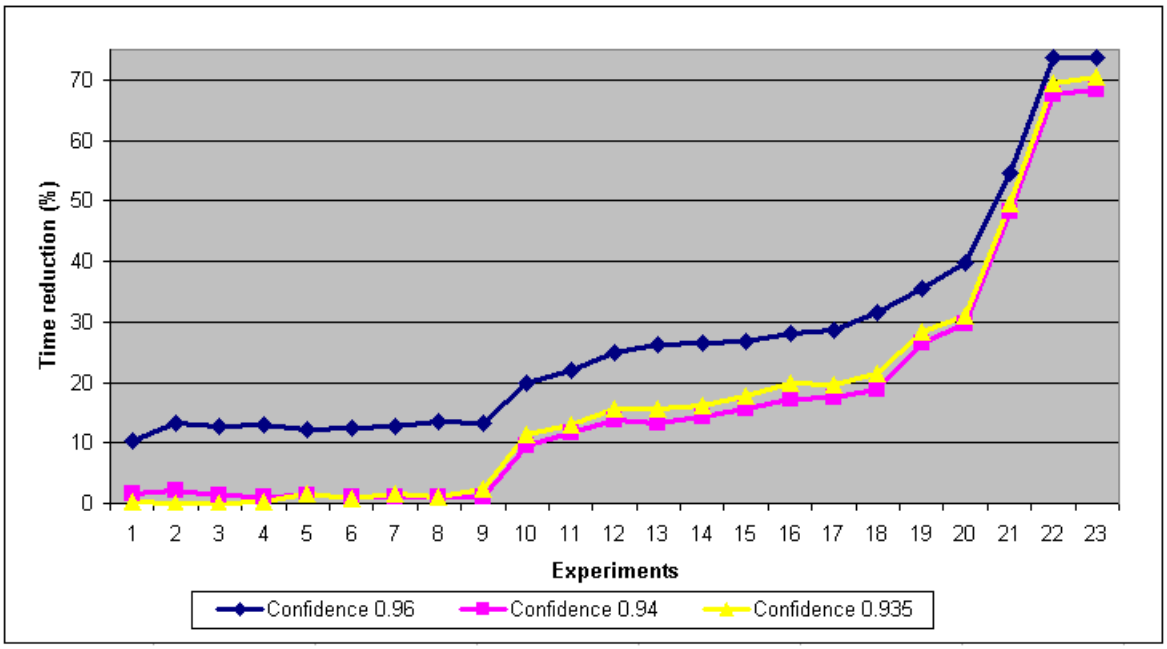

Figure 4.6: Results for the first synthetic data

than the time needed to obtain the rules non-adaptively, i.e by running from scratch the algorithm for finding the rules, and this emphasizes the effectiveness of our proposal.

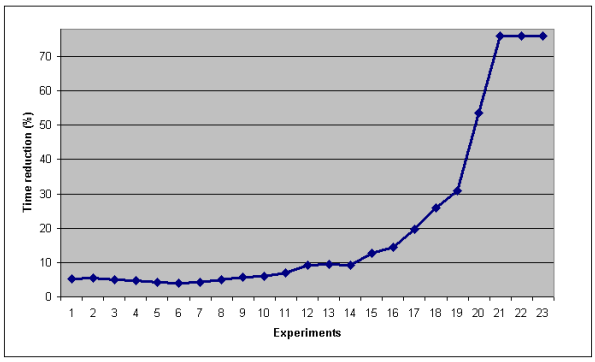

Figure 4.7: Results for the second synthetic data

To conclude about the performance of the *ARARM* method introduced in this paper, additional experiments were performed. The experiments were performed on an open source data set from the NASA repository [41] which was used in the literature for software defect prediction. The $PC1$ data set is built for functions from a flight software for earth orbiting satellite, written in C. Experiments were performed on the subset of $PC1$ consisting of 644 instances which are non-defects. On this data set, a total number of 40947 relational association rules having a minimum confidence of 0.9 were identified.

Experiments with different number of instances (which lead to different number of relational association rules) are performed on the $PC1$ non-defects data set (with a minimum confidence threshold of 0.9, $m = 35$ and $s = 2$). Figure 4.8 depicts how the running time of *ARARM* decreases with respect to the running time of *DRAR* regarding the number of considered instances. The performance of *ARARM* in comparison with *DRAR* considering

the number of discovered relational association rules is illustrated in Figure 4.9.

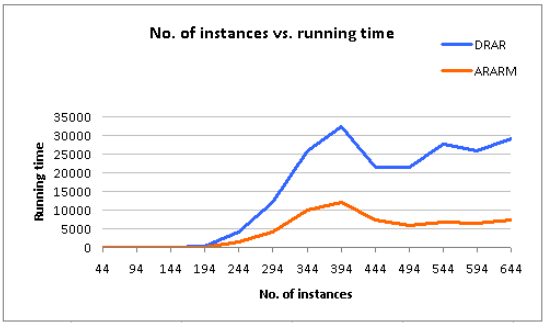

Figure 4.8: *ARARM* performance - number of instances vs. running time for the *PC*1 data set.

Figure 4.9: *ARARM* performance - number of rules vs. running time for the *PC*1 data set.

From Figures 4.8 and 4.9 one can conclude that the adaptive relational association rule method is more effective (in running time performance) than the the non-adaptive one.

Table 4.8 indicates, for each data set we have considered in our experimental evaluation, the reduction in the average running time of *ARARM* with respect to the average running time of *DRAR* applied from scratch.

In order to perform a statistical analysis the obtained results, for each case study considered for evaluation, we computed a 95% Confidence Interval (CI) [5] for the average of running time reductions obtained by applying the *ARARM* method.

Data set	**Average of number of rule**	**Average of running time for *ARARM* (ms)**	**Average of running times for *DRAR* from scratch (ms)**	***ARARM* running time reductions (%)**
First synthetic data	75455	112907.54	137110.59	21.75 ± 4.7
Second synthetic data	38811	33161	36314	**20.38±2.46**
PC1 data set	25637	4767	15389	**71.18±3.58**

Table 4.8: Average running time reduction for the performed experiments. A 95% CI is used for the values on the fifth column.

The reduction in running time obtained through the adaptive method is natural. It comes from the fact that the *ARARM* method starts from the set of relational association rules discovered in the data set of m-dimensional instances and adapts it (using the adaptive

algorithm from Figure 4.3) considering the newly added s attributes. This, obviously, is more effective, since we are starting from the set of relational association rules identified in the data set of m-dimensional instances and adapt it, instead of running the relational association rule mining method from scratch on the $m + s$ dimensional instances. It is very likely that the adaptation time decreases with decreasing of the number of added attributes. Clearly, the reduction in running time of $ARARM$ with respect to $DRAR$ increases as the number n of rules from the data set before the feature set extension is larger. When n becomes close to 0, then the $ARARM$ running time becomes closer to the running time of $DRAR$.

From Table 4.8 we notice an average of 38% for the $ARARM$ running time reduction with respect to the running time from scratch. We note that, when the number of rules is very large (e.g **147042** in Table 4.6) the reduction in running time obtained through the adaptive mining algorithm (an average of **30**% when adding at least 14 new attributes) is significant, since it may lead to saving hours (even days) of running.

Considering the experiments from Section 4.3 and the comparisons shown above, we can conclude that $ARARM$ is more efficient, from the running time point of view, than $DRAR$ applied from scratch. Still, there may be situations in which the running time of $ARARM$ is not significantly smaller than the running time of $DRAR$ applied from scratch. These situations occur when a small number of interesting relational association rules occur within the data set before the feature set extension. As shown by the experimental results, this usually happens when the number s of initial attributes is small.

An extension of the $ARARM$ method to a *fuzzy* approach [11, 12] would be relevant in practical applications, where we are dealing we noisy data. We are currently working on extending the concept *relational* association rules towards *fuzzy relational* association rules and developing a method (similar to $DRAR$) for mining interesting fuzzy relational association rules in data sets. Afterwards, it would be possible to investigate the adaptive fuzzy relational association rule mining. The fuzzy extension of the $ARAM$ method will be one of our further research.

The main goal of the proposed $ARARM$ algorithm is to adaptively mine relational association rules within a data set. The $ARARM$ method is complete and it reduces the time needed to discover the rules from scratch, when the attribute set increases. Certainly, it would be of great interest to analyze the relational association rules which were discovered in data (as shown in Section 4.1). Further work will be carried out in order to investigate the usefulness of the mined information [12, 13]. We plan to use the $ARARM$ method for classification tasks: in *software engineering* for predicting defective software entities (see the PC1 data set).

4.4.2 Comparison to related work

The adaptive relational association rule mining approach introduced in Section 4.2 is innovative, since there are no existing similar approaches in the data mining literature. Existing approaches deal with non-relational associational rules and they are adaptive with respect to other aspects, like data dependent parameters. Thus, we can not compare our approach with the existing ones, since the perspectives are different. Still, we will present in the following several existing data mining methods that are somehow similar with our approach.

Sarda and Srinivas [45] present an adaptive algorithm for incremental mining of association rules, which is able to decide based on the type of increment – similar or significantly different – whether to scan the original database for updating the rules obtained in earlier mining processes. This approach deals with incremental rule mining, in which new instances are added to the data set, unlike ours in which new features are added to the existing objects.

Het et al propose in [28] an adaptive fuzzy association rule mining approach for decision support. Their algorithm, called $FARM - DS$, builds a decision support system for binary classification problems in biomedical applications, which besides the class label also returns

the rules fired for an unseen sample. The parameters of the algorithm are adaptive in the sense that they are data-dependent and optimized by cross validation. Our *ARARM* algorithm is essentially different from the $FARM - DS$ algorithm, since the only *adaptive* viewpoint in [28] is related to parameters optimization.

An association rule mining model with dynamic adaptive thresholds is introduced in [46]. Their algorithm, called *DASApriori* is an extension of Apriori algorithm for finding large item sets. They propose two minimum support counts: Dynamic Minimum Support and Adaptive Minimum Support and two confidence thresholds: Dynamic Minimum Confidence and Adaptive Minimum Confidence.

Zhang et al introduce in [57] an adaptive association rule mining technique, which adapts the support value according to the F-score obtained through prior classification in order to classify web video content based on mixed visual and textual information. The support in this case is computed as a measure for the similarity of the set of terms characterizing the videos under classification. A similar technique for adapting the support value is presented in [37], in which the minimum support threshold is established during the rule generation process in order to maintain the number of generated rules within a desired range. An algorithm called $FRG - AARM$, using a similar approach to the aforementioned paper in order to devise an efficient market basket analysis method, is introduced in [20].

The adaptive association rule mining approaches [46, 57, 37, 20] described above focus on adapting the support value used in rule generation to the actual training data set, such that the generated rules are relevant. However, the algorithm proposed in our paper is adaptive to changes in the nature of the data set, such as the appearance of new features over time, whereas the articles which were presented above focus on improving the quality of the generated rules by adapting certain parameters of the rule mining process such as the support value to the data set particularities.

Chapter 5

Conclusions

We have presented in this report the original scientific results which were obtained for achieving the objectives proposed in the project's work plan for the year 2015. Our main scientific objectives were related to the *development of new classification algorithms for identifying entities with defects in software systems* and to the *design of the AMEL integrated software system*.

The report presented in the first chapter our original approach for identifying the defects in software systems using self-organizing feature maps. The proposed approach may be used for an unsupervised detection of software defects. The experimental results obtained on three open-source data sets reveal a good performance of the proposed approach, it provides better results than many of the existing approaches report in the literature.

Future work will be done in order to extend the evaluation of the proposed machine learning based model on other open source case studies and real software systems. We will also investigate the applicability of fuzzy [56] self-organizing maps for software defect detection, as well as to further consider techniques for data pre-processing and feature selection.

The second contribution of the report was an original approach for adaptive relational association rule mining which can be used for defective software entities detection. We have approached the problem of adaptive relational association rule mining and we proposed a novel algorithm, called *ARARM* for adapting the set of relational association rules mined in a data set, when the feature set describing the objects increases. The experiments were performed on case studies for software defect detection for which the application of adaptive relational association rule mining is useful. The obtained results on these data sets proved that the result were reached more efficiently using the proposed method than running the mining algorithm again from scratch, on the feature-extended object set.

A possible direction to continue our research would be to extend the linear "chain-like" representation for relational association rules to more complex structures. The problem of relational association rule mining may be viewed as a special case of finding frequent chains in a database of labeled graphs [24]. Relational association rules may be viewed as the frequent chains (frequent linear graphs) in this graph database. Using this representation, one may be able to compute the labeled graph canonical form [27] which can be used to avoid reporting duplicate rules (especially "mirror" rules), leading this way to a more efficient filtering. This representation for relational association rules also suggests an extension to allow more general graphs, like trees or graphs including cycles.

As future work we also plan to extend the evaluation of the *ARARM* method to different data sets and real life problems. In addition, an extension of the method proposed in this paper to a fuzzy approach [44, 12] and other approaches for mining association rules [35, 21] will be further considered.

Bibliography

[1] G. Abaei, Z. Rezaei, and A. Selamat. Fault prediction by utilizing self-organizing map and threshold. In *2013 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 465–470, Nov 2013.

[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*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] Rezwan Ahmed and George Karypis. Algorithms for mining the evolution of conserved relational states in dynamic networks. *Knowledge and Information Systems*, 33(3):603–630, 2012.

[4] P.S. Bishnu and V. Bhattacherjee. Software fault prediction using quad tree-based k-means clustering algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 24(6):1146–1150, June 2012.

[5] L.D. Brown, T.T. Cat, and A. DasGupta. Interval estimation for a proportion. *Statistical Science*, 16:101–133, 2001.

[6] Gabriela ¸Serban, Istvan Gergely Czibula, and Alina Câmpan. Medical diagnosis prediction using relational association rules. In *Proceedings of the International Conference on Theory and Applications of Mathematics and Informatics (ICTAMI'07)*, pages 339–352, 2008.

[7] Toon Calders, Nele Dexters, Joris J.M. Gillis, and Bart Goethals. Mining frequent itemsets in a stream. *Inf. Syst.*, 39(0):233 – 255, 2014.

[8] Alina Câmpan, Gabriela ¸Serban, and Andrian Marcus. Relational association rules and error detection. *Studia Universitatis Babes-Bolyai Informatica*, LI(1):31–36, 2006.

[9] Alina Campan, Gabriela Serban, Traian Marius Truta, and Andrian Marcus. An algorithm for the discovery of arbitrary length ordinal association rules. In *DMIN*, pages 107–113, 2006.

[10] C. Catal, U. Sevim, and B. Diri. Software fault prediction of unlabeled program modules. In *Proceedings of the World Congress on Engineering (WCE)*, pages 212–217, Dec 2009.

[11] Gaik-Yee Chan, Chien-Sing Lee, and Swee-Huay Heng. Defending against xml-related attacks in e-commerce applications with predictive fuzzy associative rules. *Applied Soft Computing*, 24(0):142 – 157, 2014.

[12] Chun-Hao Chen, Ai-Fang Li, and Yeong-Chyi Lee. A fuzzy coherent rule mining algorithm. *Applied Soft Computing*, 13(7):3422 – 3428, 2013.

[13] Chun-Hao Chen, Ai-Fang Li, and Yeong-Chyi Lee. Actionable high-coherent-utility fuzzy itemset mining. *Soft Computing*, 18(12):2413–2424, 2014.

[14] Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for object-oriented design. In *Conference Proceedings on Object Oriented Programming Systems, Languages, and Applications*, pages 197–211, 1991.

[15] Gabriela Czibula, Maria-Iuliana Bocicor, and Istvan Gergely Czibula. Promoter sequences prediction using relational association rule mining. *Evolutionary Bioinformatics*, 8:181–196, 04 2012.

[16] Gabriela Czibula, Istvan Gergely Czibula, Adela Sârbu, and Ioan-Gabriel Mircea. A novel approach to adaptive relational association rule mining. *Applied Soft Computing journal*, 36:519–533, November 2015.

[17] Gabriela Czibula, Zsuzsanna Marian, and István Gergely Czibula. Software defect prediction using relational association rule mining. *Inf. Sci.*, 264:260–278, 2014.

[18] Gabriela Czibula, Zsuzsanna Marian, and István Gergely Czibula. Detecting software design defects using relational association rule mining. *Knowledge and Information Systems*, January 2014.

[19] Tera-promise repository. http://openscience.us/repo/.

[20] M. Dhanabhakyam and M. Punithavalli. An efficient market basket analysis based on adaptive association rule mining with faster rule generation algorithm. *The SIJ Transactions on Computer Science Engineering and its Applications*, 1(3):105–110, 2013.

[21] YaJun Du and HaiMing Li. Strategy for mining association rules for web pages based on formal concept analysis. *Applied Soft Computing*, 10(3):772 – 783, 2010.

[22] N. Elfelly, J.-Y. Dieulot, and P. Borne. A neural approach of multimodel representation of complex processes. *International Journal of Computers, Communications & Control*, III(2):149–160, 2008.

[23] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, 2006.

[24] Joseph A. Gallian. A dynamic survey of graph labeling. *The Electronic Journal of Combinatorics*, 17:1–384, 2014.

[25] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[26] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[27] S. G. Hartke and A. J. Radcliffe. Mckay's canonical graph labeling algorithm. *Contemp. Math.*, 479:99–111, 2009.

[28] Yuanchen He, Yuchun Tang, Yan-Qing Zhang, and Rajshekhar Sunderraman. Adaptive fuzzy association rule mining for effective decision support in biomedical applications. *Int. J. Data Min. Bioinformatics*, 1(1):3–18, June 2006.

[29] S. Henry and D. Kafura. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, 7(5):510–518, September 1981.

[30] Rui hua Chang, Xiaodong Mu, and Li Zhang. Software defect prediction using non-negative matrix factorization. *JSW*, 6(11):2114–2120, 2011.

[31] S. Kaski and T. Kohonen. Exploratory data analysis by the self-organizing map: Structures of welfare and poverty in the world. In *Neural Networks in Financial Engineering. Proceedings of the Third International Conference on Neural Networks in the Capital Markets*, pages 498–507. World Scientific, 1996.

[32] Andreas Khler, Matthias Ohrnberger, and Frank Scherbaum. Unsupervised feature selection and general pattern discovery using self-organizing maps for gaining insights into the nature of seismic wavefields. *Computers & Geosciences*, 35(9):1757 – 1767, 2009.

[33] Peter K. Kihato, Heizo Tokutaka, Masaaki Ohkita, Kikuo Fujimura, Kazuhiko Kotani, Yoichi Kurozawa, and Yoshio Maniwa. Spherical and torus som approaches to metabolic syndrome evaluation. In Masumi Ishikawa, Kenji Doya, Hiroyuki Miyamoto, and Takeshi Yamakawa, editors, *ICONIP (2)*, volume 4985 of *Lecture Notes in Computer Science*, pages 274–284. Springer, 2007.

[34] Teuvo Kohonen, Ilari T. Nieminen, and Timo Honkela. On the quantization error in SOM vs. VQ: A critical and systematic study. In *Advances in Self-Organizing Maps, 7th International Workshop, WSOM 2009, St. Augustine, FL, USA, June 8-10, 2009. Proceedings*, pages 133–144, 2009.

[35] R.J. Kuo, C.M. Chao, and Y.T. Chiu. Application of particle swarm optimization to association rule mining. *Applied Soft Computing*, 11(1):326 – 336, 2011.

[36] J. Lampinen and E. Oja. Clustering properties of hierarchical self-organizing maps. *Journal of Mathematical Imaging and Vision*, 2(3):261–272, 1992.

[37] Weiyang Lin, Sergio A. Alvarez, and Carolina Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data Min. Knowl. Discov.*, 6(1):83–105, 2002.

[38] Sayyed Garba Maisikeli. *Aspect Mining Using Self-Organizing Maps With Method Level Dynamic Software Metrics as Input Vectors*. PhD thesis, Graduate School of Computer and Information Sciences Nova Southeastern University, 2009.

[39] Zsuzsanna Marian, Gabriela Czibula, and Istvan Gergely Czibula. Using software metrics for automatic software design improvement. *Studies in Informatics and Control*, 21(3):249–258, 2012.

[40] Zsuzsanna Marian, Istvan Gergely Czibula, Gabriela Czibula, and Sergiu Sotoc. Software defect detection using self-organizing maps. *Studia Universitatis Babes-Bolyai*, LX(2):55–69, December 2015.

[41] Tim Menzies, Bora Caglayan, Zhimin He, Ekrem Kocaguneli, Joe Krall, Fayola Peters, and Burak Turhan. The promise repository of empirical software engineering data, June 2012.

[42] Thomas M. Mitchell. *Machine learning*. McGraw-Hill, Inc. New York, USA, 1997.

[43] Mikyeong Park and Euyseok Hong. Software fault prediction model using clustering algorithms determining the number of clusters automatically. *International Journal of Software Engineering and Its Applications*, 8(7):199–205, 2014.

[44] Slobodan Ribaric and Tomislav Hrkac. A model of fuzzy spatio-temporal knowledge representation and reasoning based on high-level petri nets. *Inf. Syst.*, 37(3):238 – 256, 2012.

[45] N. L. Sarda and N. V. Srinivas. An adaptive algorithm for incremental mining of association rules. In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, DEXA '98, pages 240–, Washington, DC, USA, 1998. IEEE Computer Society.

[46] C. S. Kanimozhi Selvi and A. Tamilarasi. Association rule mining with dynamic adaptive support thresholds for associative classification. In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007) - Volume 02*, ICCIMA '07, pages 76–80, Washington, DC, USA, 2007. IEEE Computer Society.

[47] Gabriela Serban, Alina Câmpan, and Istvan Gergely Czibula. A programming interface for finding relational association rules. *International Journal of Computers, Communications & Control*, I(S.):439–444, June 2006.

[48] Frank Simon, Frank Steinbruckner, and Claus Lewerentz. Metrics based refactoring. In *CSMR '01: Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 30–38, Washington, DC, USA, 2001. IEEE Computer Society.

[49] Panu Somervuo and Teuvo Kohonen. Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters*, 10:151–159, 1999.

[50] Basma Soua, Amel Borgi, and Moncef Tagina. An ensemble method for fuzzy rule-based classification systems. *Knowledge and Information Systems*, 36:385–410, 2013.

[51] Stephen V. Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, 62(1):77 – 89, 1997.

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

[53] Ayse Tosun, Burak Turhan, and Ayse Basar Bener. Validation of network measures as indicators of defective modules in software systems. In *Proceedings of the 5th International Workshop on Predictive Models in Software Engineering, PROMISE 2009, Vancouver, BC, Canada, May 18-19, 2009*, pages 5–14, 2009.

[54] Swati Varade and Madhav Ingle. Hyper-quad-tree based k-means clustering algorithm for fault prediction. *International Journal of Computer Applications*, 76(5):6–10, August 2013.

[55] Renato Vimieiro and Pablo Moscato. A new method for mining disjunctive emerging patterns in high-dimensional datasets using hypergraphs. *Inf. Syst.*, 40:1–10, March 2014.

[56] Lotfi A. Zadeh. A summary and update of "fuzzy logic". In *2010 IEEE International Conference on Granular Computing, GrC 2010, San Jose, California, USA, 14-16 August 2010*, pages 42–44, 2010.

[57] Chengde Zhang, Xiao Wu, Mei-Ling Shyu, and Qiang Peng. Adaptive association rule mining for web video event classification. In *IRI*, pages 618–625. IEEE, 2013.

[58] Kuan Zhang, David Lo, Ee-Peng Lim, and Philips Kokoh Prasetyo. Mining indirect antagonistic communities from social interactions. *Knowledge and Information Systems*, 5(3):553–583, 2013.