SUPERVISED LEARNING FOR ORPHAN ADOPTION PROBLEM IN SOFTWARE ARCHITECTURE RECOVERY

Maryum Bibi[1], Onaiza Maqbool[2], Jaweria Kanwal [3]

[1]Department of Computer Science and Information Technology, University of Azad Jammu and Kashmir.
[2,3]Department of Computer Science, Quaid-i-Azam University, Islamabad, Pakistan

Email: mariyam.hamdani@gmail.com[1], omaqbool@gmail.com[2], kjaweria09@yahoo.com[3]

ABSTRACT

Maintenance of architectural documentation is a prime requirement for evolving software systems. New versions of software systems are launched after making the changes that take place in a software system over time. The orphan adoption problem, which deals with the issue of accommodation of newly introduced resources (orphan resources) in appropriate subsystems in successive versions of a software system, is a significant problem. The orphan adoption algorithm has been developed to address this problem. For evolving software systems, it would be useful to recover the architecture of subsequent versions of a software system by using existing architectural information. In this paper, we explore supervised learning techniques (classifiers) for recovering the architecture of subsequent versions of a software system by taking benefit of existing architectural information. We use three classifiers, i.e., Bayesian classifier, k-Nearest Neighbor classifier and Neural Network for orphan adoption. We conduct experiments to compare the performance of the classifiers using various dependencies between entities in a software system. Our experiments highlight correspondence between the orphan adoption algorithm and the classifiers, and also reveal their strengths and weaknesses. To combine strengths of individual classifiers, we propose using a multiclassifier approach in which classifiers work cooperatively to improve classification accuracy. Experiments show that there is significant improvement in results when our proposed multiclassifier approach is used.

Keywords: orphan adoption, supervised learning, architecture recovery, multiclassifiers

1.0 INTRODUCTION

Software systems evolve, particularly when changes take place in user requirements over time. Pioneering work was carried out by Lehman in the area of software evolution when he postulated laws of software evolution governing software change [1]. Ramil and Lehman [2] define software evolution as 'all programming activity that is intended to generate a new software version from an earlier version'. Thus during software evolution, new versions are launched to accommodate the changes suggested by stakeholders.

To meet stakeholder requirements, new resources may be added in the software systems, and existing resources may change. Vasa et al. [3] explored 275 versions for 12 object-oriented software systems and noted that for 85% of the versions, the numbers of classes changed are greater than the number of classes deleted. Moreover they highlighted that for 80% of the versions, on the average there are 20% classes which are modified, 8% are newly added classes and 5% are deleted classes. In [4], Ali and Maqbool studied 472 releases of 4 software systems and found high addition and modification activities.

Due to changes made in software systems, architectural level documentation deteriorates. Even if the documentation was complete when the system was initially developed, it may not have been updated to reflect the changes. Thus the current architecture may be different from the documented architecture. In this situation, the architecture of the system must be recovered to update existing documentation. For evolving software systems, recovery of architecture is required each time a new version is introduced.

In order to recover the architecture for a certain version, it is important to appropriately accommodate newly introduced resources (called 'orphan resources') in the existing structure. Tzerpos and Holt refer to this issue as the orphan adoption problem [5], and propose the orphan adoption algorithm to address it. In an attempt to maintain the architecture of software systems, the orphan adoption algorithm finds appropriate subsystems in

the existing subsystem hierarchy for the placement of orphan resources. The orphan adoption technique has been adopted in many well-known reverse engineering tools [6, 7, 8, 9, 10, 11].

Machine learning is a growing field whose techniques learn from background knowledge to extract/predict relevant information for the future. These techniques have been used to address problems in diverse areas [12], e.g., biology [13, 14], banking [15] and marketing [16]. Machine learning techniques can broadly be categorized into unsupervised and supervised learning. Unsupervised learning techniques work by placing data into different groups based on similarity [17]. Supervised learning techniques are trained using pre-classified data and build a classifier model. The classifier model then predicts a class label for data instances having unknown class labels.

Clustering, an unsupervised learning technique has been widely adopted for software architecture recovery [6, 8, 18, 19, 20]. Clustering recovers the architecture for a certain version by rebuilding the architecture (grouping together similar entities), and does not rely on the existing architectural information. Although it is useful to use clustering to automatically recover the architecture when documentation is not present, in many cases existing documentation (recovered architecture) for a certain version may be available. Rather than recovering the architecture each time a new version is launched, it may be less expensive, less time consuming and more meaningful to update the existing architecture for subsequent versions. Supervised learning techniques (classifiers) can use the existing information about architecture of previous versions of a software system and update it for subsequent versions.

Multiple classifier combination methods refer to the use of more than one classifier to improve overall classification accuracy. These methods ensemble a number of classifiers using different combination schemes, typically serial and parallel. In the parallel ensemble classifier approach, classifiers are trained in parallel. Output from classifiers is then combined in some way to generate the final output. In the serial classifier combination, the output of one classifier is given as an input to another classifier (next in series). Different terms are used for combined classification methods e.g. aggregation, combination, ensemble, multiple classifier systems [21]. The term cooperation also refers to using more than one classifiers, where classifiers work cooperatively to improve accuracy. Cooperation aims to combine the strengths of different classifiers, therefore overcoming their weaknesses [22]. Delegating classifiers refer to a serial transferring multiclassifier method in which the first classifier classifies those instances for which it is confident and rejects remaining instances. The rejected instances are then forwarded to next classifier(s) for their appropriate classification [23].

In this paper, we explore supervised learning techniques to facilitate architecture recovery of evolving software systems, i.e., software systems with multiple versions whose architecture is not available as they evolve. We specifically focus on the orphan adoption problem, i.e., orphans introduced in the new versions of software systems are accommodated in their appropriate subsystems. Our research contributions in this paper can be summarized as: 1) We explore supervised learning techniques for architecture recovery of subsequent versions of software systems. Three supervised techniques and their characteristics are discussed in detail, 2) We conduct orphan adoption experiments on software systems, and highlight correspondence between the orphan adoption algorithm and supervised learning techniques. Moreover, we analyze different dependencies/relationships such as inheritance, containment and function calls, to evaluate their performance for orphan adoption, 3) We propose using a cooperative approach based on multiple classifiers for the orphan adoption problem. We then conduct experiments in order to show the effectiveness of our approach in reducing misclassifications, 4) When orphans cannot be accommodated in existing subsystems, we suggest a criteria for new subsystem creation.

The rest of the paper is organized as follows. Section 2.0 gives an overview of related work. Section 3.0 presents the orphan adoption algorithm. Section 4.0 describes our orphan adoption approach based on supervised learning techniques. Section 5.0 presents detailed experimental setup. Section 6.0 discusses the experimental results. Section 7.0 describes our proposed Delegating/Cooperative approach and experimental results. Section 8.0 defines our proposed criteria for new subsystem creation. Section 9.0 presents the threats to validity. Section 10.0 concludes the paper.

2.0 RELATED WORK

Different techniques have been suggested to automatically recover the architecture of software systems. Ducasse and Pollet presented a state of the art for software architecture recovery approaches [24]. Their survey is organized on the basis of the goals, the process, the inputs, the techniques and the outputs of SAR approaches.

They mention clustering, a machine learning technique, to be a quasi-automatic technique adopted for architecture recovery. Various clustering algorithms have been proposed in this area. In [25], a clustering technique based on hill climbing and genetic algorithms is presented, which intends to discover partitions with low inter-cluster similarity and high intra-cluster similarity from the partitioned MDG (module dependency graph). The technique is implemented in the BUNCH clustering tool [7]. In [11], an improved BUNCH clustering is proposed for modularization of object oriented systems. It is shown that improved BUNCH clustering is able to produce clusters which are more stable with respect to benchmarks. The ACDC (Algorithm for Comprehension-Driven Clustering) algorithm follows a pattern driven approach to aid the comprehension process of software systems [6]. This algorithm is based on some familiar patterns which are: source file pattern, directory structure pattern, body-header pattern, leaf collection pattern, support library pattern, central dispatcher pattern and subgraph dominator pattern. Results show that ACDC works well in terms of performance, stability, skeleton size and quality of clusters created.

In [18], a linkage algorithm, Weighted combined algorithm (WCA) is proposed for software clustering. Comparative analysis is performed between WCA, Complete linkage (CL) and Combined Algorithm (CA) [27]. It is concluded that WCA performs better than CL and CA. In [19], a scalable hierarchical clustering algorithm LIMBO is introduced to efficiently decompose large systems. Weighting schemes are incorporated and results reveal that they can be useful in clustering. Comparative analysis with other algorithms, ACDC [6], BUNCH and cluster analysis algorithms (Single Linkage (SL), Complete Linkage (CL), Weighted Average Linkage (WA) and Unweighted Average Linkage (UA)) show that LIMBO performs better than all algorithms except UA.

A hybrid approach is proposed in [28] which is based on using lexical information and structural information in the software clustering process. Kleinberg algorithm, a link analysis algorithm, is used along with the Vector Space Model. Evaluation is performed and results are encouraging in terms of authoritativeness, stability, and non-extremity cluster distribution.

In [29], the role of lexical information is explored in software clustering. Lexical information is weighted using weighting scheme based on TF-IDF. Hierarchical Agglomerative Clustering (HAC) is used for clustering. Evaluation is performed using two criteria: Authoritativeness and Non-Extremity Distribution (NED). Results showed that the proposed approach is effective as compared to existing approaches.

In [30], a comparative analysis is performed between different software architecture recovery techniques. Six techniques are selected for this purpose which are: ACDC [6], BUNCH [7], WCA [18], LIMBO [19], Zone-Based Recovery (ZBR) [31] and Architecture Recovery using Concerns (ARC) [32]. These techniques are evaluated in terms of accuracy, authoritativeness and in terms of the recovery criteria used. Overall results are better for the ACDC and ARC algorithms. This is due the fact that the recovery criteria used in these two algorithms reflects the mapping actually done by engineers. Recently, semantic clustering has been studied for software remodularization which intends to use lexical based relations and conceptual metrics [33]. Semantic clusters are generated using clustering and information retrieval techniques. It is indicated that semantic clustering can be useful to discover the facts relevant to architects' ideas for module decomposition.

The term “cooperative” has been used in the area of clustering [34, 35]. Naseem et al. propose Cooperative Clustering Techniques (CCT) in the area of software modularization [34]. They show that it is useful to employ more than one algorithm in a combined way rather than using individual algorithms. For this purpose, cooperative clustering approach is proposed in which weaknesses of individual similarity measures are avoided by combining their corresponding strengths. Two different types of agglomerative hierarchical software clustering algorithms are considered for experimentation. Experiments are conducted on five test software systems. Results show that performance is significantly improved when using cooperative clustering.

Other than clustering, graph partitioning approaches, search based techniques and association rule mining have also been used in the area of software modularization. Mitchell utilized a meta heuristic search based technique for software clustering [36]. He emphasized creating efficient search algorithms and tools for architecture recovery. In [37], a graph based approach is proposed in which a module dependency graph is constructed. Module can be a file, function etc. The graph is then partitioned into subgraphs using elder vector with the goal to increase cohesion and reduce coupling between nodes. In [38], another graph based approach is presented in which entity-relationship graph of software systems is constructed. The recovery process is modeled as a graph pattern matching problem. Architectural patterns are matched by applying a graph matching algorithm.

Harman applied search based algorithms for software modularization. In [39], a new cross over operator is proposed which is more suitable for genetic techniques as compared to existing standard operators. Software clustering is considered as a multi objective search problem in [40]. Two multi objective representations are formed in which cohesion and coupling principles are separately represented. The multi objective approach gives better performance as compared to the single objective approach. Association rule mining has also been explored for architecture recovery [41]. Experimental results show the ability of proposed method to aid in program comprehension.

Although much work has been done for software architecture recovery using clustering, supervised learning techniques (e.g. classification) have not been explored in detail in this area. An initial study has been carried out in [42] in which Bayesian learning has been used to update incomplete or out of date documentation of software systems. Existing documentation of a software system is utilized and Bayesian classification has been used to classify the new software modules into appropriate subsystems. Experiments are conducted on one software system to validate the effectiveness of the approach.

In [43], "No Free Lunch" theorems are formulated. These theorems highlight mathematically that performance of a single classifier cannot always be better for all problems. In [21], multiclassifier combination methods are presented on the basis of "No Free Lunch" theorems. It is discussed that the multiclassifier approach is useful in order to combine interesting characteristics of individual classifiers [21]. Therefore, other than using individual classifiers, multiclassifier approach is also focused in diverse areas e.g. pattern recognition and biomedical systems.

In [22], a multiclassifier approach has been used in the area of pattern recognition. Three different methods are used to combine output from classifiers i.e. the highest rank, the Borda count and logistic regression methods. Experiments are conducted on different applications e.g. on degraded machine printed characters and handwritten digits. Results show that multiclassifier approach plays a vital role in order to improve overall performance.

Delegation refers to the use of a classifier for the predictions of those instances for which it has a certain level of confidence. Instances which cannot be classified by the first classifier are rejected and passed to other classifier(s). In [23], delegating classifiers are used to study performance improvements. Experiments are conducted on 22 datasets from the UCI dataset repository (e.g. Breast cancer, Heart diseases). It is experimentally shown that results are significantly improved.

In [44], an extended version of Fuzzy rule-based multiclassification system is proposed. Experiments are performed on 29 data sets (high dimensional) collected from UCI and KEEL repositories. Results are shown to be promising in terms of high accuracy and less complexity.

Software architecture recovery is crucial for maintenance of evolving software systems. New versions of software system are launched when software changes take place [45]. Relatively less work has been done on utilizing software versions in the area of architecture recovery. In [46], version information is used in an unsupervised learning technique, i.e., clustering, to provide an understandable view of a software system. Information from multiple versions is used to improve the clustering quality.

For maintenance of architectural documentation, an orphan adoption algorithm was designed, that is based on incremental clustering and corrective clustering [5]. Incremental clustering is proposed in an attempt to accommodate newly introduced resources (orphan resources) in their appropriate subsystems and corrective clustering is proposed to accommodate structural changes (resource re-adoption). The orphan adoption approach has been adopted in many well-known reverse engineering tools [6, 7, 8, 9, 11].

The present work is different from other architecture recovery approaches based on clustering, which assume that architectural documentation is unavailable. Since software systems generally evolve over time, and have multiple versions, our approach is more practical since it utilizes information about architecture of a certain version to recover architecture of a subsequent version. We specifically focus on placement of new resources in subsystems, for which we explore supervised learning techniques. This makes our work different from other studies done in the area.

3.0 THE ORPHAN ADOPTION ALGORITHM

The orphan adoption algorithm [5] is concerned with accommodation of newly added resources (e.g. source files, procedures, variables) in existing subsystems. If the software structure is documented, the orphan adoption algorithm uses certain criteria to find an appropriate subsystem for an orphan resource.

Before discussing the orphan adoption criteria, we define the terminology and an example used in subsequent sections. Let $R = \{r1, r2, r3 \dots \dots . ri\}$ is the set of all resources, $S = \{s1, s2, s3 \dots . . sj\}$ is the set of all subsystems in a software system version having architectural documentation. Newly introduced resources (orphan resources) in some subsequent version of the software system are represented as $O = \{o1, o2, o3 \dots on\}$.

Consider the example in Fig. 1, in which there are six resources r_1- r_6 and three subsystems s_1-s_3 in a version with architectural documentation. Subsystem s_1 contains r_1 and r_2, s_2 contains r_3 and r_4 and s_3 contains r_5 and r_6. Suppose there is one newly introduced resource o_1 in a subsequent version of the software system. In this example r_1, r_3, r_5 and r_6 depend (dependencies may be of different types, e.g., function calls in structured systems, inheritance relationships in object oriented systems) on orphan resource o_1. r_2 is dependent on r_6, r_3 depends on r_1 and r_5 depends on r_4 (dependencies are indicated by directed lines).

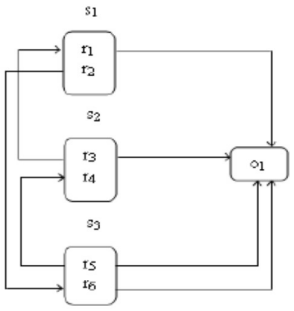

Fig. 1. Classification techniques and orphan adoption algorithm example.

3.1 Orphan Adoption Criteria

Tzerpos and Holt [5] defined a set of criteria to address orphan adoption. In the 'Naming' criteria, subsystems for newly introduced resources are identified by using naming conventions followed e.g. a file 'chartcolor' may belong to a subsystem 'chart'. In the 'Structural criteria', an appropriate subsystem is found using static dependencies, e.g., source inclusion, function calls, data references.

The set of all dependencies between resources are represented as $d = \{d_j, j: 1 \dots \dots . n_d\}$. Dependency d_j can be denoted in the form of an ordered pair, e.g., if d_1 denotes a function call, then in Fig. 1, $d_1 = \{(r_2, r_6), (r_3, r_1)\}$ means r_2 calls r_6 and r_3 calls r_1. Generally we can define all dependencies as $D = \cup_{j=1}^{n} d_j$. rDr' represents that resource r depends on resource r'.
Let $pa(o)$ represent parent or subsystem of orphan resource o. Orphan resource o will be adopted in a subsystem which depends most on o for its functionality, i.e.

$$pa(o) = s_{k,} \ if \ W(s_{k,}, o) \geq W(s_{i,}, o), \forall \ i \in 1 \dots n_s \quad (1)$$

where W is the weight of dependencies between resources of existing subsystems and newly introduced resources, $W(s_{k,}, o)$ represents the weight of dependencies between resources of existing subsystem $s_{k,}$ and the orphan resource o. Formally, $W(s_{k,}, o) = |N|, \ N = \{r | pa(r) = s_{k,} \wedge rDo\}$

Equation 1 will find the subsystem for orphans by calculating the number of dependencies that exist from the existing subsystems to the orphans.

As an example, consider Fig. 1 in which we have to find an appropriate subsystem for o_1. According to structural criteria in equation 1, dependencies will be calculated as, $W(s_1, o_1) = 1$, $W(s_2, o_1) = 1$, $W(s_3, o_1) = 2$ as one resource r_1 of s_1, one resource r_3 of s_2 and two resources r_5 and r_6 of s_3 are dependent on o_1 which implies that $W(s_3, o_1) > W(s_2, o_1)$ and $W(s_1, o_1)$. Subsystem s_3 seems to depend most on orphan o_1, as the number of dependencies from subsystem s_3 towards o_1 are higher. Therefore, o_1 will be accommodated in s_3.

3.2 Tie-breakers in the Orphan Adoption Algorithm

Tie breakers are applicable in the situations where more than one subsystems carry equal dependency weight for a given orphan resource (such subsystems are referred to as candidate subsystems). Two criteria are defined to resolve such ties. The first criterion suggests accommodating the new resource in the candidate subsystem whose interface size decreases by incorporation of the new resource. The interface size refers to the number of resources of a subsystem on which resources of other subsystems depend for their functionality. Suppose *I(s)* represents interface for subsystem s then,

$$I(s) = \{r | pa(r) = s \wedge \exists r' : pa(r') \neq s \wedge r' D r\} \quad (2)$$

where *r* represents the resource of a subsystem s upon which resource r' depends for its functionality. r' is not a part of the subsystem s.

In the example given in Fig. 1, suppose that there is one resource r_7 in s_3 and two more resources r_8 and r_9 in s_2, resource r_7 of subsystem s_3 is also dependent on orphan resource o_1 and o_1 is dependent on $r_8$and r_9. For the orphan adoption algorithm, weights of dependencies for subsystems s_1, s_2 and s_3 then become, $W(s_1, o_1) = 1$, $W(s_2, o_1) = 3$, $W(s_3, o_1) = 3$. Two subsystems s_2 and s_3 have the same weight therefore a tie results between these subsystems. The tie between candidate subsystems s_2 and s_3 is resolved as follows: Defining the interfaces for the two subsystems s_2 and s_3 as, $I(s_2) = \{r_4, r_8, r_9\}$ and $I(s_3) = \{r_6\}$. Therefore $Size(I(s_2)) = 3$ and $Size(I(s_3)) = 1$ [5]. Now, if orphan resource o_1 is accommodated in s_2 then $I(s_2) = \{r_4, o_1\} = 2$, if accommodated in s_3 then, $I(s_3) = \{r_6, o_1\} = 2$. As the size of interface of s_3 increases and interface size for s_2 decreases, therefore o_1 will be accommodated in s_2.

The second criterion suggests that the new resource be adopted in the candidate subsystem whose supplier set is increased little. Suppose *Sup(s)* is a set of supplier subsystems for s then,

$$Sup(s) = \{s' | s' \neq s \wedge \exists r : pa(r) = s \wedge \exists r' : pa(r') = s' \wedge r D r'\} \quad (3)$$

where s' represents the supplier for subsystem s if resource *r* of s depends on the resource r' of s' for its functionality.
For the above example, according to second criterion supplier set for subsystems s_2 and s_3 is defined as, $Sup(s_2) = \{s_1\}$, $Sup(s_3) = \{s_2\}$. After accommodation of o_1 in s_2, its supplier set remains $Sup(s_2) = \{s_1\}$. If o_1 is accommodated in s_3, then its supplier set also remains $Sup(s_3) = \{s_2\}$. As the supplier set for both systems remains the same therefore o_1 will be accommodated either in s_2 or s_3.

If tie breakers cannot resolve the tie between candidate subsystems, then a subsystem is assigned to the orphan resource randomly.

4.0 ORPHAN ADOPTION BASED ON SUPERVISED LEARNING

In this section, we present our approach for orphan adoption. We intend to use supervised learning for orphan adoption. For this purpose, we highlight correspondence between the orphan adoption algorithm and supervised learning in general. We then introduce three supervised learning techniques in this section.

4.1 Correspondence between Supervised Learning and the Orphan Adoption Algorithm

The orphan adoption problem and working of the orphan adoption algorithm is illustrated in Fig. 2, in which we have existing resources R with their corresponding subsystems $S = \{s_1, s_2\}$. Using this information, the orphan adoption algorithm will find appropriate subsystems for orphan resources $O = \{o_1, o_2\}$ depending upon the orphan adoption criteria.

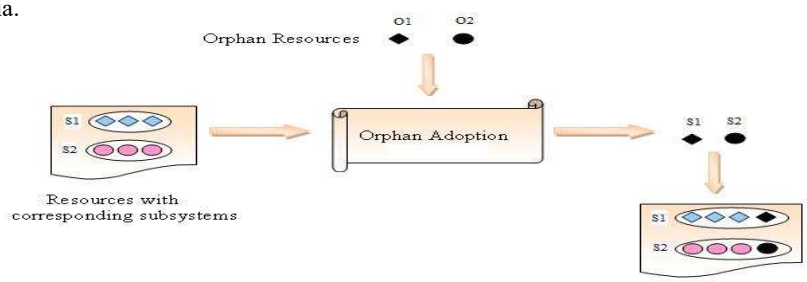

Fig. 2. Orphan adoption problem.

As described earlier, supervised learning techniques use pre-classified data for training. For the orphan adoption problem, resources of the existing system and their corresponding subsystems represent the pre-classified data which can be used to train a classifier. The classifier model thus, built predicts the subsystems (class labels) for new resources (data points).

Thus, there is similarity in the working of supervised learning techniques and the orphan adoption algorithm. Using existing information, both approaches intend to find class/subsystem for the resources with unknown class/subsystem.

We describe three well known classification techniques, Bayesian (based on probability theory), k-Nearest-Neighbor (an instance based learner) and Neural Networks (capable of forming relatively more complex decision surfaces) for the orphan adoption problem.

4.2 Bayesian classification

Bayesian classification is a probability based classifier which assigns the unseen instance to a class with maximum posterior probability. In the context of our study, Bayesian classifier can be used to predict subsystem/class for newly introduced resources (orphan resources) based on structural criteria [5]. Bayesian classifier follows the Bayes theorem for calculation of probabilities. For our problem, we can write the Bayes theorem as:

$$p(s_k|o_t) = \frac{p(o_t|s_k)p(s_k)}{p(o_t)} \quad (4)$$

where $k=1,2,3....j$ represents the number of subsystems, and $t=1,2,3....n$ represents the number of orphans. $p(s_k)$ and $p(o_t)$ are the prior probabilities for subsystem s_k and orphan resource o_t respectively. $p(o_t|s_k)$ is the posterior probability of o_t conditioned on s_k and $p(s_k|o_t)$is the posterior probability of s_k conditioned on o_t. Since value of $p(o_t)$ does not change for a certain resource it can be ignored.
For the example in Fig. 1, Bayesian classifier will find the probabilities as:

Prior probabilities:
$p(s_1)=\frac{2}{6}= 0.33$, $p(s_2)=\frac{2}{6} = 0.33$, $p(s_3)=\frac{2}{6}= 0.33$
Posterior probabilities:
$p(o_1|s_1)=\frac{1}{2}= 0.5$, $p(o_1|s_2)=\frac{1}{2} = 0.5$, $p(o_1|s_3)=\frac{2}{2}= 1$
$p(s1|o1)=p(o1|s1)p(s1)=0.5* 0.33 = 0.16$,
$p(s2|o1)=p(o1|s2)p(s2)=0.5* 0.33 = 0.16$,
$p(s3|o1)=p(o1|s3)p(s3)=1* 0.33 = 0.33$

From the above posterior probabilities, the maximum posterior probability is 0.33 for subsystem *s3*, therefore new resource *o1* will be assigned to subsystem *s3*.

4.3 k-Nearest-Neighbor Classification

k-Nearest-Neighbor learning is a supervised learning technique that classifies an unseen tuple by extracting k closest points (nearest neighbors) from the training data [17]. k is user defined and shows the number of nearest neighbors. Distance metrics such as Euclidean distance, and Manhattan distance [47] are usually used to find the nearest points.

In the context of our study, k-Nearest-Neighbor may be used to predict the subsystem for orphan resources by extracting k number of nearest points (resources) from training data on the basis of structural criteria [5]. The orphan resource is assigned to the subsystem having majority of k nearest resources.

We define the distance between two resources as follows: If some dependency exists between two resources then their distance will be considered as 0. If there is no dependency between two resources then the distance will be taken as 1. It can be seen from Fig. 1 that r_1, r_3, r_5 and r_6 depend on orphan resource o_1 and two resources r_2 and r_4 do not depend on o_1. Therefore distance between the existing resources and orphan resources will be calculated as:

dist(o1, r1)=0 dist(o1, r2)=1,
dist(o1, r3)=0 dist(o1, r4)=1,
dist(o1, r5)=0, dist(o1, r6)=0

k-Nearest-Neighbor will extract the specified number of nearest resources from resources dependent on o_1. If no resource is dependent on o_1 then the nearest resources will be selected randomly. In order to select three nearest neighbors, i.e., k = 3, the classifier will randomly pick three nearest neighbors from r_1, r_3, r_5 and r_6 as these four resources are identified to be dependent on o_1. If the k-Nearest-Neighbor classifier selects r_1, r_5 and r_6, then o_1 will be accommodated in s_3 according to majority voting assumption. If r_1, r_3 and r_5 are selected as nearest neighbors, k-Nearest-Neighbor results in a tie between s_1, s_2 and s_3 therefore o_1 will be accommodated arbitrarily in one of these subsystems.

There is a need to avoid random selection for nearest neighbors in k-Nearest-Neighbor classifier which is due to fixed k. Thus rather than extracting k number of nearest neighbors, we select all extracted resources having dependencies with orphan resources (NNall). This method of all nearest neighbor selection is known as the local method [48]. In example Fig. 1, we select k = 1 + 1 + 2 = 4 nearest resources. Therefore we have r_1, r_3, r_5, and r_6 as nearest neighbors. We can see that for subsystem s_3 we have majority voting of 2 nearest neighbors. Therefore o_1 will be accommodated in s_3.

4.4 Neural Network

A Neural Network is a classification technique in which different neurons work collaboratively [16]. A Neural Network is organized into input, hidden and output layers of neurons as shown in Fig. 3. There can be a number of hidden layers between the input layer and output layer. The input layer accepts input in the form of features for each training tuple (training data). Input is processed by the input layer, and is then weighted and fed into the hidden layer(s). Net input for hidden and output layer is computed using the following relation:

$$I_j=\sum_i wijOi+\theta_j \quad (5)$$

where, w_{ij} is the weight of the link between previous unit i and current unit j as shown in Fig. 3. It is a small number generated randomly e.g. [-0.5 to 0.5]. O_i represents output of previous unit i. θ_j represents bias for current input unit j which is also a small random number. Hidden layer and output layers accept the net input and then use an activation function in order to give corresponding output. Different activation functions can be used, e.g. sigmoid function which is as follows:

$$O_j = \frac{1}{1+e^{-I_j}} \quad (6)$$

In the Backpropagation method [17], the weights are adjusted for learning iteratively. Iterative learning involves comparison of network's predicted value with target value. To minimize the error, the weights are modified and propagated to backward layers i.e. to input layer through hidden layer(s). Error for the output layer is computed as follows:

$$Err_{j=}O_j(1-O_j)(T_j-O_j) \tag{7}$$

where, T_j is the target value for a given training tuple. Error at hidden layer is calculated as follows:

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk} \tag{8}$$

where, w_{jk} represents weight of the link between unit j (previous layer) and unit k (next layer), Err_k is the error of unit k.
Weights and biases are updated using the following relations:

$$\Delta w_{ij} == (l) Err_j O_i \tag{9}$$
$$w_{ij} \;\; w_{ij} + \Delta w_{ij}$$

$$\Delta\theta_j == (l) Err_j \tag{10}$$
$$\theta_{j=}\theta_j + \Delta\theta_j$$

where l is learning rate, whose value usually varies between 0.0 to 1.0.

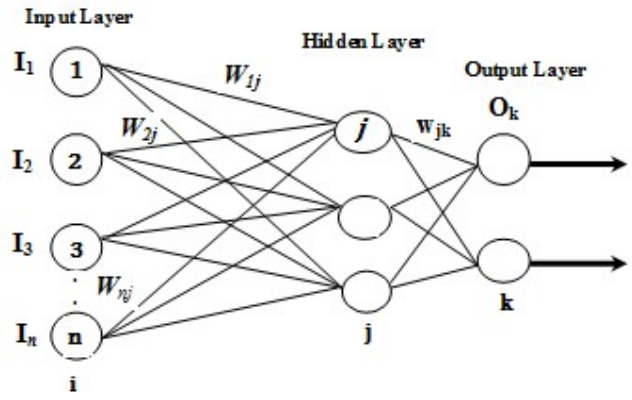

Fig. 3. Neural Network.

For our orphan adoption problem, a Neural Network may be trained by using resource dependencies as input and subsystems as output. Considering Fig. 1, dependencies r_1- r_6 between resources are given as input and the three subsystems s_1-s_3 represent the output neurons in the output layer. When given an orphan resource, the Neural Network will try to find an appropriate subsystem for it by finding similar dependency patterns. Since the result of a Neural Network depends on various parameters, it is not possible to be certain about the subsystem which is identified for the orphan resource in Fig. 1. This is unlike the working of the Bayesian and K-Nearest-Neighbor classifier discussed earlier.

5.0 EXPERIMENTAL SETUP

5.1 Overview of Process for Experiments

We performed experiments to evaluate the performance of the orphan adoption algorithm and supervised learning techniques. In Fig. 4, we illustrate our approach for utilizing the documentation of some software version V1 and updating it for a subsequent version V2. Each step in the process is described in more detail in the following sections.

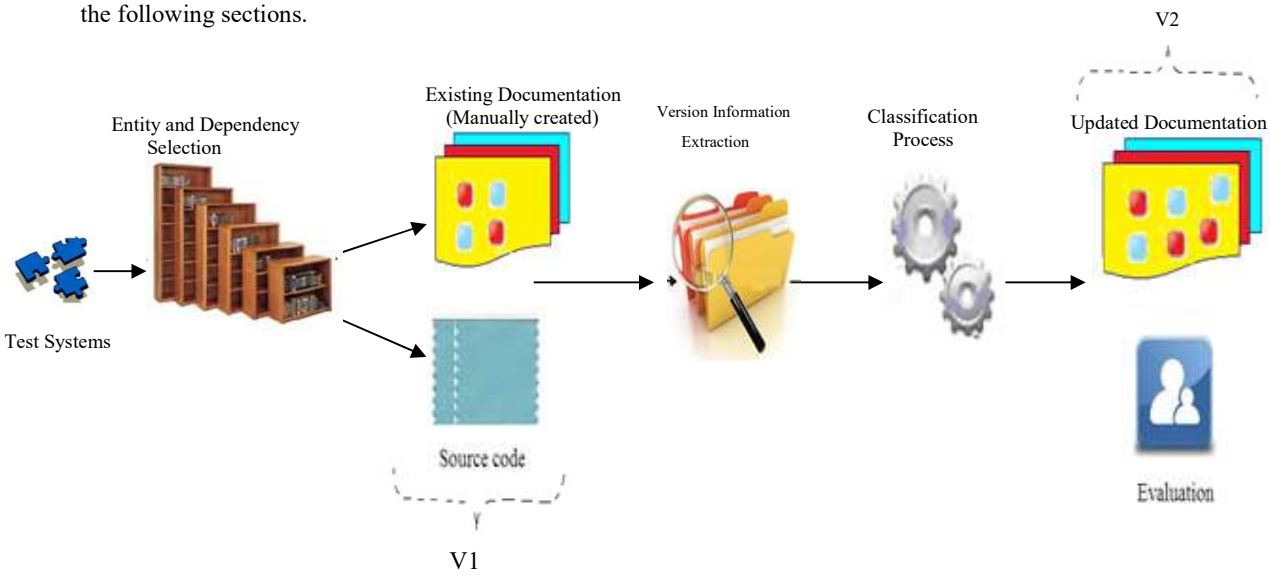

Fig. 4. Process for experiments.

5.2 Test Systems

For our experiments, we selected six object oriented open source systems JHotdraw [49], Jedit [50], Jfreechart [51], Pmd [52], Junit [53] and Jabref [54]. These systems have been developed in Java and have continuously evolved with the passage of time. Relevant statistics and versions of the test systems are given in Table 1. All these systems have previously been examined in [29]. JHotdraw and Jedit have also been explored in [28].

Table 1. Test systems' statistics

System	Selected Versions	Files	Classes	Functions	Added classes	Deleted classes
JHotdraw	5.2	160	168	1458	-	-
	5.3	195	235	2183	68	1
Jedit	4.0.3	296	483	3140	-	-
	4.1	323	532	3564	71	22
Jfreechart	1.0.12	960	989	11363	-	-
	1.0.13	989	1020	11783	31	0
Pmd	4.1	184	197	1963	-	-
	4.2	214	229	2392	34	2
Junit	4.5	101	111	781	-	-
	4.8	121	135	909	24	0
Jabref	2.4.2	530	762	4488	-	-
	2.5	553	798	4655	36	0

5.3 Entity and Dependency Selection

We define entities to be the resources of software systems. Since in our study we used object oriented systems, therefore we selected the classes as entities.

Dependencies are basically relationships that exist between resources. According to the structural criteria [5], the orphan resource is placed in a subsystem which depends on the orphan resource the most. In object

oriented systems, common dependencies are, for example, inheritance, containment, and function calls. Dependencies/relationships that we used are listed in Table 2. We have categorized these dependencies into Direct (D) and Indirect (I). Entities are said to have a direct relationship if they are directly connected with each other. As an example, inheritance (D) relationship exists if a class c_1 is inherited from a new class c_n. To differentiate this from the case where the new class c_n is inherited from an existing class c_1, we call the latter relationship "Invert inheritance". Boolean representation is used to indicate whether a relationship exists or not, i.e., here the relationship between c_1 and c_n is represented as 1. Entities are indirectly related to each other if they share some common features [55]. For example, same inheritance hierarchy (I) relationship exists between new class c_n and existing class c_1 if they are inherited from the same class c_2.

Table 2. Relationships and categories

Relationship	Category	Description
Inheritance	D	Existing class inherited from orphan class
Invert Inheritance	D	Orphan class inherited from existing class
Same Inheritance Hierarchy	I	Orphan and existing class inherited from same class
Second Level Inheritance	D	Orphan class inherited from existing class which is further inherited from some other existing class
Containment	D	Object of orphan class is declared as member variable in existing class
Invert Containment	D	Declaration of object of existing class in orphan class
Same Class Containment	I	Orphan class and existing class have member variable of some other existing class
Second level Containment	D	Object of orphan class is member variable of existing class whose object is member variable of some other existing class
Function calls	D	Function of existing class calls function of orphan class
Invert Function calls	D	Function of orphan class calls function of existing class
Same Function Hierarchy	I	Orphan class and existing class calls function of some other existing class
Second Level Function Calls	D	Orphan class calls function of existing class which further calls function of other existing class

5.4 Existing Architectural Documentation

As shown in Fig. 4, architectural documentation for test systems is required. This documentation indicates the decomposition of a system into subsystems, along with entities contained in each subsystem. We utilized manually created documentation for a version to aid in the automatic recovery of the documentation for a later version. To differentiate such a manually created documentation from the one created automatically, we call it an expert decomposition.

As an example, consider JHotdraw. We used an expert decomposition for JHotdraw 5.2 to recover the documentation for version JHotdraw 5.3.

The expert decompositions for the test systems were developed in the following manner. Expert Decompositions for software systems Jhotdraw, Jedit, Jfreechart and Pmd were obtained using the directory structure as in [28], [29], where a two-step procedure is followed in order to create expert decompositions for these systems, which is described below:

- Construct the hierarchy of systems' directories and assume each directory to be a subsystem.
- If any subsystem in a hierarchy contains total number of five or less than five source files then merge it with its parent subsystem.

In the above processes, source files are considered as entities in order to build the expert decompositions. Since we selected classes as entities, therefore in an expert decomposition if source file f belongs to some subsystem s, all classes within f will be considered to belong to s.

For Junit and Jabref we consulted a human expert to create decompositions for the software systems. The expert had an MS in Computer Science, and was selected on the basis of experience in the area of object oriented

system development, data mining and software architecture recovery. We asked the expert to develop decompositions of systems by considering classes as entities. We provided the source code of the three software systems, but did not reveal details about relationship information and techniques being studied. This was to allow the expert to develop decompositions based on information he thought relevant, without influencing him with details about the relationships we used. We gave duration of three months for creating decompositions of the two test systems.

5.5 Version Information Extraction

To extract classes and dependencies from the source code of test systems, we used the open source fact extractor Infusion [56]. Infusion supports c, c++ and Java based systems. It parses the source code and generates an mse file which contains information about entities and dependencies between entities in a version of a software system. From the mse file, we needed to extract newly introduced classes, existing classes, deleted classes and dependencies. For this purpose, we developed our ZoomFactExtractor which extracts these entities and relationships.

5.6 Classification Process

We used Bayesian, k-Nearest-Neighbor and Neural Network classification techniques discussed in Sections 4.2, 4.3 and 4.4 for classification of orphan resources. Information in the form of classes and dependencies from the previous version (Section 5.5), along with the expert decomposition (Section 5.4) was used as training data for the classifiers.

For Neural Network, we need to set some parameters, e.g. number of input neurons (IN) and output neurons (ON). The number of input neurons was set as the number of dependencies (features) for resources. Since the number of features varies for each test system, a separate Neural Network was required for each features. For this purpose, we designed a generalized Neural Network, in which the number of input neurons was adjusted to be the number of input features.

Similarly, the numbers of output neurons are the number of subsystems. The number of output neurons also varies with respect to the test systems. We experimented with a different number of hidden neurons (HN), systematically increasing it from one. We chose HN=3 due the fact that we got better results for this configuration. We also need to set some other parameters relevant to a Neural Networks configuration i.e. learning rate (l), target value (t). We set the parameters as: l=0.7, t=0.8. Target value is the output value for a class (subsystem). Generally the target value is 1 or 0. The value 1 for a particular subsystem means that the given instance (class here) belongs to that subsystem, and 0 means it does not belong to that particular subsystem. To avoid complexity in backpropagation iterations, we set the target value as 0.8. If the Neural Network finds a target value greater or equal to 0.8 for some particular subsystem, then that subsystem is considered to be the appropriate subsystem for an orphan. We use the Sigmoid activation function represented as a non-linear function (equation 6) as it can model those inputs which are linearly inseparable. Its output lies between 0 and 1 so that its values can be taken as probabilities. Moreover, it has been commonly adopted in various studies [57, 58, 59].

Table 3. Feature vector representation

Existing Resources	r1	r2	r3	r4	r5	r6	Subsystems
r2	0	0	0	0	0	1	S3
r3	1	0	0	0	0	0	S1
r5	0	0	0	1	0	0	S2

Considering Fig. 1, the binary representation of features for corresponding resources is given in Table 3. Those resources which do not convey any dependency information (do not depend on any resource) are not shown in

Table 4. According to Table 3, there will be six input neurons and three output neurons (**S1- S3**) in the Neural Network structure. The number of input and output neurons can vary for each test system.

5.7 Evaluation

To evaluate the classification techniques, we used external assessment in which the automatically created decomposition is compared with an expert decomposition. Thus expert decompositions were prepared not only for the previous version, but also for the version for which the decomposition was recovered automatically. The procedure followed for developing the decompositions has been discussed in Section 5.4.

To perform a comparison between the automatically created decomposition, and the one created manually for the later version, we used the Misclassifications (MC) measure (equation 11) which calculates the percentage of orphan classes which are classified in inappropriate subsystems:

$$MC = \frac{number\ of\ orphan\ classes\ misclassified}{total\ number\ of\ orphan\ classes} * 100 \quad (11)$$

MC value of 0 shows that the orphan classes are assigned to their appropriate subsystem (where appropriate denotes that the orphan has been placed in the same subsystem as the one designated to be the correct subsystem in the expert decomposition). An MC value of 100 shows that all orphan classes are accommodated in inappropriate subsystems.

6.0 EXPERIMENTAL RESULTS AND ANALYSIS FOR INDIVIDUAL CLASSIFIERS AND OOA

To compare the performance of the classifiers and the orphan adoption algorithm and ease analysis, we conducted separate sets of experiments for two cases that may arise in software systems with respect to dependencies between the orphan and subsystems. These two cases are presented in Fig. 5, in which we have 12 resources, r_1-r_{12}, 2 subsystems, s_1 and s_2 and one orphan resource o_1. In general, there can be x dependencies from subsystem s_1 towards o_1 and y dependencies from s_2 towards o_1. In Case I, r_1 and r_6 of subsystem s_1 and r_7, r_8 and r_{12} of s_2 depend on o_1, thus x≠ y. In Case II, equal number of dependencies are present from existing subsystems to orphan resource o_1, so we have x ==y.

Table 4. Relationship statistics of the test systems

System	All Inheritance	All Direct Inheritance	All Indirect Inheritance	All Containment	All Direct Containment	All Indirect Containment
JHotdraw	80	43	37	149	31	118
Jedit	50	30	20	61	54	7
Jfreechart	107	19	88	27	21	6
Pmd	240	101	139	55	18	37
Junit	21	10	11	5	1	4
Jabref	78	31	47	61	21	40
System	**All Function calls**	**All Direct Function calls**	**All Indirect Function calls**	**All Direct**	**All Indirect**	**All Direct and Indirect**
JHotdraw	57319	9839	47480	9913	47635	57548
Jedit	56301	21070	35231	21154	35258	56412
Jfreechart	30483	2627	27856	2648	27950	30598
Pmd	33594	18417	15177	18536	15353	33889
Junit	2691	1182	1509	1210	2691	1479
Jabref	34032	2941	31091	2993	31178	34171

Before the discussion of results, we present relationship statistics for the test systems in Table 4. It can be seen from Table 4 that the relationship statistics vary for each test system. However, for all test systems, the number of function calls is greater than the inheritance and containment relationships. Moreover, except for Pmd which has a higher number of direct function calls, all other systems have a higher number of indirect function calls. JHotdraw and JEdit have a higher number of containment relationships as compared to inheritance relationships,

but in all other test systems, number of inheritance relationships is higher.

6.1 Results and Discussion for Case I (x≠y)

Results for Bayesian, K-Nearest Neighbor, Orphan Adoption algorithm and Neural Network are presented for the different relationships in Table 5. Some observations are as follows:

1. **Performance of individual classifiers and orphan adoption algorithm -** Results for Bayesian, K-Nearest Neighbor and the Orphan Adoption algorithm are identical, but different from the results of Neural Networks.

   To explain these identical results, consider Case I in Fig. 5. The Bayesian classifier will calculate the probabilities for this case as:

   $p(s1|o1)=p(o1|s1)p(s1)=2/6*6/12 = 0.16$

   $p(s2|o1)=p(o1|s2)p(s2)=3/6 *6/12 = 0.25$

   The maximum posterior probability for s_2 is 0.25; therefore o_1 will be accommodated in s_2. For k-Nearest Neighbor, using NNall criteria, r_1, r_6, r_7, r_8 and r_{12} are selected as nearest resources, 3 nearest resources belong to s_2 and 2 nearest resources belong to s_1. Thus, o_1 will be placed in s_2. The Orphan adoption algorithm will calculate the weightage of dependencies as:

   $W(s_1, o_1) = 2, W(s_2, o_1) = 3, W(s_2, o_1) > W(s_1, o_1)$

   Therefore according to orphan adoption algorithm, o_1 will be accommodated in s_2 as it has higher number of dependencies towards orphan o_1.
   Thus in this case, Bayesian classification, k-Nearest-Neighbor, and orphan adoption algorithm behave identically. Behavior of these techniques is identical in all such cases where x < y or y < x, which means that we have one subsystem among the various subsystems with higher number of dependencies with the orphan.

   Neural Network results are different from those of the other classifiers. For some relationships they are better, and for some of them they are worse. It is relevant to note that for Case I, the Neural Networks were trained using the dependencies between resources and subsystems (interdependencies) as features. This was to make a comparison between them and the Bayesian classifier, k-Nearest-Neighbor, and orphan adoption algorithm fair, since these classifiers also use only these dependencies. Thus the Neural Network may not have been able to learn when there were a limited number of features.

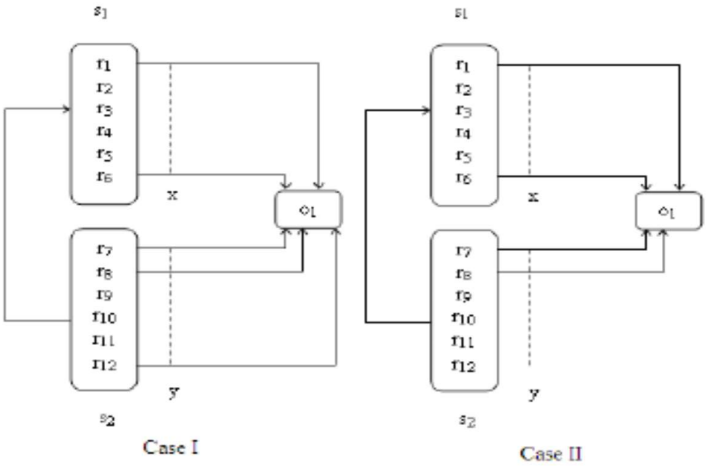

Fig. 5. Two cases to study behavior of orphan adoption algorithm and supervised learning techniques.

2. **Performance of relationships** - It can be seen that on the average, the Inheritance relationship has performed better than other relationships for all test systems. Moreover, indirect Inheritance performs better than direct Inheritance for all systems except Pmd (in case of Neural Networks, results of Pmd are better for indirect inheritance). Given the much larger number of function calls, this is an interesting result. It indicates that the Inheritance relationship is more useful for orphan adoption as compared to the Function call relationship.

Another observation is that combining relationships (represented by All in Table 5) degrades results significantly. This indicates that for orphan adoption, it is more useful to analyze individual relationships. Instead of providing useful information, combining relationships may confuse the classifiers, thus resulting in deteriorated results. This is similar to our observation while clustering object-oriented systems, where reasons for this behavior have also been discussed [60].

Table 5. Experimental results for Case I (x≠y)

System	All Inheritance		All Direct Inheritance		All Indirect Inheritance		All Containment		All Direct Containment		All Indirect Containment	
	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN
JHotdraw	50.0	46.15	62.50	55.56	28.57	50	73.33	66.67	66.67	76.15	37.50	50
Jedit	43.75	46.67	56.25	61.11	12.50	44.44	55.56	66.70	44.44	84.62	66.67	75
Jfreechart	0	42.86	12.50	45.45	0	28.57	75	62.50	75	75	85	85.71
Pmd	14	33.33	16.67	50	33.33	16.67	57	60	80	60	80	40
Junit	60	40	60	50	33	33.33	0	0	0	0	0	0
Jabref	45.45	50	81.82	58.33	36.36	41.67	83	83.33	83	66.67	100	50
Performance Avg	35.53	43.16	48.29	53.40	23.96	35.78	57.31	56.53	58.18	60	64.86	50
System	**All Function calls**		**All Direct Function calls**		**All Indirect Function calls**		**All Direct**		**All Indirect**		**All Direct and Indirect**	
	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN	Bayesian/ OOA/ kNN	NN
JHotdraw	90.20	75.56	88	66	75	57.5	70	80	56.92	54.41	72.73	67.86
Jedit	62.50	70.83	62.5	67.50	56.52	61.29	71.43	72.09	61	64.62	70.31	65.71
Jfreechart	88.89	80	100	65	100	53.33	78.26	65.22	75	62.07	65.38	66.67
Pmd	50	71.43	27	85.71	50	57.14	25	75	70	76.67	63.33	74.19
Junit	88.89	60	69.23	52.94	66.67	53.85	64	66.67	78.57	57.14	75	66.67
Jabref	88.24	62.5	88.24	72.73	76.47	63.64	84.61	75	77.78	68.57	82.76	72.41
Performance Avg	78.12	70	72.49	68.31	70.77	57.79	67	72.33	70	63.90	72	68.90

3. **Performance of direct vs. indirect relationships -** As pointed out earlier, indirect inheritance has produced better results as compared to direct inheritance. This shows that when some existing class and the orphan are inherited from the same class, it is reasonable to place the two classes in the same subsystem. In case of containment, direct containment has produced better results than indirect containment for the orphan adoption algorithm, as well as kNN and Bayesian classifiers. This indicates that when the orphan is contained within an existing class or vice versa, this is a stronger and more meaningful relationship than the orphan and some existing class sharing an attribute. Function call results are almost the same for direct and indirect, and worse than those of Inheritance and Containment despite their larger number. This indicates that function calls may not be useful to place orphans in their appropriate subsystems.

Overall, it can be seen that misclassification rate is high. This is because certain relationships, though higher in number, do not convey meaningful information for placement of an orphan in its appropriate subsystem. This is an indicator that it is important to identify relevant relationships, and that semantic details may be required in addition to the structural criteria adopted by the above approaches.

6.2 Result Discussion for Case II (x=y)

In this case, equal number of dependencies exist from existing subsystems to orphan resource o_1. Since this is a special case where the number of dependencies is equal, we first had to identify such orphans within the test systems. It was observed that within our test systems, the number of orphans with ties between subsystems are few even when relationships are combined and considered. For example, in case of Jfreechart we found only four such orphans. Moreover, for two test systems i.e. Junit and Pmd we found no orphan that results in a tie. Therefore, in order to evaluate our techniques for case II more effectively, we generated orphans with ties. We adopted the following procedure for this purpose:

- We randomly selected 20 orphans for each test system.
- For each of these orphans, we added dependencies to create a tie. For example, if some orphan *o1* has one dependency with *s1* and two dependencies with *s2*, we added a dependency between *o1* and *s1*. This results in a tie between *s1* and *s2.*

Table 6. Experimental results for Case II (x==y)

System	No. of orphans	No. of Orphans accurately classified		Misclassifications	
		Tie Breakers	Neural Network	Tie Breakers	Neural Network
JHotdraw (actual)	12	4	7	66.67	41.67
JHotdraw (generated)	20	8	11	60	45
JHotdraw (overall)	32	12	18	63	43
Jedit (actual)	6	3	2	50	66.67
Jedit (generated)	20	9	8	55	60
Jedit (overall)	26	12	10	53	63.30
Jfreechart (actual)	4	1	3	75	25
Jfreechart (generated)	20	7	11	65	45
Jfreechart (overall)	24	8	14	70	35
Pmd (actual)	0	-	-	-	-
Pmd (generated)	20	3	9	85	55
Junit (actual)	0	-	-	-	-
Junit (generated)	18	6	10	77.78	44.44
Jabref (actual)	6	1	3	83.30	50
Jabref (generated)	20	0	8	100	60
Jabref (overall)	26	1	11	91.65	55
Average Misclassifications				73.40	49.29

We then performed experiments for the actual and generated orphans. Results are presented in Table 6. Please note that in this case we have not considered individual relationships, rather we have combined them together in order to find orphans with ties (all direct and indirect).

Once again, it can be seen that results of Bayesian classifier, k-Nearest-Neighbor classifier, and orphan adoption algorithm are identical. To understand the reason, consider Case II in Fig. 5. For Bayesian classification, probabilities will be calculated as:

$p(s1|o1)=p(o1|s1)p(s1)=\frac{2}{6}*\frac{6}{12}=0.16$

$p(s2|o1)=p(o1|s2)p(s2)=\frac{2}{6}*\frac{6}{12}=0{:}16$

Thus there is a need to use tie breakers as posterior probabilities for both subsystems are equal.

k-Nearest-Neighbor classification using NNall criteria also results in a tie between s_1 and s_2, for which we will have to use tie breakers.

In case of the Orphan Adoption Algorithm, weight of dependencies for subsystems s_1 and s_2 is $W(s_1, o_1) = 2, W$

$(s_2, o_1) = 2$ which implies, $W(s_1, o_1) = W(s_2, o_1)$

The orphan adoption algorithm recommends using tie breakers if more than one subsystems exhibit equal weightage. Therefore in this case tie breakers are applied. We have two candidate subsystems s1 and s2. Using the criteria specified in Section 3.2:

1. Interfaces for candidate subsystems are defined as, *I(s1) = {r3} = 1, I(s2) = {} = 0.* After accommodation of orphan o1 in s1 its interface will become *I(s1) = {r3, o1} = 2.* If o1 is placed in s2 then its interface will be, *I(s2) = {o1} = 1*. It can be seen that cardinality of interfaces for both subsystems increases. Therefore, appropriate subsystem for *o1* cannot be identified using this criterion.

2. Let us define the supplier set for candidate subsystems. *Sup(s1) = {} = 0, Sup(s2) = {s1} = 1.* When o1 is assigned to subsystem s1 then *Sup(s1) = {} = 0.* If accommodated in s2 then *Sup(s2) = {s1} = 1* which shows that the supplier set for both subsystems remains unchanged.

Therefore tie breakers are unable to find one subsystem to which *o1* belongs, therefore it will be arbitrarily placed in one of the candidate subsystems.

To summarize, all three algorithms will use tie breakers in situations where x = y, i.e., there are equal number of dependencies from existing subsystems to the orphan resource.

It is relevant to note that the dependencies between resources within subsystems are not given any significance by the Bayesian classifier, the orphan adoption algorithm or the k-Nearest Neighbor during orphan adoption. However these dependencies may be important for subsystem identification. Therefore, we need to examine such techniques which can utilize the dependencies between the resources within existing subsystems in order to find some pattern for an appropriate subsystem in case of equal number of dependencies (Case II).

For example, consider Fig. 6. Its tabular representation can be seen in Table 7.

In Fig. 6, we have three subsystems *s1, s2* and *s3* and one orphan resource o. It can be seen that the orphan resource o has equal number of dependencies with the subsystems *s1* and *s2* and one dependency with subsystem *s3*. In this case, the orphan adoption algorithm, Bayesian learning and k-Nearest Neighbor will simply indicate a tie between *s1* and *s2*. The Interface and Supplier criteria (Section 3.2) for tie breakers when applied to this problem fail to resolve the tie.

In Fig. 6, dependencies between resources of existing subsystems indicate that if some resource has dependencies with *r2* or *r3* then appropriate subsystem for that resource may be s1. Similarly, if some resource has dependency with *r5*, then that resource may be accommodated in *s2*. A pattern which can be clearly seen is that orphan resource o has dependency with *r2* and *r3*. Therefore, orphan resource o should be accommodated in *s1*. It should be explored whether other classification techniques can make use of intradependencies.

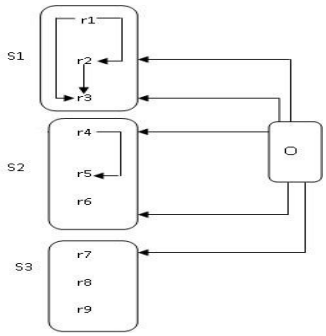

Fig. 6. Intradependencies between resources of subsystems

For this case, Neural Networks were trained using intradependencies as features. It can be seen from Table 6

that results for Neural Networks are better than for tie breakers for all systems except Jedit. It is also interesting to note that the results are also significantly better than for all direct and indirect relationships in Case I. Results of Neural Networks indicate that they are capable of detecting patterns that other classifiers, i.e. Bayesian and KNN, were unable to detect. Comparatively low misclassifications for Neural Networks also indicate that intradependencies between resources of existing subsystems can play a role and should be used to resolve ties.

Table 7. An example for classification techniques and orphan adoption algorithm

Existing Resources	r1	r2	r3	r4	r5	r6	r7	r8	r9	subsystems
r1	0	1	1	0	0	0	0	0	0	**S1**
r2	0	0	1	0	0	0	0	0	0	**S1**
r4	0	0	0	0	1	0	0	0	0	**S2**
0	0	1	1	1	0	1	1	0	0	?

6.3 Strengths and Weaknesses of Individual Classifiers

After detailed analysis of experimental results we conclude that individual classifiers (Bayesian, kNN and NN) have their own strengths and weaknesses for Case I ($x \neq y$) and Case II ($x == y$):

- For case I, where ($x \neq y$), performance of Bayesian and kNN classifier is exactly same. In comparison with Neural Network, it can be seen that for certain relationships Bayesian and kNN give better results and for some relationships NN is better. No one classifier seems to be preferable over the other based on the results in Table 5. However, Bayesian and kNN are relatively simple classifiers and computationally less expensive as compared to Neural Network.

- For case II where ($x == y$), it can be seen from Table 6 that Neural Network perform better as compared to Bayesian and kNN classifiers. In this case Neural Network discovers hidden patterns for appropriate orphan adoption. Bayesian results in equal probabilities for more than one subsystem and kNN results in equal number of nearest neighbors in this case. These classifiers are unable to detect hidden patterns for appropriate placement of orphans. Therefore, tie breakers are used to resolve the tie.

  We propose to combine strengths of individual classifiers (Bayesian, kNN and NN) rather to use individual classifier for both cases. For this purpose, we propose a Delegation/ Cooperative approach.

7.0 PROPOSED DELEGATION/COOPERATIVE APPROACH

The multiclassifier approach aims to combine a set of classifiers. Predictions of classifiers are combined in order to improve classification accuracy [21, 22, 61, 62]. Various studies discovered that classification accuracy can be improved by using features and classifiers of different types [63, 64, 65]. In [22], it is highlighted that strengths of individual classifiers can be combined in order to improve accuracy. Various combination schemes have been proposed in this regard e.g. series and parallel classifiers [21].

Delegating classifiers have also been proposed to combine the strengths of classifiers [23]. These classifiers are used in such a way that one classifier classifies those instances for which its confidence level is above a certain threshold. Those instances which are difficult to classify by first classifier are delegated to another classifier. For delegation, it is required to define a threshold (confidence) or decision rules which make a decision as to for

which instances the first classifier is to be applied and for which instances the other classifier(s) is to be applied. In other words, the first classifier contributes for the subset of input for which it is confident and rejects the rest which are then delegated to the other classifier(s). Therefore, delegated classifiers work in a cooperative way in order to improve classification accuracy [23].

In this section, we discuss our proposed delegated/cooperative framework. As shown in Fig. 7, Classifier 1 will classify certain instances and delegate difficult instances to Classifier 2. Results for both classifiers are combined at the end.

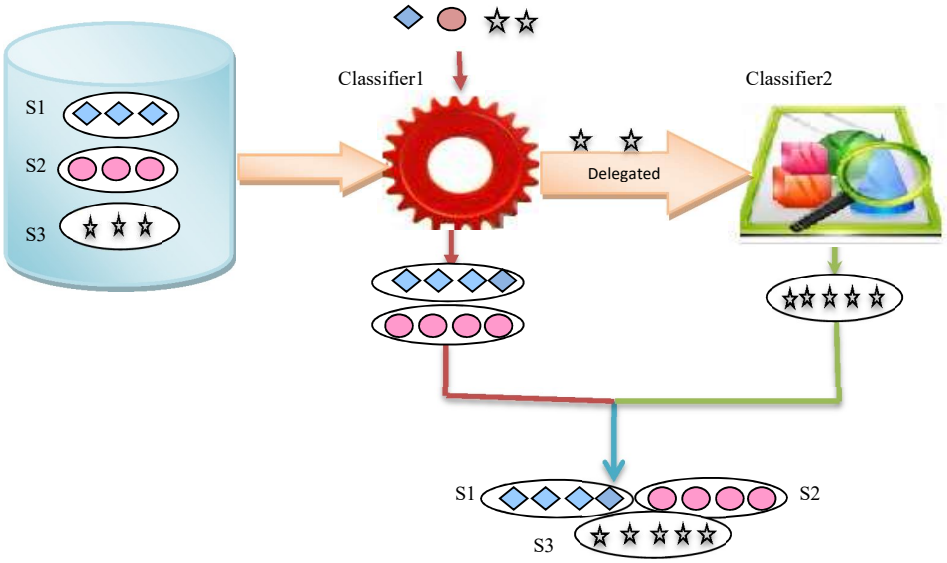

Fig. 7. Delegated/Cooperative framework for orphan adoption

For the orphan adoption problem, Classifier1 will classify those orphans for which it is confident and reject other orphans. In our case, we formulate a decision rule that Classifier1 will classify those orphans for which it finds x≠y . Therefore, Bayesian or kNN can be used as Classifier1, as for x≠y, performance of both classifiers is the same and NN do not offer any clear advantage over these classifiers. Those orphans for which Classifier1 finds equal probability (Bayesian) or equal number of neighbors (kNN) for more than one subsystem can only be placed randomly or by using tie breakers. At this stage, we can take advantage of a Neural Network which is shown to identify hidden useful patterns for appropriate placement of orphans for which x==y. Therefore, we combine classifiers in such a way that Classifier1 delegates orphans with ties to Classifier2 (Neural Network) [1].

We conducted experiments for comparing performance of our proposed delegating classifiers (DC), and individual techniques i.e the Orphan adoption algorithm (OAA) and Neural Networks (NN) [2]. For this purpose, we consider all orphans in a test system (both for x≠y and x==y) as an input to OAA, a NN and our proposed approach. OAA uses structural criteria (Section 3.1) for x≠y Tie breakers (Section 3.2) for the cases where x==y. The results for OAA, NN and proposed cooperative approach are presented in Table 8. We report our findings as follows:

[1] *We selected these classifiers as classifier1 (Bayesian or kNN) and classifier2 (NN) on the basis of their characteristics. However our approach is general, i.e. any classifier can be examined and selected as classifier1 or classifier2 based on its characteristics for this and other research problems.*

[2] *Experiments are not conducted for Bayesian and kNN as these classifiers are used to classify those orphans for which x≠y. Therefore, their results will be same as reported in Table 5 for this case.*

1. **Comparison between delegating classifiers, OAA and NN-** Our proposed delegation approach performs better than individual techniques i.e. OAA and NN. For OAA, performance of the proposed approach is better for all relationships as shown in Table 8. For NN, performance of the proposed approach is better for 10 out of 12 relationships. Thus in all cases, misclassifications are reduced through the proposed approach. This clearly shows that utilizing more than one classifier in a cooperative manner is useful for orphan adoption. Moreover, when information cannot be obtained from interdependencies, intradependencies between resources are important, and classifiers that can utilize patterns based on the intradependencies (e.g. in this case Neural Networks) can be employed to reduce misclassifications. Since the orphan adoption algorithm is unable to identify such patterns, it results in more misclassifications. It is interesting to note that for some relationships, e.g. All indirect containment for pmd test system, behavior of OAA and delegating classifier is similar. This is due to the fact that no such orphan exists which has equal number of dependencies (x==y). In this situation, orphans will not be delegated to Classifier2.

Table 8. Experimental results for OAA, Neural Network (NN) and Delegating Classifiers (DC)

System	All Inheritance			All Direct Inheritance			All Indirect Inheritance			All Containment			All Direct Containment			All Indirect Containment		
	OOA	NN	DC	OOA	NN	DC	OOA	NN	DC	OOA	NN	DC	OOA	NN	DC	OOA	NN	DC
JHotdraw	58	58	47	68.42	63	52.6	44.44	44	22.22	76.19	57	52.38	66.67	71	66.67	64.28	43	28.57
Jedit	52.6	63	36	63.15	58	47.36	22.22	44	11.11	69.23	62	53.84	53.84	54	38.46	75	75	50
Jfreechart	27	55	0	27.27	45	9.09	12.5	50	0	75	50	75	75	63	75	85	57	85
Pmd	14	57	14	16.67	57	16.67	33.33	50	33.33	75	86	75	80	71	80	80	57	80
Junit	60	80	60	60	80	60	33	66	33	0	100	0	0	100	0	0	0	0
Jabref	50	50	41	83.33	83	75	41.66	50	33.33	83	100	83	83	100	83	100	83	100
Performance Avg	43.6	60	33	53.14	64	43.45	36.39	50	22.16	63.07	76	56.53	59.75	76	57.18	67.38	52	57.26
System	All Function calls			All Direct Function calls			All Indirect Function calls			All Direct			All Indirect			All Direct and Indirect		
	OOA	NN	DC	OOA	NN	DC	OOA	NN	DC	OOA	NN	DC	OOA	NN	DC	OOA	NN	DC
JHotdraw	91.07	84	82	89.28	87	80.35	73.17	71	63.41	74	70	65.5	57	58	57	74	68	66
Jedit	67.5	63	57.5	62.5	70	57.5	58.06	65	48.38	71.43	72	71.43	63.38	63	56.33	71.83	75	64.78
Jfreechart	86.95	78	73.91	100	87	86.95	100	81	86.95	78.26	79	78.26	75	76	75	68.96	65	55.55
Pmd	50	55	50	27	55	27	50	60	50	25	50	25	70	65	70	63.33	67	63.33
Junit	88.23	76	76.47	58.82	82	47.07	61.53	54	53.84	64	65	64	78.57	62	78.57	75	72	75
Jabref	90.90	77	68.18	90.90	77	68.18	81.81	56	59.09	85	74	77	80	67	69.44	83.33	75	72.22
Performance Avg	79.10	72	68.01	71.41	76	61.17	70.76	64	60.27	66.28	68	63.53	70.6	65	67.72	72.74	70	66.14

2. **Performance of Delegating/Cooperative Approach-** A comparison between the misclassifications presented in Table 8 and Table 5 reveals some interesting facts. It can be seen that misclassification rate is higher for the OOA and NN in Table 8 as compared to Table 5 for almost all relationships**. This suggests that although OOA is useful for the simple case (**$x \neq y$), when orphans with ($x=y$) are added to the dataset, OOA's performance deteriorates instead of improving. Similar behavior is seen in the case of Neural Networks, which although performs well for the case ($x=y$), cannot perform well when the dataset contains both inter and intradependency information. On the other hand, it can be seen from the misclassification rate for the delegating approach that there is improvement in the results in Table 8 as compared to individual classifiers (Table 5) for all relationships except three, i.e. All indirect containment, All indirect function calls and All indirect relationships. Thus the proposed delegating approach is capable of making use of the orphan information to reduce misclassifications.

3. **Performance of Relationships-** As far as performance of relationships is concerned**,** it can be seen from Table 8 that on the average, the inheritance relationship has performed better than other relationships. This is similar to the result reported in Table 5.

8.0 CASE III: NEW SUBSYSTEM CREATION

We present another situation in Fig. 8. It can be seen that there is no dependency between existing resources and orphan resource *o1*.

In this case the Bayesian classifier will result in zero probabilities as calculated below:

$p(s1|o1)=p(o1|s1)p(s1)=\frac{0}{6}*\frac{6}{12}$

$p(s2|o1)=p(o1|s2)p(s2)=\frac{0}{6}*\frac{6}{12}$

In order to avoid such 0 probabilities Laplace correction is used. Using Laplace correction [17]:

$p(s1|o1)=p(o1|s1)p(s1)=\frac{0+1}{6+2}*\frac{6}{12}=0.06$

$p(s2|o1)=p(o1|s2)p(s2)=\frac{0+1}{6+2}*\frac{6}{12}=0.06$

Posterior probabilities for *s1* and *s2* are equal, which implies that there is a tie between these subsystems. Tie breakers (Section 3.2) are applied to resolve the issue.

For k-Nearest Neighbor, no nearest resource is found as no resource is dependent on o_1. In this situation if we use tie breakers then k-Nearest-Neighbor will behave like the orphan adoption algorithm, if we use Laplace correction [16] then it will behave like the Bayesian classifier.

For the Orphan Adoption Algorithm, weight of dependencies for both subsystems will be 0 (equal), i.e., $W(s_1, o_1) = W(s_2, o_1)$ Therefore in this case tie breakers are applied (Section 3.2).

In all such cases where x = 0 and y = 0, i.e., there is no dependency from existing resources to orphan resource, Bayesian classification will apply Laplace correction. Laplace correction will result in a tie between subsystems in all such cases where the subsystems contain equal number of resources. Otherwise for orphan resource, Laplace correction discovers the subsystem which contains the highest number of resources. The Orphan adoption algorithm will apply tie breakers in this case. k-Nearest-Neighbor classifier will behave like the orphan adoption algorithm if we use tie breakers. If we use Laplace correction then it will behave like the Bayesian classifier.

Handling Case III

Case III refers to the fact that entities in a system have independent existence. Such independent existence of an entity seems unlikely as it is very unusual that an entity is completely isolated (i.e. does not relate with other entities in a system). Moreover, if such entities exist in a system then supervised learning techniques will not work for their placement in existing subsystems. This is due to the reason that dependency information between orphans and existing subsystems, which is utilized by supervised learning techniques, does not exist. It is also relevant to note that it may not always be the case that appropriate subsystems for newly introduced resources are the existing subsystems. There are some situations where there is a need to create new subsystems for accommodation of new resources. Carmichael et al. [66] highlighted the need for creating new containers in order to place resources for which no appropriate subsystems are found. This case has not been explored in the orphan adoption algorithm. We propose the use of this idea of new subsystem creation for Case III.

We can generalize the case by introducing the concept of threshold which is defined as the number of dependencies that exist between orphan classes and existing classes. For example, threshold 0 means that orphan class has no dependency with existing classes. However, the threshold may be changed to accommodate system characteristics that vary from system to system. Therefore, we may define it in terms of the average number of dependencies with orphan classes within the system. Having defined the threshold either as average dependency or zero dependency, it can be used as follows:

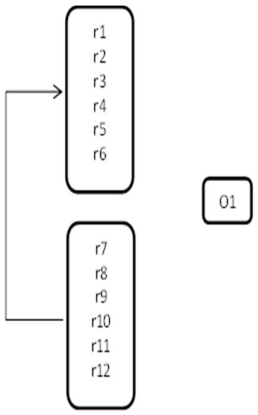

Fig. 8. Zero dependency scenario

Separate all orphan classes having dependencies less than the threshold. Now examine the dependencies that exist between separated orphan classes. All orphan classes which are similar with each other (number of dependencies is greater than or equal defined threshold) are grouped together in a new subsystem. All remaining orphans (with interdependencies less than threshold) will be accommodated:

i) existing subsystems according to structural criteria, or
ii) into a separate new subsystem.

9.0 THREATS TO VALIDITY

In this section we discuss the threats that may affect our work and results, and how we have tried to reduce the threats.

Internal validity

We have shown the similar behavior of supervised learning techniques and orphan adoption algorithm analytically. Thus the similar results of the three techniques cannot be by chance. We acknowledge that for Case II where x = y, the number of orphans was not large, thus to check the behavior of Neural Networks, we randomly modified orphan details to create the required scenario. Since the additions/deletions for this purpose were random, we expect that Neural Networks will in general perform better and their better performance is not due to chance or any particular characteristic of the test systems. This is also supported through the results of the multiclassifier approach where we did not change any orphan particulars (no additions/deletions), yet the approach gave better results.

Construct validity

We have used well known relationships that are present in object oriented systems. These relationships have been divided into direct and indirect for better understanding and analysis of results. The misclassification measure is an objective measure to indicate the number of misplaced orphans. Some element of subjectivity is introduced during evaluation, since evaluation of results of our approach is carried out through comparison with an expert decomposition prepared manually. However, to reduce problems, expert decompositions for software systems Jhotdraw, Jedit, Jfreechart, and Pmd were obtained using the directory structure as in [28], [29]. For Junit and Jabref we requested a human expert to develop the decompositions. The expert had experience in object oriented development and had a thorough understanding of architectural concepts. He was also given sufficient time to understand the systems and form the decompositions.

External validity

For our experiments we selected six software systems written in Java. We selected common relationships

between classes for these systems e.g. inheritance, containment, which are expected to be present in any object oriented system and are not specific to Java. The test systems are reasonably large, with the average number of classes in the considered version and subsequent version being 452 and 491. The average number of orphans in the subsequent version is 44. We acknowledge that the number of orphans is not large when we consider the case x == y (case II). However, we have tried to randomly generate orphans for this case to evaluate it more effectively (6.2).

10.0 CONCLUSIONS AND FUTURE WORK

The orphan adoption problem is a significant problem faced when recovering the architecture of evolving software systems. Clustering, an unsupervised machine learning technique has been focused for software architecture recovery. However, clustering techniques usually do not utilize the existing architectural documentation for a certain version even if available for recovering the architecture of subsequent versions of evolving software systems. In this paper, we employed supervised learning techniques (classifiers) for the orphan adoption problem in order to aid the architecture recovery of evolving software systems.

Supervised learning techniques avail existing architectural information to appropriately adopt the orphan resources (newly introduced resources) in subsystems in subsequent versions of software systems. We conducted experiments on six Java systems having multiple versions and analyzed the correspondence between orphan adoption algorithm and supervised learning techniques. Our experimental results reveal the strengths of the various classifiers and also indicate that intradependencies (dependencies between entities within subsystems) may be important in addition to interdependencies (dependencies between entities among subsystems). Based on the identified strengths, we proposed a delegated/cooperative approach in which more than one classifier cooperate to solve the orphan adoption problem. We conducted experiments to evaluate the delegated approach and found significant improvement in orphan classification accuracy. We also recommended a criterion for new subsystem creation.

In the future, we intend to explore semantic relationships that exist in a software system and study their performance. Moreover, we aim to use other classifiers in our proposed framework e.g. Support vector machine.

REFERENCES

[1] M. Lehman. “Laws of software evolution revisited”. *Proceedings of the European Workshop on Software Process Technology*, pp. 108-124, 1996.

[2] J.F. Ramil and M.M. Lehman. “Effort estimation from change records of evolving software”. *Proceedings of the International Conference on Software Engineering*, pp. 777, 2000.

[3] R. Vasa, J.G. Schneider, and O. Nierstrasz. “The inevitable stability of software change”. *Proceedings of the International Conference on Software Maintenance*, pp. 4-13, 2007.

[4] S. Ali and O. Maqbool. “Monitoring software evolution using multiple types of changes”. *Proceedings of the International Conference on Emerging Technologie*s, pp. 410-415, 2009.

[5] V. Tzerpos and R.C. Holt. “The orphan adoption problem in architecture maintenance”. *Proceedings of the Working Conference on Reverse Engineering*, pp. 76-82, 1997.

[6] V. Tzerpos and R.C. Holt. “ACDC: an algorithm for comprehension-driven clustering”. *Proceedings of the Working Conference on Reverse Engineering*, pp. 258-267, 2000.

[7] B.S. Mitchell and S. Mancoridis. “On the automatic modularization of software systems using the bunch tool”.
IEEE Transactions on Software Engineering, Vol. 32, No. 3, pp.193-208, 2006.

[8] C. Patel, A. Hamou-Lhadj, and J. Rilling. “Software clustering using dynamic analysis and static dependencies”. *Proceedings of the European Conference on Software Maintenance and Reengineering*, pp. 27-36, 2009.

[9] PBStookit. Pbs toolkit. http://www.swag.uwaterloo.ca/pbs/ , 2011.

[10] A.E. Hassan and R.C. Holt. "Towards a better understanding of web applications". *Proceedings of the Interna-tional Workshop on Web Site Evolution*, pp. 112-116, 2001.

[11] Zhao, Junfeng and Zhou, Jiantao and Yang, Hongji, "Modularizing Legacy System through an Improved Bunch Clustering Method in Cloud Migration", *International Journal of Grid and Distributed Computing,* Vol. 8, No. 4, pp. 1-10, 2015

[12] T.M. Mitchell. 'The discipline of machine learning". *Technical Report CMU-ML-06-108, Machine Learning Department , Carnegie Mellon University*, 2006.

[13] A.C. Tan and D. Gilbert. "An empirical comparison of supervised machine learning techniques in bioinformatics". *Proceedings of the Asia-Pacific Bioinformatics Conference on Bioinformatics*, pp. 219-222, 2003.

[14] M. Giardina, F. Azuaje, P. McCullagh, and R. Harper. "A supervised learning approach to predicting coronary heart disease complications in type 2 diabetes mellitus patients*". Proceedings of the Symposium on BioInfor-matics and BioEngineering*, pp. 325-331, 2006.

[15] S. Maes, K. Tuyls, B. Vanschoenwinkel, and B. Manderick. "Credit card fraud detection using bayesian and neural networks". *Proceedings of the 1st International NAISO Congress on Neuro Fuzzy Technologies*.

[16] I.H. Witten and E. Frank. "Data Mining: practical machine learning tools and techniques". Morgan Kaufmann, 2005.

[17] L. B. Huang, V. Balakrishnan, R.G. Raj, "Improving the relevancy of document search using the multi-term adjacency keyword-order model." Malaysian Journal of Computer Science, Vol. 25, No. 1, pp. 1-10, 2012.

[18] O. Maqbool and HA Babri. "The weighted combined algorithm: A linkage algorithm for software clustering".
Proceedings of the European Conference on Software Maintenance and Reengineering, pp. 15-24, 2004.

[19] P. Andritsos and V. Tzerpos. "Information-theoretic software clustering". *IEEE Transactions on Software Engineering*, Vol. 31, No. 2, pp.150-165, 2005.

[20] Q. Zhang, D. Qiu, Q. Tian, and L. Sun. "Object-oriented software architecture recovery using a new hybrid clustering algorithm". *Proceedings of the International Conference on Fuzzy Systems and Knowledge Discovery*, pp. 2546-2550, 2010.

[21] R.G. Raj, S. Abdul-Kareem, "Information Dissemination And Storage For Tele-Text Based Conversational Systems' Learning", Malaysian Journal of Computer Science, Vol. 22(2), pp. 138-159, 2009.

[22] M. A. Shayegan, S. Aghabozorgi, R. G. Raj, "A Novel Two-Stage Spectrum-Based Approach for Dimensionality Reduction: A Case Study on the Recognition of Handwritten Numerals," Journal of Applied Mathematics, vol. 2014, Article ID 654787, 14 pages, 2014. doi:10.1155/2014/654787

[23] Ferri, Cesar and Flach, Peter and Herandez-Orallo, Jose, "Delegating classifiers", *Proceedings of the twenty-first international conference on Machine learning,* pp. 37, *2004.*

[24] Stephane Ducasse and Damien Pollet. "Software architecture reconstruction: A process-oriented taxonomy". *IEEE Transactions on Software Engineering*, Vol. 35, No. 4, pp.573-591, 2009.

[25] S. Mancoridis, B.S. Mitchell, C. Rorres, Y. Chen, and E.R. Gansner. "Using automatic clustering to produce high-level system organizations of source code". *Proceedings of the International Workshop on Program Comprehension*, pp. 45-52, 1998.

[26] Vanderlei, Taciana A and Durao, Frederico A and Martins, Alexandre C and Garcia, Vinicius C and Almeida, Eduardo S and de L Meira, Silvio R, "A cooperative classification mechanism for search and retrieval software components", *Proceedings of the 2007 ACM symposium on Applied computing,* pp. 866-871, 2007

[27] M. Saeed, O. Maqbool, HA Babri, SZ Hassan, and SM Sarwar. "Software clustering techniques and the use of combined algorithm". *Proceedings of the European Conference on Software Maintenance and Reengineering*, pp. 301-306, 2003.

[28] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico. "Using the kleinberg algorithm and vector space model for software system clustering". *Proceedings of the International Conference on Program Comprehension*, pp. 180-189, 2010.

[29] R.G. Raj and S. Abdul-Kareem. "A Pattern Based Approach for The Derivation Of Base Forms Of Verbs From Participles And Tenses For Flexible NLP". Malaysian Journal of Computer Science, Vol. 24(2), pp 63-72, 2011.

[30] Joshua Garcia, Igor Ivkovic, and Nenad Medvidovic. "A comparative analysis of software architecture recovery techniques. In Automated Software Engineering (ASE)", *International Conference on Automated Software Engineering (ASE),* pp. 486-496. IEEE, 2013.

[31] Anna Corazza, Sergio Di Martino, and Giuseppe Scanniello. "A probabilistic based approach towards software system clustering". *In European Conference on Software Maintenance and Reengineering (CSMR)*, pp. 88-96. IEEE, 2010.

[32] Joshua Garcia, Daniel Popescu, Chris Mattmann, Nenad Medvidovic, and Yuanfang Cai. "Enhancing architectural recovery using concerns". *International Conference on Automated Software Engineering IEEE/ACM*, pp. 552-555, 2011.

[33] Gustavo Santos, Marco Tulio Valente, and Nicolas Anquetil. "Remodularization analysis using semantic clustering". *In the European Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE),* pp. 224-233. IEEE, 2014.

[34] Naseem, Rashid and Maqbool, Onaiza and Muhammad, Siraj, "Cooperative clustering for software modularization", *Journal of Systems and Software,* Vol. 8, No. 86, pp. 2045—2062, 2013

[35] Kashef, Rasha, "Cooperative clustering model and its applications", Ph.D. dissertation, 2008

[36] Brian S Mitchell. "A heuristic approach to solving the software clustering problem. In International Conference on Software Maintenance", 2003. *In the European Conference on Software Maintenance*, pp. 285-288. IEEE, 2003.

[37] Shaheda Akthar and Sk MD Ra. "Recovery of software architecture using partitioning approach by edler vector and clustering". *Computer and Information Science*, 3(1): pp. 72, 2010.

[38] K. Sartipi. Software architecture recovery based on pattern matching. Proceedings of the International Conference on Software Maintenance, pp. 293-296, 2003.

[39] Mark Harman, Robert M Hierons, and Mark Proctor. "A new representation and crossover operator for search-based optimization of software modularization". *In Proceedings of the Genetic and Evolutionary Computation Conference*, Vol. 2, pp. 1351-1358, 2002.

[40] Kata Praditwong, Mark Harman, and Xin Yao. "Software module clustering as a multi-objective search problem". *IEEE Transactions on Software Engineering*, Vol. 37, No. 2, pp. 264-282, 2011.

[41] C. Tjortjis, L. Sinos, and P. Layzell. "Facilitating program comprehension by mining association rules from source code". *Proceedings of the International Workshop on Program Comprehension,* pp. 125-132, 2003.

[42] O. Maqbool and HA Babri. "Bayesian learning for software architecture recovery". *Proceedings of the*

Interna-tional Conference on Electrical Engineering, pp. 1-6, 2007.

[43] Wolpert, David H, The lack of a priori distinctions between learning algorithms, *Neural Computation*, Vol. 8, No. 7, pp. 1341—1390, MIT Press, 1996

[44] Trawinski, Krzysztof and Cordon, Oscar and Quirin, Arnaud, Random oracles fuzzy rule-based multiclassifiers for high complexity datasets, *International Conference on Fuzzy Systems (FUZZ),* pp. 1-8, 2013

[45] H. Kagdi. "Improving change prediction with ne-grained source code mining". *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pp. 559-562, 2007.

[46] A. Wierda, E. Dortmans, and L. Somers. "Using version information in architectural clustering-a case study".
Proceedings of the European Conference on Software Maintenance and Reengineering, pp. 214-228, 2006.

[47] B. Liu. Web data mining. Springer, 2007.

[48] T.M. Mitchell. Machine learning. Mc Graw Hill, 1997.

[49] WebsiteJHotdraw. Jhotdraw source system. http://www.jhotdraw. org/, 2011.

[50] WebsiteJedit. Jedit source system. http://www.jedit.org/, 2011.

[51] WebsiteJfreechart. Jfreechart source system. http://www.jfree.org /jfreechart/, 2011.

[52] WebsitePmd. Pmd source system. http://www.pmd.sourceforge.net/, 2011.

[53] WebsiteJunit. Junit source system. http://www.junit.org/, 2011.

[54] WebsiteJabref. Jabref source system. http://www.jabref. sourceforge.net/, 2011.

[55] M. Trifu. Architecture-aware, adaptive clustering of object oriented systems. Master's thesis, University of Timisoara.

[56] WebsiteInfusion. Infusion parser. http://www.intooitus.com/, 2011.

[57] Guoqiang Zhang, B. Eddy Patuwo, Michael Y. Hu , "Forecasting with artificial neural networks: The state of the art", *International Journal of forecasting,* Vol. 14, No. 1, pp. 35-62, 1998

[58] Warner, Brad and Misra, Manavendra , "Understanding Neural Networks as Statistical Tools", *The american statistician,* Vol. 50, No. 4, pp. 284-293, 1996

[59] *"*Tuning of the Structure and Parameters of Neural Networks using an Improved Genetic Algorithm", *IEEE transactions on Neural Networks,* Vol. 14, No. 1, pp. 79-89, 2003

[60] Siraj Muhammad, Onaiza Maqbool, Abdul Qudus Abbasi, "Evaluating Relationship Categories for Clustering Object-Oriented Software Systems", *IET Software*, 6(3): pp. 260-274, June 2012

[61] Valentini, Giorgio and Masulli, Francesco. "Ensembles of learning machines". *LNCS, Springer*, pp. 3-20, 2002

[62] A. Qazi, H. Fayaz, A. Wadi, R. G. Raj, N.A. Rahim, W. A. Khan, "The artificial neural network for solar radiation prediction and designing solar systems: a systematic literature review", Journal of Cleaner Production, vol. 104, pp. 1-12, 2015. ISSN 0959-6526, http://dx.doi.org/10.1016/j.jclepro.2015.04.041.(http://www.sciencedirect.com/science/article/pii/S0959652615004096)

[63] R. Bradford and T. Nartker, "Error correlation in contemporary OCR systems," *1st International. Conference on Document Analysis and Recognition*, Saint-Malo. France, pp. 516-523, 1991.

[64] T. K. Ho, J. J. Hull, and S. N. Srihari, "Combination of structural classifiers", *IAPR Syntactic and Structural Pattern Recognition Workshop*. Murray Hill, NJ, pp. 123-136, 1990.

[65] T. K. Ho, A Theory of Multiple Classifier Systems and Its Application to Visual Word Recognition. Ph.D. dissertation, Dept. of Computer Science. SUNY at Buffalo. 1992

[66] I. Carmichael, V. Tzerpos, and R.C. Holt. "Design maintenance: unexpected architectural interactions (expe-rience report)", *Proceedings of the International Conference on Software Maintenance*, pp. 134-137, 1995.