BABEŞ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

New hybrid Machine Learning models. Applications in Software Engineering

PhD Thesis - Summary

PhD student: Diana-Lucia Miholca
Scientific supervisor: Prof. Dr. Gabriela Czibula

September 2020

Thesis contents

Summary contents

Acronyms

Acc Accuracy. 25

AGRARM Adaptive Gradual Relational Association Rules Miner. 10, 12, 18, 21–23

ANN Artificial Neural Network. 10–15, 24, 25, 27, 29, 35, 36, 42

AR Association Rule. 9, 10, 20–22

ARARM Adaptive Relational Association Rules Miner. 21

ARM Association Rules Mining. 9, 21, 22

AUC Area Under the Receiver Operating Characteristics (ROC) Curve. 25, 27, 37, 38, 40

CBO Coupling Between Objects. 30

DAC Data Abstraction Coupling. 30

DynGRAR Dynamic Gradual Relational Association Rules Miner. 10, 12, 18, 22, 23, 43

ESC Estimation of Software Coupling. 9, 11–14, 29, 31, 34, 35, 42

GRANUM Gradual Relational Association Rules Miner. 10–12, 18, 20, 21, 23, 25

GRAR Gradual Relational Association Rule. 10–14, 18–25, 27, 35, 36, 42, 43

ICH Information Flow Based Coupling. 30

LOO Leave-One-Out. 40

LSI Latent Semantic Indexing. 12, 31, 33, 36, 37, 39

ML Machine Learning. 9–12, 14, 15, 24, 27, 29, 34, 36, 38, 40, 42, 43

MLP Multilayer Perceptron. 10–12, 15, 16, 26, 27, 36

MPC Message Passing Coupling. 30

OAR Ordinal Association Rule. 10

PCA Principal Component Analysis. 30

Prec Precision. 25

RAR Relational Association Rule. 10, 12, 14, 15, 18–21, 23, 42

RBFN Radial Basis Functions Network. 10, 15, 16

SBSE Search Based Software Engineering. 9, 27, 42

SDP Software Defects Prediction. 9–14, 16–18, 21–24, 27, 35, 36, 38–43

SE Software Engineering. 9–12, 14, 17, 35, 42

Sens Sensitivity. 25

SOM Self Organizing Map. 15, 16, 30, 32

Spec Specificity. 25

t-SNE t-Distributed Stochastic Neighbour Embedding. 37, 40

List of Figures

List of publications

All rankings are listed according to the 2014 classification of journals[1] and conferences[2] in Computer Science.

Publications in Web of Science - Science Citation Index Expanded

1. [75] **Diana-Lucia Miholca**, Gabriela Czibula, Istvan Gergely Czibula. A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks. *Information Sciences*, 441, 2018, pp. 152 – 170. (**indexed Web of Science, IF=4.305**)

   Rank A, 8 points.

2. [32] Istvan Gergely Czibula, Gabriela Czibula, **Diana-Lucia Miholca**, Zsuzsanna Onet-Marian. An aggregated coupling measure for the analysis of object-oriented software systems. *Journal of Systems and Software*, 148, 2019, pp. 1 – 20. (**indexed Web of Science, IF=2.278**)

   Rank A, 4 points.

3. [25] Gabriela Czibula, Istvan Gergely Czibula, **Diana-Lucia Miholca**, Liana Maria Crivei. A novel concurrent relational association rule mining approach. *Expert systems with Applications*, 125, 2019, pp. 142 – 156. (**indexed Web of Science, IF=3.768**)

   Rank A, 4 points.

Publications in Web of Science, Conference Proceedings Citation Index

1. [70] **Diana-Lucia Miholca**, Gabriela Czibula, Zsuzsanna Marian and Istvan-Gergely Czibula. An unsupervised learning based conceptual coupling measure. *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2017, pp. 247 – 254. (**indexed Web of Science**)

   Rank C, 1 point.

2. [74] **Diana-Lucia Miholca**, Adrian Onicaș. Detecting depression from fMRI using relational association rules and artificial neural networks. *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, 2017, pp. 85 – 92. (**indexed Web of Science**)

   Rank C, 2 points.

[1] http://informatica-universitaria.ro/getpfile/16/CSafisat2.pdf;http://hfpop.ro/standarde/doctorat/2014-jurnale.pdf

[2] http://informatica-universitaria.ro/getpfile/16/CORE2013_Exported.xlsx;http://hfpop.ro/standarde/doctorat/2014-conferinte.pdf

3. [69] **Diana-Lucia Miholca**. An improved approach to software defect prediction using a hybrid machine learning model. *20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2018, pp. 443 – 448. (**indexed Web of Science**)

   Rank C, 2 points.

4. [74] **Diana-Lucia Miholca**, Gabriela Czibula, Liana Maria Crivei. A new incremental relational association rules mining approach. *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference (KES)*, Belgrade, Serbia, 2018, pp. 126 – 135. (**indexed Web of Science**)

   Rank B, 4 points.

5. [72] **Diana-Lucia Miholca**, Gabriela Czibula. DynGRAR: A dynamic approach to mining gradual relational association rules. *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23th International Conference (KES)*, Budapest, Hungary, 2019, pp. 10 – 19. (**indexed Web of Science**)

   Rank B, 4 points.

6. [73] **Diana-Lucia Miholca**, Gabriela Czibula. Software defect prediction using a hybrid model based on semantic features learned from the source code. *International Conference on Knowledge Science, Engineering and Management (KSEM)*, Athens, Greece, 2019, pp. 262 – 274. (**indexed Web of Science**)

   Rank B, 4 points.

7. [77] **Diana-Lucia Miholca**, Gabriela Czibula, Vlad Tomescu. COMET: A conceptual coupling based metrics suite for software defect prediction. *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference (KES)*, Verona, Italy, 2020, *accepted for publication* (**indexed Web of Science**)

   Rank B, 4 points.

Publications in journals and conference proceedings

1. [31] Istvan Gergely Czibula, Gabriela Czibula, **Diana-Lucia Miholca**. Enhancing relational association rules with gradualness. *International Journal of Innovative Computing, Information and Control* 13(1), 2017, pp. 289 – 305. (**indexed Scopus**)

   Rank B, 4 points.

2. [29] Istvan Gergely Czibula, Gabriela Czibula, **Diana-Lucia Miholca**, Zsuzsanna Marian. Identifying hidden dependencies in software systems. *Studia Universitatis Babeș-Bolyai Informatica*, 62(1), 2017, pp. 90 – 106. (**indexed Mathematical Reviews**)

   Rank D, 0.5 points.

3. [71] **Diana-Lucia Miholca**. An adaptive gradual relational association rules mining approach. *Studia Universitatis Babes-Bolyai Series Informatica*, 63(1), 2018, pp. 94 – 110. (**indexed Mathematical Reviews**)

   Rank D, 1 point.

Publications score: 42.5 points.

Keywords

- Machine Learning
- Hybrid classification models
- Relational Association Rules
- Artificial Neural Networks
- Software Defects Prediction
- Software Coupling
- Conceptual Coupling
- Aggregated Coupling
- Software metrics
- Change Impact Analysis
- Software Restructuring

Introduction

This Ph.D. thesis, titled *New hybrid Machine Learning models. Applications in Software Engineering*, focuses on the development of new Machine Learning models, some of which are hybrid, and on their applicability to solve important Software Engineering problems, in particular Software Defect Prediction and Estimation of Software Coupling.

Software Defects Prediction (SDP) consists in identifying defective software components. It has a broad applicability by assisting measuring project evolution, supporting process management [22], predicting software reliability [117], streamlining testing and guiding code review [46]. All these allow to significantly reduce the costs involved in developing and maintaining software products [45]. Moreover, particularly when it comes to safety-critical systems, SDP helps in detecting software anomalies that might have a negative effect on human lives. Despite its importance and extensive applicability, SDP remains a difficult problem, especially in large-scale complex systems, and a very active research area [43].

Estimation of Software Coupling (ESC) consists in assessing the degree of dependence between software components. There are many types of coupling measures in the literature. The prevalent but not exhaustive ones measure *structural* inter-dependencies [6, 83]. Other types of coupling are *conceptual* coupling [91], *dynamic* coupling and *change* coupling [9]. Knowing how closely related two software components are helps to better understand the software system and therefore facilitates its evolution and maintenance. Particularly, ESC supports change impact analysis [91, 12] which is essential given that a software is subject to frequent change. Software coupling also supports other Software Engineering (SE) activities, such as software restructuring [30, 70], maintainability prediction [89] and regression testing [110].

Both SDP and ESC contribute to software quality assurance which gets more difficult as software systems become more complex. Software defects are effects of poor software quality, while different types of software coupling are properties that strongly relate to the quality of the software design. Design flaws make the software more defects-prone [33]. Consequently, there is likely to be a connection between software coupling, which may indicate design flaws, and software defects. This is why in this thesis we do not approach SDP and ESC only independently, but also in relation to each other.

Machine Learning (ML) [79] is a subdiscipline of Artificial Intelligence (AI). It aims at developing systems that improve their performance through experience in order to enable learning, reasoning and decision making outside of human interaction. Such systems are intended to solve complex problems. Due to the increasing size and complexity of software projects, the problems related to software development are computationally difficult. Consequently, the ML based solutions are appropriate for addressing SE problems, including SDP and ESC. This is confirmed by the major interest in the Search Based Software Engineering (SBSE) literature towards using ML based solutions. From a ML perspective, SDP can be formulated as a binary classification problem that consists in discriminating between defective and defect-free software components, while ESC is also benefited by ML especially when it comes to non-structural coupling [91].

Association Rules Mining (ARM) [93], also called Association Rules Learning, is a rule-based, unsupervised ML method for identifying regularities in data sets [4] [86] [96]. Classical Association Rules (ARs) [104] mining discovers frequent co-occurring attribute-value associations.

So, it does not consider potentially significant relations between the attribute values, excepting their co-occurrences [40]. Ordinal Association Rules (OARs) [15] are specific ARs which express frequent ordinal relations. But informative relations that are not ordinal may also exist between the attribute values. In this context, OARs have been extended to Relational Association Rules (RARs) which express various types of relations, including non-ordinal ones. Compared to classical ARs mining, RARs mining uncovers much more powerful rules. Such rules allow explaining and understanding current data, while ML can be applied further to learn from them how to accurately predict future data [106]. The effectiveness of RARs mining has been proven in various domains including medicine [101], bioinformatics [24] and SE [27, 28]. In SE, RARs mining has been already applied for SDP [27] and software design defects detection [28].

The first original conceptual contribution presented in this thesis is introducing in the ARM literature **Gradual Relational Association Rules (GRARs) mining** [31]. GRARs mining generalizes RARs mining by using gradual relations instead of boolean, non-gradual ones. The gradual relations which are, in fact, fuzzy [54] relations are enriched with the degrees to which they are satisfied, while also inheriting the noise-tolerance of the fuzzy approaches. Consequently, GRARs mining is both more expressive and more stable to noise than non-gradual RARs mining. For unsupervisedly learning interesting (i.e frequent and sufficiently strong) GRARs, we have proposed an algorithm named Gradual Relational Association Rules Miner (GRANUM) [31], which is an adaptation of the Apriori ARs mining algorithm [5].

There are situations when the data sets to be mined are dynamic, given that the sets of attributes can be incrementally expanded as new information become available. In order to keep the sets of interesting GRARs up-to-date in such cases, GRANUM can be reapplied every time the data sets are extended with one or more new attributes. But running GRANUM from scratch could be inefficient, especially if the set of attributes is only slightly expanded. This motivated us to introduce, as a second conceptual contribution, a GRARs learning algorithm named **Adaptive Gradual Relational Association Rules Miner (AGRARM)** [71]. AGRARM is a more time-efficient alternative to resuming GRANUM when new data attributes become available. It adapts the set of GRARs which have been found to be interesting before extension by including new possibly interesting rules involving the newly added attributes.

Dynamic data sets can be updated periodically by adding not only new attributes, but also new instances. To efficiently update in such scenarios the set of interesting GRARs discovered before extension we have proposed, as a third conceptual contribution, an algorithm named **Dynamic Gradual Relational Association Rules Miner (DynGRAR)** [72]. So, DynGRAR extends AGRARM and provides a more efficient alternative to the resumption of GRANUM algorithm when the data set of interest is expanded with new attributes, new instances or both of them. There are numerous application domains involving dynamic data. One example is SE, due to the intrinsic dynamic nature of the software development process.

Our fourth fundamental research contribution presented in this thesis is **developing new hybrid ML models by hybridizing GRARs mining with Artificial Neural Networks (ANNs)**. In a first hybrid classification model, named **HyGRAR** [75], the ANNs, in particular Multilayer Perceptrons (MLPs) and Radial Basis Functions Networks (RBFNs), supervisedly learn binary gradual relations between attributes. Each of these relations estimate to what extent a combination of two values for a pair of two attributes places an instance in one of the possible classes. The subsequent GRARs mining process, performed by GRANUM, discovers the interesting GRARs defined using these gradual relations. The interesting GRARs of any length are mined from the training subsets corresponding to each of the classes. Thus, they express the particularities of each class and form a basis for classification.

By **proposing HyGRAR as a solution to Software Defects Prediction (SDP)** [75] and thus proving its practical applicability, we have also contributed to the applied research in SE. Our proposal was motivated by the intuition that relations between the values of relevant

software metrics may indicate a software entity's defect proneness. Accordingly, the first step in applying HyGRAR for predicting software defects is to learn such relations by training ANNs on labeled data generated by previous software projects or previous versions of a project with a current version under development. The gradual relations between pairs of software metrics are then used by GRANUM to discover interesting GRARs that discriminate between the defective and the non-defective software components of the analyzed system. Finally, in the classification phase, a predefined, heuristic method is used to classify a new software component as defective or non-defective, based on the degrees to which it satisfies the interesting GRARs.

The non-adaptive, heuristic classification rule may, however, be a limitation of HyGRAR. We have addressed this by **proposing HyGRAR* as an improved version of HyGRAR** [69], in which the classification phase is automated. The improvement is achieved by replacing the non-adaptive classification rule with an adaptive one learned using MLPs. So, HyGRAR* involves MLPs not only to learn the gradual relations, but also to learn how to classify an instance based on interesting GRARs. In other words, MLPs make automatic correlations between interesting GRARs and classes, thus eliminating the need to predefine a classification rule. **HyGRAR*** has also been **proposed and evaluated as a solution for SDP**.

We have contributed to the applied research in SE by also approaching **Estimation of Software Coupling (ESC)**. A first original contribution in this regard was to propose **a new ML based conceptual software coupling measure, named ConC** [70]. ConC expresses the conceptual coupling between two software components as the similarity between the semantics of their source code. The semantic information is extracted from the source code and represented in a high-dimensional numeric space using Doc2Vec [56], an ANN based prediction model. The proposed conceptual coupling measure has been empirically evaluated in the context of **software restructuring** at package level.

Since conceptual coupling and structural coupling are partially complementary [9] and the potential of combining different coupling measures has been confirmed in the literature [48], we have integrated the proposed conceptual coupling measure in **a new ML based aggregated coupling measure** [32], which also considers the structural coupling. The aggregated coupling measure is named ACE and is defined as a linear combination of conceptual coupling and structural coupling. The conceptual coupling is still computed based on high-dimensional representations learned by Doc2Vec from the source code, while for expressing the structural dimension of coupling we have used an adaptation of a generic coupling measure from literature [102]. We have analyzed ACE in terms of its effectiveness for **change impact analysis**.

Most software defects are caused by violations of dependencies involved in software development processes. These dependencies are organizational or technical. The technical ones include software coupling [18]. Even if coupling has been considered in the SDP literature, only its structural dimension has been exploited extensively. The structural coupling metrics have been reported by numerous studies as effective for predicting software defects [92]. Even so, change coupling has been empirically confirmed to have superior impact on SDP compared to structural coupling [18]. Other dimensions of coupling, including the conceptual one, have been only very slightly investigated in relation to software defects [109].

In these conditions, we have proposed **a hybrid approach, named DePSem, for detecting software defects based on conceptual** (or semantic) **features learned from the source code** [73]. This approach opens our study on the interplay between conceptual coupling and software defects, while simultaneously following the two prevalent research directions on SDP: building improved classification techniques and designing relevant software features.

From a fundamental point of view, DePSem is a variant of HyGRAR*. It differs from HyGRAR* by the fact that the gradual relations used in the GRARs mining process are predefined and not learned using ANNs, but MLPs are still employed to learn how to classify an entity based on the interesting GRARs.

For predicting software defects, DePSem replaces the classic software metrics used by HyGRAR and HyGRAR* with semantic attributes extracted from the source code. The classic software metrics include structural coupling measures, while the semantic features help measuring conceptual coupling. For extracting the semantic features from source code we have used both Doc2Vec and Latent Semantic Indexing (LSI).

The DePSem's promising performance motivated us to make a step forward in capitalizing on conceptual coupling to predict software defects. Consequently, we have proposed **a conceptual coupling based metrics suite, named COMET, for SDP** [77]. As in the case of DePSem, the conceptual coupling is computed based on the source code representations provided by Doc2Vec and LSI. The 36 metrics composing the COMET suite are derived directly from conceptual coupling. Therefore, the impact of conceptual coupling on software defects is inferrable from how accurately software components measured using COMET are classified as defective or non-defective. To discern between the relevance of the COMET metrics and the power of the classification algorithm, the evaluation has been performed using standard ML algorithms. The same classification algorithms have been also applied on the widely-used metrics in the SDP literature in order to deduce the relative relevance of the COMET metrics suite for SDP. Therefore, with this last original contribution, we have also separately pursued the second SDP research direction, the one of designing new relevant software features.

To summarize, this thesis brings conceptual contributions to ML and contributes to the applied research in SE. The original conceptual contributions are: generalizing RARs mining towards GRARs mining [31], developing adaptive and dynamic GRARs unsupervised learning algorithms [71, 72] and developing new hybrid ML classifications models by combining GRARs mining with ANNs [75, 69, 73]. We have contributed to the applied research in SE through: applying new hybrid ML models as solutions for SDP [75, 69, 73], proposing new ML based approaches for measuring conceptual and aggregated software coupling [70, 32] and defining new conceptual coupling based metrics as support for SDP [77].

The aforementioned original contributions are presented in Chapters 2, 3, 4 and 5 of this thesis, that consists of five chapters and whose content is structured as follows.

The **first chapter** starts by presenting in its first section the background on the ML models involved in the research, including RARs mining and different ANN or ANN-based models. The second section of the first chapter, Section 1.2, defines, motivates and discusses in the context of the state of the art the two SE problems approached: SDP and ESC.

Chapter 2 presents our conceptual contributions to RARs mining. Section 2.1 motivates and defines the new concept of GRARs [31], introduces the GRANUM algorithm for mining interesting GRARs and presents an experimental evaluation of the GRARs mining approach. Section 2.2 presents the *adaptive* version of the GRANUM algorithm, named AGRARM, which adaptively uncovers the interesting GRARs within data sets whose attributes sets are incrementally extended [71]. Section 2.3 describes the *dynamic* version of the GRANUM mining algorithm, named DynGRAR. Section 2.4 concludes the second chapter and gives perspectives for future work.

Chapter 3 describes our original hybrid ML models, as solutions for predicting software defects based on classic software metrics. So, the chapter introduces both conceptual contributions resulted from hybridizing GRARs with ANNs and applied research contributions that benefit the SDP literature. Section 3.1 presents our first hybrid ML model, called HyGRAR, in the context of being proposed for predicting software defects [75]. Section 3.2 presents our proposal for an improved version of HyGRAR [69], in which an adaptive, MLP-based classification rule replaces the non-adaptive heuristic classification rule used in the initial version of the model. HyGRAR* is also proposed as a solution to SDP. The conclusions of the third chapter are drawn in its final section, which is Section 3.3. Directions for future research are pointed out in the same section.

Our new ML based approaches to Estimation of Software Coupling (ESC) are introduced in

Chapter 4. Our original approach of expressing the conceptual coupling between two software components using unsupervisedly learned high-dimensional representations of their source code [70] is proposed in Section 4.1. Section 4.2 introduces our proposal for an aggregated coupling measure that combines structural and conceptual coupling [32]. Section 4.3 summarizes the fourth chapter and indicates possible directions for future research in the area of ESC.

Chapter 5 contains our original contributions towards using coupling to assist SDP. Section 5.1 presents DePSem [73], an original approach to SDP in which a variation of our hybrid classification model combining GRARs with ANNs detects defective software entities based on conceptual features learned from the source code. Our proposal of a new conceptual coupling based metrics suite for supporting SDP is presented in Section 5.2. The ending section, Section 5.3, draws the conclusions of the last chapter and identifies directions for future investigations.

Finally, the last section of this thesis summarizes and brings together the main areas covered in the thesis. It also lists more general directions for future work.

Chapter 1

Background

The current chapter gives background on the Machine Learning (ML) models involved in the present thesis, as well as the statement, the motivation and the literature review for the two Software Engineering (SE) problems we have addressed: Software Defects Prediction (SDP) and Estimation of Software Coupling (ESC).

The first section introduces Relational Association Rules (RARs) mining, which we have extended to Gradual Relational Association Rules (GRARs) mining, and overviews the Artificial Neural Networks (ANNs) models we have used for defining new coupling measures, for building original hybrid models or for performing experimental analyzes.

The second section states, motivates and reviews SDP and ESC.

1.1 Related Machine Learning models

This section is concerned of the different ML models related to the research presented in this thesis. Its first subsection provides a brief overview of RARs mining, while the second one discusses several ANN or ANN based models.

1.1.1 Relational Association Rules

RARs mining [100] is an unsupervised learning technique that discovers frequent generic relations between the attributes values in a data set.

Let $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ be a set of instances, where each instance is characterized by a list of p attributes, $\mathcal{A} = (a_1, \ldots, a_p)$. Each attribute a_i takes values from a non-empty and non-fuzzy domain D_i, which also contains a *null* value. We denote by $\Phi(e_j, a_i)$ the value of attribute a_i for an instance e_j [66]. We denote by $\mathcal{R}$ the set of all the relations that can be defined between two domains D_i and D_j [100].

Definition 1. A Relational Association Rule (RAR) [100] is an expression $(a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}) \Rightarrow (a_{i_1}\ R_1\, a_{i_2}\ R_2\ a_{i_3} \ldots R_{\ell-1}\ a_{i_\ell})$, where $\{a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}\} \subseteq \mathcal{A} = \{a_1, \ldots, a_p\}$, $a_{i_j} \neq a_{i_k}$, $j, k = 1..\ell$, $j \neq k$ and $R_j \in \mathcal{R}$ is a relation over $D_{i_j} \times D_{i_{j+1}}$, D_{i_j} being the domain of the attribute a_{i_j}.

a) If $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ are non-missing in k instances from $\mathcal{E}$ then we call $s = \frac{k}{n}$ the *support* of the rule.

b) If we denote by $\mathcal{E}' \subseteq \mathcal{E}$ the set of instances where $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ are non-missing and the relations $\Phi(e_j, a_{i_1})R_1\Phi(e_j, a_{i_2})$, $\Phi(e_j, a_{i_2})R_2\Phi(e_j, a_{i_3})$, ..., $\Phi(e_j, a_{i_\ell-1})R_{\ell-1}\Phi(e_j, a_{i_\ell})$ hold for each instance e from $\mathcal{E}'$ then we call $c = \frac{|\mathcal{E}'|}{n}$ the *confidence* of the rule.

Definition 2. We call a RAR *interesting* if its support s and confidence are greater than or equal to given minimum thresholds, i.e $s \geq s_{min}$ and $c \geq c_{min}$.

The interesting RARs express frequent patterns in data thus being particular interest in data mining tasks. A complete and efficient algorithm, similar to Apriori [5], for mining all the interesting RARs, of any length, within a data set has been introduced in [100].

1.1.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) [112] are ML models that provide robust approaches for approximating discrete-valued, real-valued or vector-valued target functions [79]. They have been successfully applied in various domains ranging from bioarchaeology [76] to neurophysiology [78].

In the following, we present Multilayer Perceptron (MLP) and Radial Basis Functions Network (RBFN), both supervised ANNs, Self Organizing Map (SOM), an unsupervised ANN, and, ultimately, Doc2Vec, an unsupervised MLP based approach for representing documents as numeric vectors.

1.1.2.1 Multilayer Perceptron

The Multilayer Perceptron (MLP) is a supervised feedforward ANN. It has been inspired by the biological learning system, which is built of complex networks of interconnected neurons [79]. Accordingly, it consists of multiple fully interconnected layers of neurons, including one input layer, one output layer and one or more hidden layers. Except for the input neurons, each constitutive neuron, or computational unit, has attached an *activation function*. All the connections in the network are weighted.

The MLP is designed to be trained using an inductive learning algorithm called *backpropagation* [79]. Learning starts with a random initialization of the weights and consists in an effort to associate example inputs with their corresponding outputs. For this, an error is derived from the difference between the correct outputs and the network's outputs and is propagated backwards through the network. The error backpropagation is followed by the gradient computation [79], all the weights being updated accordingly, in order to reduce the error.

1.1.2.2 Radial Basis Functions Network

Another supervised neural network model is the Radial Basis Functions Network (RBFN). It has been introduced by Broomhead and Lowe [14].

The motivation of the RBFN model lies in the locally tuned response of biologic neurons, while its theoretical foundation is given by the interpolation of multivariate functions [98]. Accordingly, the output of a RBFN is obtained as a linear combination of the hidden units activations [79], while the outputs of the hidden units are generated by Radial Basis Functions (RBFs).

There are many training algorithms for RBFNs. Schwenker et al. [98] have classified these algorithms into one-, two- and three-phase learning schemes. The *two-phase RBFN learning scheme* is the most commonly applied. It involves training the hidden layer and the output weights separately. The centers are usually unsupervisedly learned using clustering algorithms, while the weights are supervisedly learned, through pseudo-inverse solution or gradient descent.

1.1.2.3 Self-Organizing Maps

A Self Organizing Map (SOM) is a type of unsupervised ANN. It has been introduced by Kohonen [53]. The model is biologically motivated by the function of the visual cortex [95].

SOM provides a low-dimensional representation of data, called a (Kohonen) *map* [103], that preserves the topological ordering of the instances. So, a trained SOM also indicates clusters of data instances [55]. Consequently, SOM merges the goals of data projection and clustering [49].

From the architectural point of view, a SOM consists, unlike MLPs and RBFNs, of just two fully connected layers: the input layer and the output layer that models the actual map and represents the internal state of the model.

From the training perspective, a SOM is trained unsupervisedly, through competitive learning, as opposed to the error-correction learning specific to supervised ANN models.

1.1.2.4 Doc2Vec

Paragraph Vector, or Doc2Vec, is a MLP based model proposed by Le and Mikolov [56]. It allows expressing variable-length textual information as a fixed-length dense numeric vector, called *paragraph vector*, thus being an alternative to common models such as bag-of-words and bag-of-n-grams.

A first advantage of Doc2Vec over the traditional models is that it considers the semantic distance between words [56]. An additional advantage over bag-of-words is that it also takes into consideration the words order, at least in a small context.

1.2 Approached Software Engineering problems

This section presents the main Software Engineering (SE) problems addressed in the current thesis, namely predicting software defects and estimating software coupling. In the following, we give the statement, motivation and the literature review for each of them.

1.2.1 Software Defects Prediction

1.2.1.1 Problem statement and relevance

Software defects are logic or implementation errors that cause the system to operate in unintended ways or to produce incorrect results. SDP consists in identifying the software components that contain defects.

SDP has a broad applicability. It helps managers to measure how software projects evolve [22] and supports process management, by assessing the software product's quality and the process' performance [22]. It also reduces the costs of the processes that aim at ensuring the software's quality [45], for instance by allowing to focus on the components identified as defective [46].

Despite its importance and extensive applicability, the detection of defective software modules is a complex task. One of the main challenges is performing cross-project SDP [87]. Another one is learning from highly imbalanced SDP data sets.

1.2.1.2 Literature review

The prediction of defects in software systems is a highly active research area. For instance, Hall et al. have considered, in a systematic review of SDP [43], 208 studies on defect prediction, all published between 2000 and 2010.

The research efforts in the field of SDP take one of the following two directions: proposing new high-performance classifiers [3, 64, 16, 67, 87] or designing new relevant features on the basis of which to distinguish between defective and non-defective software modules [7, 51, 52].

Along the first direction, the vast majority of existing studies [80, 81, 111, 113] have considered, as experimental data, some of the SDP data sets available in Promise Software Engineering

Repository [97]. Accordingly, many of the existing SDP approaches are based on the Promise software metrics [41], which include, among others, structural coupling measures. Logic (or evolutionary) coupling has also been used, even if a very limited number of studies [7, 51, 52], for predicting software defects.

1.2.2 Software Coupling Estimation

1.2.2.1 Problem statement and relevance

Coupling is a fundamental property of software systems, being strongly connected with the quality of software design and having high impact on program understanding. Coupling expresses the degree of dependence between software components, including the hidden dependencies.

Coupling metrics have proven to be useful in different software engineering activities, such as: predicting defect proneness [52], predicting maintainability in service-oriented designs [89], crosscutting concerns identification [68, 23], software restructuring [30], change impact analysis [91, 12] and regression testing [110].

1.2.2.2 Literature review

The concept of software *coupling* has been introduced in the literature by Stevens et al. [105], in the context of structured design. Subsequently, coupling has been adapted for object-oriented (OO) programming. For instance, one well-known and often used set of OO metrics, including structural coupling, is the Chidamber and Kemerer (CK) [21] metrics set. Another one is composed of the so-called MOOD metrics [36].

Regarding the types of coupling metrics, the prevalent coupling metrics in the SE literature are *structural* [6, 83]. They measure the structural connections between software components. A framework that provide a unified view through the existing structural coupling measures has been proposed by Briand et al. [13].

The structural coupling measures are complemented by *conceptual* coupling measures, which express the degrees of semantic correlation between software entities [91].

Other coupling types defined in the literature are *dynamic* coupling [8, 61], which takes into consideration connections that appear during the execution of the program, and *logic* coupling (also called evolutionary coupling or change coupling) [7, 9, 52], which considers entities that are changed together.

Chapter 2

Contributions to Relational Association Rules Mining

This chapter presents our conceptual contributions to Relational Association Rules (RARs) mining. These contributions include extending the conventional RARs mining towards Gradual Relational Association Rules (GRARs) mining [31], as well as developing *adaptive* [71] and *dynamic* [72] versions of the GRARs mining algorithm.

The first section, Section 2.1, motivates and defines GRARs [31], introduces the Gradual Relational Association Rules Miner (GRANUM) algorithm for mining interesting GRARs and presents the experimental evaluation of the GRARs mining approach. GRARs mining is evaluated against non-gradual RARs mining in terms of expressiveness, ability to capture semantically relevant rules and robustness to noise.

Section 2.2 introduces and evaluates the *adaptive* version of the GRANUM mining algorithm, named Adaptive Gradual Relational Association Rules Miner (AGRARM) [71]. AGRARM efficiently uncovers the interesting GRARs within dynamic data whose attributes sets are incrementally extended. The approach is evaluated in multiple scenarios of extending the attributes set. Since the intention behind the proposal was to have a more efficient alternative to resume GRANUM in such scenarios, the AGRARM's running time is compared to the one of GRANUM run from scratch on the extended data.

Section 2.3 presents and evaluates the *dynamic* version of the GRANUM mining algorithm, named Dynamic Gradual Relational Association Rules Miner (DynGRAR) [72]. DynGRAR extends AGRARM in order to dynamically uncover the interesting GRARs within data sets that are extended not only with new attributes, but also with new instances. A comparative evaluation of DynGRAR against GRANUM, consisting of multiple experiments performed in various dynamic mining settings and using different data sets from the Software Defects Prediction (SDP) literature, is conducted. The presentation of the experimental results is followed by a comparison to related work.

Section 2.4 concludes the chapter and provides directions for future work.

The approaches presented in the current chapter are original research works published in [31], [71] and [72].

2.1 Gradual Relational Association Rules Mining

The current section presents Gradual Relational Association Rules (GRARs) mining, which we have introduced in [31] as an extension of the non-gradual RARs mining. The aim of our proposal was to create a more expressive and stable RARs mining approach. By using gradual relations, specifically fuzzy [54] relations, instead of boolean, non-gradual ones, GRARs become

aware of the degrees to which the relations are satisfied, while inheriting the noise-tolerance specific to the fuzzy approaches.

2.1.1 Motivation

An inconvenience of using the crisp, non-gradual RARs in some data mining tasks is that the set of interesting rules discovered is sensitive to noisy data [24].This is because the crisp approach fails to overlook the noise. But noisy data is common in real-word problems [62]. Many studies have suggested that fuzzy [85] concepts are particularly convenient to model noisy data [38]. Consequently, by extending RAR through replacing the boolean relations with fuzzy relations, we aim at obtaining a more stable approach for mining noisy or imprecise (approximated) data [38].

Furthermore, there are situations in which it is relevant to consider the degree to which a relation between two attributes values is satisfied. But the non-gradual approach is unaware of the degree to which a relation is violated. Gradual relations [47] take advantage of a membership function which expresses the degree to which a relation is verified. Accordingly, by considering fuzzy relations in the mining process, we target increasing the expressiveness.

Concluding, the motivation for introducing Gradual Relational Association Rules is two-fold: they are expected to be both more noise-tolerant and more expressive.

2.1.2 Theoretical model

Let $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ be a set of instances, where each instance is characterized by a list of p attributes $\mathcal{A} = (a_1, \ldots, a_p)$. Each attribute a_i takes values from a non-empty and non-fuzzy domain D_i, which also contains a *null* (empty) value. We denote by $\Phi(e_j, a_i)$ the value of attribute a_i in the instance e_j.

Definition 3. A *fuzzy binary relation* [11] $\mathcal{G}$ between two non-fuzzy domains D_i and D_j is defined as follows: $\mathcal{G} = \{< (x, y), \mu_{\mathcal{G}}(x, y) >: x \in D_i, y \in D_j\}$, where $\mu_{\mathcal{G}} : D_i \times D_j \to [0, 1]$ is a membership function which associates to each pair $(x, y), x \in D_i, y \in D_j$ the *membership degree* $\mu_{\mathcal{G}}(x, y)$ measuring the degree to which the relation $\mathcal{G}$ is satisfied.

We use the notation $\mathcal{F}$ for the set of the binary relations considered in the GRARs mining process.

Definition 4. A Gradual Relational Association Rule (GRAR), $\mathcal{G}Rule$, is a sequence $(a_{i_1}\ \mathcal{G}_1\ a_{i_2}\ \mathcal{G}_2\ a_{i_3} \ldots \mathcal{G}_{\ell-1}\ a_{i_\ell})$, where $\{a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}\} \subseteq \mathcal{A} = \{a_1, \ldots, a_p\}$, $a_{i_j} \neq a_{i_k}$, $j, k = 1..\ell$, $j \neq k$ and $\mathcal{G}_j \in \mathcal{F}$ is a binary fuzzy relation over $D_{i_j} \times D_{i_{j+1}}$, D_{i_j} being the domain of the attribute a_{i_j}.

The *membership* of a GRAR called $\mathcal{G}Rule$ *for an instance* $e \in \mathcal{E}$ is defined using the *min* t-norm function [39] as $\mu_{\mathcal{G}Rule}(e) = \min\{\mu_{\mathcal{G}_j}(\Phi(e, a_{i_j}), \Phi(e, a_{i_{j+1}})), j = 1, 2, \ldots, \ell-1\}$ and expresses the magnitude to which the rule is satisfied.

a) If $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ are non-missing in k instances from $\mathcal{E}$ then we call $s = \frac{k}{n}$ the *support* of the GRAR $\mathcal{G}Rule$.

b) If $\mathcal{E}' \subseteq \mathcal{E}$ is the set of all instances in $\mathcal{E}$ for which $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ are non-missing and $\mu_{\mathcal{G}Rule}(e) > 0$ for each instance e from $\mathcal{E}'$, then we call $c = \frac{|\mathcal{E}'|}{n}$ the *confidence* of the rule.

c) Using the notation from b), we call $m = \frac{\sum_{e \in E'} \mu_{\mathcal{G}Rule}(e)}{n}$ the rule's *membership at the level of the data set* $\mathcal{E}$.

The *interestingness of a GRAR* is defined as follows.

Definition 5. We call a gradual relational association rule *interesting* if its support s and confidence c are greater than or equal to given minimum thresholds, i.e $s \geq s_{min}$ and $c \geq c_{min}$.

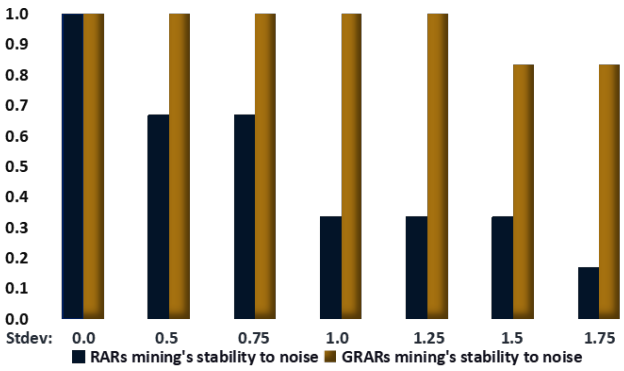


Figure 2.1: Stability to noise: RARs vs GRARs

2.1.3 The Gradual Relational Association Rules Mining algorithm

In order to discover all the interesting GRARs within a data set, we have adapted the non-gradual RARs mining algorithm introduced in [100], thus obtaining the Gradual Relational Association Rules Miner (GRANUM) algorithm. Using a length-level, iterative process, GRANUM learns all the interesting GRARs, of any length, that occur in data.

GRANUM leads to a significant pruning of the exponential search space given by all possible GRARs. It guides the search to those regions where interesting rules may be found, while pruning the other regions.

It is provable that the GRANUM algorithm is complete and correct.

2.1.4 Experiments and results

We have experimentally evaluated the GRARs mining approach, emphasizing its advantages over the non-gradual approach.

In a first case study, we have shown that the gradual approach is able to capture rules which are inaccessible to the non-gradual one either because of small approximation errors or because they lack the expressive power of the membership which captures the magnitude of rules, i.e. how strong or well satisfied they are.

A second case study have emphasized that GRARs mining is more robust to noise than the non-gradual approach. To test the stability of the gradual and non-gradual mining approaches, we incrementally introduced Gaussian noise of mean 0 and increasing standard deviation in data. Figure 2.1 shows that, as expected, the stability to noise decreases, for both gradual and non-gradual approaches, as the standard deviation of noise increases, but we note that the set of (interesting) non-gradual RARs, in dark blue, is much more affected by noise.

Based on the results of the experiments performed, we concluded that GRARs are more comprehensive, more expressive and less sensitive to noise than the non-gradual RARs.

2.1.5 Comparison to related work

The concept of GRAR is a novel one since there are no other approaches using fuzzy [50] relations in Association Rules (ARs) mining.

2.2 AGRARM: An Adaptive Gradual Relational Association Rules Mining approach

In the current section, we present the method called Adaptive Gradual Relational Association Rules Miner (AGRARM), which we have introduced in [71]. AGRARM discovers all interesting GRARs in a dynamic data set whose attributes set is extended with one or more new attributes, through adapting the set of interesting rules mined before extension, so as to preserve the completeness.

2.2.1 Motivation

The GRANUM algorithm briefly described in Section 2.1.3 discovers all the interesting GRARs within a known set of objects that are measured against a known set of features. But there are situations when the data is horizontally dynamic, in the sense that the attributes set characterizing its objects evolves. Clearly, for obtaining, in such a setting, the interesting GRARs, the mining algorithm can be re-applied every time the feature set changes. But this could be inefficient and unworthy, especially if the attribute set is only very slightly expanded. Consequently, we were motivated to introduce an alternative to resuming the GRANUM mining algorithm when the data set is enlarged with a number of new attributes. We proposed, therefore, AGRARM, which is an algorithm that adapts the set of all interesting GRARs mined before extension so as to obtain all interesting GRARs that characterize the extended data.

2.2.2 Methodology

AGRARM is a complete algorithm that, starting from set of interesting rules mined before extension and considering the newly added features, adapts the rule set so as to obtain all interesting GRARs characterizing the extended data.

So, AGRARM starts by performing an initial pass over the extended data to identify new interesting binary rules. These are rules contain at least one newly added attribute. In the subsequent iterations, they may be used for generating new interesting GRARs of increasing length.

2.2.3 Experiments and results

We have evaluated the AGRARM against GRANUM run from scratch on the extended data, in terms of time efficiency. We have considered three different SDPs data sets (Tomcat, Ar and JM1), various possibilities of extending the data with new attributes and multiple values for the minimum support, confidence and membership thresholds.

The experimental evaluation results confirmed that AGRARM provided the interesting GRARs within the enlarged data significatly more rapidly than resuming the mining algorithm GRANUM.

2.2.4 Comparison to related work

AGRARM, the adaptive mining approach introduced in Section 2.2.2 is new in the Association Rules Mining (ARM) literature. The existing approaches [82, 35, 84, 19, 114, 60] consider non-relational ARs and their adaptability refers to other aspects, except for Adaptive Relational Association Rules Miner (ARARM) [26], which handles non-gradual RARs.

2.3 DynGRAR: A Dynamic Gradual Relational Association Rules Mining approach

We briefly present, in the current section, the Dynamic Gradual Relational Association Rules Miner (DynGRAR) algorithm we have introduced in [72]. DynGRAR extends AGRARM and uncovers all interesting GRARs in dynamic data sets which are incrementally extended with both new data instances and new data attributes.

2.3.1 Motivation

In Section 2.2 we presented AGRARM, which efficiently discovers all interesting GRARs in a data set whose attributes set is extended with one or more new attributes. However, the dynamics of the data set to be mined is not exclusively determined by adding new attributes to it. The data set can also be extended with new data instances.

Accordingly, we are introducing a new approach, DynGRAR for uncovering the interesting GRARs in such dynamic data sets.

SDP is a potential application domain for dynamic GRARs mining, due to the intrinsic dynamic nature of the software development process.

2.3.2 Methodology

DynGRAR is divided in two successive phases. The first phase consists of filtering the set of interesting rules mined before extension, to keep only the rules which are interesting in the extended data set, too. The second phase consists in extending the subset resulted from the first phase with new interesting GRARs. The new interesting GRARs are either GRARs containing only attributes from the initial set of attributes, but which were not interesting within the initial data and only become interesting on the extended data set due to the newly added instances or interesting GRARs containing at least one newly added attribute.

DynGRAR maintains the completeness of the GRARs generation procedure, but is also expected to be more efficient in terms of the required mining time.

2.3.3 Experiments and results

We evaluated DynGRAR against GRANUM run from scratch on the extended data, in terms of time efficiency, by considering different SDP data sets, various possibilities of extending the data with new attributes and/or instances and multiple values for the minimum support, confidence and membership thresholds. Figure 3.2 summarizes the reduction of the mining time obtained when replacing GRANUM run from scratch with DynGRAR on five different case studies. The mining time has been reduced, in average, with 43% and at most with 93,8%.

2.3.4 Comparison to related work

The DynGRAR algorithm is new in the data mining literature. The problem of dynamically mining the interesting GRARs in a data set which is extended with both new attributes and new instances has not been approached in the literature, so far. There are recent approaches handling incremental and adaptive Association Rules Mining (ARM) [82, 35, 84, 114, 19, 60, 99, 115], but they consider classical ARs and not gradual relational ones.

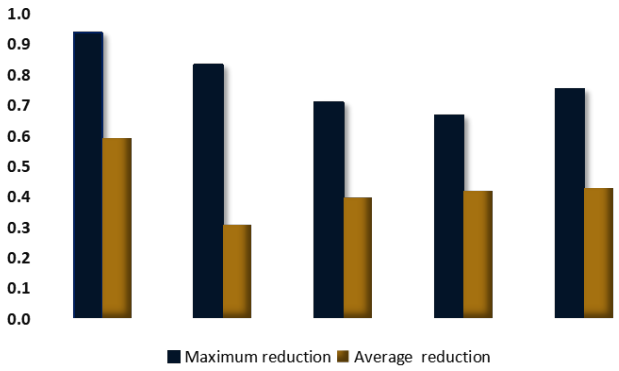


Figure 2.2: Reduction in mining time when using DynGRARs instead of GRANUM

2.4 Conclusions and further work

This section summarized our conceptual contributions to RARs mining, namely GRARs mining, which is our proposal for extending RARs mining by enhancing it with increased expressiveness and more noise stability, and two GRARs mining algorithms, AGRARM and DynGRAR, that dynamically uncover the interesting GRARs within data sets which evolve in time by being enlarged with new attributes and/or instances.

Multiple experiments have been performed to comparatively evaluate the performance of the two adaptive algorithms against applying GRANUM from scratch on the extended data. The obtained results highlight that both AGRARM and DynGRAR significantly reduce the time required to uncover the interesting GRARs in dynamic data sets.

A direction for further extending the GRARs mining approach is to include unary gradual relations, in addition to the binary ones. The generalization can go even further, by also adding ternary gradual relations, quaternary gradual relations and so on.

Future work will also focus on developing a concurrent version of the DynGRAR algorithm in order to improve its time efficiency.

Another possible area of future research would be to use DynGRAR for incremental SDP.

Chapter 3

Novel hybrid approaches to Software Defect Prediction

The current chapter simultaneously presents our fundamental research contribution to Machine Learning (ML), obtained through hybridizing Gradual Relational Association Rules (GRARs) mining and Artificial Neural Networks (ANNs) in new classification models, and our applied research contributions to Software Defects Prediction (SDP), produced by applying the developed hybrid classification models for predicting software defects, based on conventional software metrics.

Section 3.1 introduces our first hybrid ML model, named HyGRAR [75], in the context of being proposed for predicting software defects. The section also summarizes an extensive experimental evaluation of the model, conducted on 10 publicly available SDP data sets.

Section 3.2 introduces HyGRAR* [69], which is an improved version of HyGRAR, that is also described and evaluated as a SDP model. Both HyGRAR and HyGRAR* combine GRARs mining with ANNs, but HyGRAR uses ANNs only before the mining step, to learn gradual relations, while HyGRAR* also utilizes them to automatize the classification step.

The chapter's conclusions drawn in Section 3.3, along with directions for future research.

The approaches presented in the current chapter are original research works published in [75] and [69].

3.1 HyGRAR: A hybrid software defects predictor based on software metrics

This section introduces HyGRAR, the initial version of the hybrid classification model combining GRARs mining with ANNs, together with its experimental evaluation that has been conducted using 10 publicly available SDP data sets.

For background on ANNs and GRARs, we refer the reader to Section 1.1.2 and 2.1.

3.1.1 Motivation

The statement and the motivation for the SDP problem are given in Section 1.2.1. In proposing HyGRAR as an approach to SDP, we have intuitively assumed that relations between relevant software metrics values may be significant for indicating vulnerability to defects. Furthermore, we have considered that learning such relations is preferable to pre-defining them because of their adaptability and expressive power.

3.1.2 Methodology

The main idea of the HyGRAR classifier is as follows.

Firstly, it employs ANNs to learn from the experience of previous projects or previous versions of the current project, gradual relations between pairs of software metrics that entail vulnerability to defects.

Secondly, Gradual Relational Association Rules Miner (GRANUM) mining algorithm is fed with these relations and separately uncovers the interesting GRARs that hold for the defective and non-defective subsets of the available training data corresponding to the current project (or version).

Finally, in the classification phase, based on the interesting discriminative rules and using a predefined strategy, a software entity is classified as defective or non-defective.

Accordingly, The following steps are performed in order to determine whether or not a software entity is defective.

1. Data pre-processing, which involves feature selection and data balancing.
2. Building the HyGRAR classifier through two distinct training phases:

   Phase 1. Learning gradual relation between any two software metrics with respect to the vulnerability to defects.

   Phase 2. Mining the interesting GRARs characterizing each of the two classes (defective and non-defective).
3. Classification (or testing).

3.1.3 Experiments and results

3.1.3.1 Data sets

Ten publicly available data sets have been used in the experimental evaluation of HyGRAR. We have first considered five data sets (Ar1, Ar3, Ar4, Ar5 and Ar6 [97]) corresponding to the Ar embedded software implemented in C thus evaluating HyGRAR in an inter-version setting. We have also evaluated HyGRAR in an inter-project setting, on three OO software systems: JEdit, Ant and Tomcat. For JEdit, we have considered three different versions.

We selected a relatively small number of the available software metrics, on the criterion of their correlation with the class label.

3.1.3.2 Evaluation

The evaluation has been performed using the leave-one-out cross-validation method and we computed multiple performance indicators (including Accuracy (Acc), Precision (Prec), Specificity (Spec), Sensitivity (Sens)), but for comparisons we used mostly Area Under the Receiver Operating Characteristics (ROC) Curve (AUC), since it is widely considered the best measure for comparing defect predictors [37].

3.1.3.3 Results

Considering 17 other approaches reporting results on the Ar data sets and presented in [1, 10, 88, 17, 108, 3, 113, 81, 65], Figure 3.1 depicts HyGRAR's AUC versus the average AUC on related work, for which standard errors bars are shown. HyGRAR's AUC is higher that the average AUC on the related work for all the five versions. It outperforms all the other approaches for three versions and lies in top 2 for all the five versions.

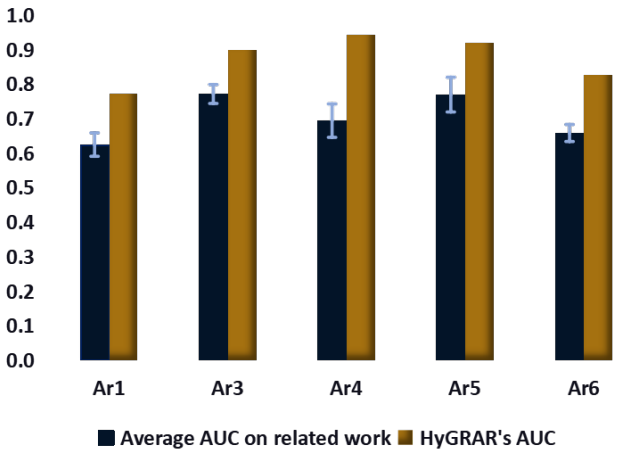


Figure 3.1: The relative performance of HyGRAR on AR data sets

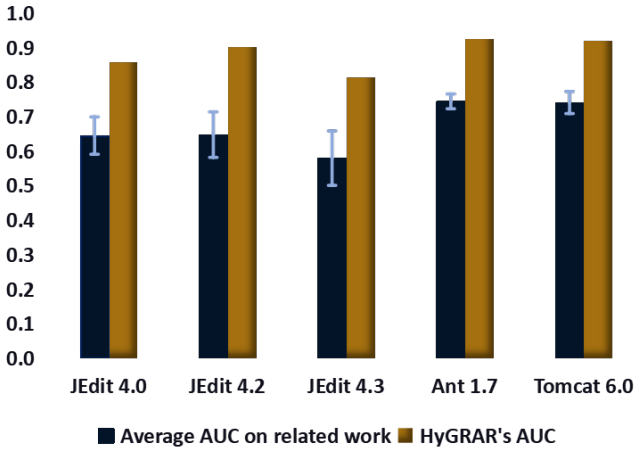


Figure 3.2: The relative performance of HyGRAR on the data sets corresponding to OOP software projects

Considering 15 other approaches for comparison [67, 64, 16, 80, 87], HyGRAR obtained an AUC that was higher not only than the average AUC on the related works reporting results on the same data sets, but also than the maximum AUC values.

The results show that among the 10 case studies, HyGRAR is the best classifier of eight data sets and it is one of the top two classifiers for the other two data sets.

3.2 HyGRAR*: An improved hybrid software defects predictor based on software metrics

In the current section, we present the enhanced version of HyGRAR, named HyGRAR*.

The improvement is achieved by replacing the non-adaptive, heuristic classification sub-algorithm in the initial version of the model with an adaptive one based on Multilayer Perceptrons (MLPs). Evaluation experiments performed on two open-source NASA data sets indicate that HyGRAR* classifier outperforms both HyGRAR, the initial version of our hybrid model, and the other related approaches evaluated on the same two data sets.

3.2.1 Motivation

A limitation of the HyGRAR approach is that the classification methodology is not adaptive.

For alleviating the previously mentioned limitation, we propose to improve the classification step of HyGRAR through autonomously learning the classification methodology.

3.2.2 Methodology

The data pre-processing step and the two training phases used for building the HyGRAR classifier remain unchanged in the case of HyGRAR*, but the heuristic classification rule used in the testing phase is replaced with an adaptive one learned by MLPs.

3.2.3 Experiments and results

3.2.3.1 Data sets

We experimented on the PC NASA software defect data sets [97] corresponding to a flight software for earth orbiting satellite by performing two case studies. The two case studies correspond to evaluations of the proposed model on two data sets: PC3 and PC4, respectively. These two data sets are the most difficult from the data sets which correspond to the first four versions of PC software, according [27].

3.2.3.2 Evaluation

For evaluating the performance of HyGRAR*, we used the same methodology as in the case of HyGRAR (see Section 3.1.2).

3.2.3.3 Results

On the two experimental case studies, HyGRAR* outperformed all the other 9 related approaches we found in the literature [63, 90, 27], as well as the initial version of our hybrid model.

The following two figures compare HyGRAR* with related work, in terms of AUC, but also accuracy since the majority of the related studies report only the accuracy.

3.3 Conclusions and future work

In this chapter we have proposed a hybrid supervised learning model called HyGRAR as an approach to SDP. HyGRAR is a combination of GRARs mining and ANNs. The proposed model is novel from the perspectives of both Search Based Software Engineering (SBSE) and ML. The experimental results show that HyGRAR performs better than similar previously proposed methods, in most of the cases. We have also proposed an improved version of HyGRAR, named HyGRAR*, that replaces the non-adaptive classification rule in its initial version with an adaptive one, based on MLPs, thus further increasing its performance.

The excellent performance of HyGRAR and HyGRAR* are most likely due to their complexity and adaptability. On the down side, the complexity affects, however, the readability of the classification rules. Another indirect limitation of the model may be the inadequate relevance of the software metrics used for prediction. Consequently, a direction for future work would be to address this limitation by automatically extracting relevant software characteristics from software artifacts. As an additional direction for further work, we plan to further optimize the HyGRAR* classifier, for example through weighting the predictions of the individual MLPs with their performances on validation data sets.

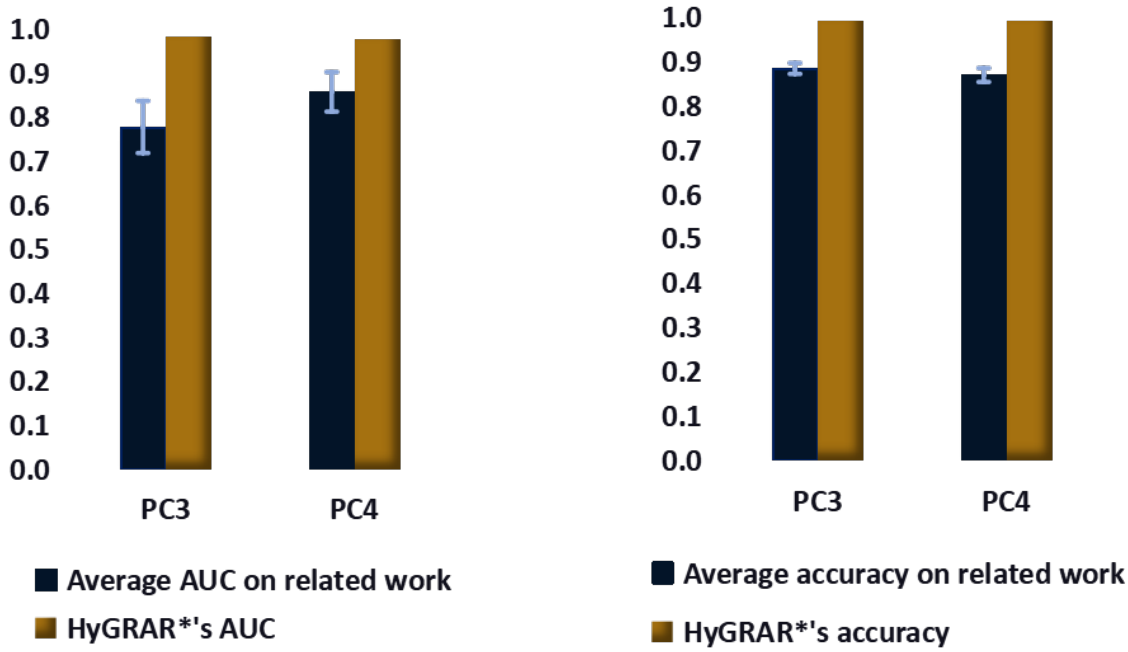


Figure 3.3: HyGRAR*'s relative classification performance

Chapter 4

New Machine Learning based approaches to Software Coupling Estimation

The current chapter presents our original Machine Learning (ML) based contributions to Estimation of Software Coupling (ESC).

These contributions are introducing a new conceptual coupling measure [70] and including it in a new aggregated coupling measure [32].

Section 4.1 presents our original approach of defining a new conceptual coupling, named ConC, based on unsupervisedly learned high-dimensional representations of the source code [70]. ConC is experimentally compared with existing coupling measures, including from the perspective of software restructuring at package level.

Section 4.2 introduces our proposal for an aggregated coupling measure, named ACE, that combines conceptual and structural coupling [32]. The section also details the experimental comparative evaluation of ACE, which has been performed with the aims of showing that ACE differs from existing coupling measures and it is more effective for change impact analysis.

Section 4.3 summarizes the chapter and indicates possible directions for future work.

The approaches presented in the current chapter are original research works published in [70] and [32].

4.1 ConC: An Artificial Neural Networks based Conceptual Coupling measure

In this section we present our original approach of expressing the conceptual coupling between two software entities using unsupervisedly learned high-dimensional representations of the source code [70]. These representation are learned using the Doc2Vec Artificial Neural Network (ANN) based model described in Section 1.1.2.4.

The intent of the experimental evaluation presented in this section is to empirically show that the proposed conceptual coupling measure expresses other aspects of coupling than structural coupling measures and is more effective in the context of software restructuring at package level.

4.1.1 Motivation

The motivation for approaching ESC is given in Subsection 1.2.2. In the following, we motivate our particular approach to it.

The conceptual coupling [91] measures the strength of the conceptual (or semantic) similarity between two software components and, according to Bavota et al. [9], is the closest to the developer's perception of coupling.

Since elements of the source code, such as identifiers and comments usually contain semantic information [91], we proposed a new conceptual coupling measure, that is learned starting from the source code.

4.1.2 The proposed conceptual coupling measure (ConC)

To extract the semantics from the source code implementing a given software entity e, the following steps are performed.

First, the source code is converted into a text corpus containing the elements of the implementation code, including comments, identifiers, etc. Then, this corpus is converted into a fixed-length feature vector of numerical values $ConV(e) = (e^1, e^2, \ldots, e^m)$. The numerical vectors associated to the software entities are unsupervisedly learned using Doc2Vec.

Finally,the conceptual coupling of two software components is computed as as the similarity between their corresponding vectors.

Definition 6. The *conceptual vector* associated to a software entity e is defined as its representation in the Doc2Vec space, $ConV(e) = (e^1, e^2, \ldots, e^m)$.

Definition 7. The *conceptual coupling* between two software entities e_1 and e_2, denoted by $ConcC(e_1, e_2)$, is defined as the *similarity* between the conceptual vectors $({e_1}^1, {e_1}^2, \ldots, {e_1}^m)$ and $({e_2}^1, {e_2}^2, \ldots, {e_2}^m)$ associated to e_1 and e_2.

In defining the ConC measure, the similarity between the conceptual vectors may be defined as the euclidean similarity expressed in Formula (4.1) or as in Formula (4.2) using the cosine similarity.

$$ConcC(e_1, e_2) = \frac{1}{1 + \sqrt{\sum_{i=1}^{m} (e_1^i - e_2^i)^2}} \tag{4.1}$$

$$CC(e_1, e_2) = \frac{\left| \sum_{i=1}^{m} ({e_1}^i \cdot {e_2}^i) \right|}{\sqrt{\sum_{i=1}^{m} ({e_1}^i \cdot {e_1}^i)} \cdot \sqrt{\sum_{i=1}^{m} ({e_2}^i \cdot {e_2}^i)}} \tag{4.2}$$

In the experimental part of this subsection, we used the euclidean distance for expressing the dissimilarity between the conceptual vectors.

4.1.3 Experiments and results

As an experimental case study, we considered Apache Commons DBUtils, version 1.3.

The experiments have been conducted in two directions.

First, we have emphasized, using Principal Component Analysis (PCA) [2], that ConC expresses new aspects of coupling, which are not detected by the structural coupling measurements (Coupling Between Objects (CBO) [21], Message Passing Coupling (MPC) [59], Data Abstraction Coupling (DAC) [59], Information Flow Based Coupling (ICH) [57]).

Second, we have comparatively tested ConC and structural coupling measures in terms of their relevance to software restructuring at package level. For performing the comparison, we used Self Organizing Maps (SOMs) which learned mappings highlighting that considering the conceptual vectors leads to a better structure than using structural coupling.

4.1.4 Comparison of ConC with existing conceptual coupling measures

The ConC conceptual coupling measure defined in this section differentiates itself from already existing conceptual coupling measures. Instead of using Latent Semantic Indexing (LSI) [42] for representing the text corpus, we have used Doc2Vec. Doc2Vec is a prediction-based model, while LSI is a count-based model. Moreover, Doc2Vec is known in the literature to better capture the text semantics than count-based information retrieval methods [58].

4.2 ACE: An Aggregated Coupling measure

In the current section, we introduce a new aggregated coupling measure, named ACE.

The major goal of the work presented in this section is to emphasize the relevance of aggregating structural and conceptual coupling into ACE, as a proof of concept, by answering the following research questions:

RQ1 Is ACE able to express new aspects of coupling, which are not detected by the structural and conceptual coupling measures existing in the literature?

RQ2 To what extent is the ACE measure correlated with structural and conceptual coupling measures existing in the literature?

RQ3 Would ACE be effective for change impact analysis and how does it compare with similar measures already proposed in the literature for this activity?

4.2.1 Motivation

The motivation for ESC is given in Section 1.2.2. Hence, in the current section we motivate combining structural and conceptual coupling measures into a new aggregated measure.

First, there are software engineering activities (e.g. software restructuring) for which the structural coupling metrics alone may be insufficient.

Second, Bavota et al. [9] have experimentally concluded that structural and conceptual coupling are partially complementary.

Third, the combination of different software metrics is encouraged in the literature [44, 48].

4.2.2 The proposed aggregated coupling measure (ACE)

The structural coupling component of ACE is an adaptation of a generic coupling measure from the literature [102]. It is computed based on the relevant properties shared by the software components.

Definition 8. The **structural coupling** between any two software entities e_1 and e_2 from a software system S, denoted by $SC(e_1, e_2)$ is expressed as

$$SC(e_1, e_2) = \frac{|rp(e_1) \cap rp(e_2)|}{|rp(e_1) \cup rp(e_2)|} \tag{4.3}$$

where $rp(e)$ is the set of the *relevant properties* of an entity $e \in S$.

- If e is a method, then $rp(e)$ contains: the method itself, the class which defines the method, all the attributes which are accessed by the method, all the other methods used by e and all the other methods which overwrite e.

- If e is an application class, then $rp(e)$ contains: the class itself, all the attributes and the methods that are defined in e, all the interfaces which are implemented by e and all the application classes which are extended by class e.

In defining the aggregated coupling measure ACE, the structural coupling measure previously defined is complemented by the conceptual coupling measure ConC introduced in Subsection 4.1.2. As component of ACE, ConC is defined using the cosine similarity, in contrast to the definition considered in the experimental evaluation performed in Subsection 4.1.3. For this reason, in the current section we use a customized notation, namely CC, and give in the following the local definition.

Definition 9. The **conceptual coupling** between two software entities e_1 and e_2, denoted by $CC(e_1, e_2)$, is defined as the absolute value of the cosine between the vectors $({e_1}^1, {e_1}^2, \ldots, {e_1}^m)$ and $({e_2}^1, {e_2}^2, \ldots, {e_2}^m)$ corresponding to their Doc2Vec representation.

$$CC(e_1, e_2) = \frac{\left|\sum_{i=1}^{m} ({e_1}^i \cdot {e_2}^i)\right|}{\sqrt{\sum_{i=1}^{m} ({e_1}^i \cdot {e_1}^i)} \cdot \sqrt{\sum_{i=1}^{m} ({e_2}^i \cdot {e_2}^i)}} \tag{4.4}$$

Considering the structural and conceptual coupling measures introduced in Formulae (4.3) and (4.4), respectively, we define below the aggregated coupling measure between two software entities as a linear combination of their structural (SC) and conceptual (CC) similarity.

Definition 10. The **aggregated coupling** between two software entities e_1 and e_2, denoted by $AC(e_1, e_2)$, is defined as a linear combination between their *structural* coupling $SC(e_1, e_2)$ and *conceptual* coupling $CC(e_1, e_2)$.

$$AC(e_1, e_2) = w \cdot SC(e_1, e_2) + (1 - w) \cdot CC(e_1, e_2) \tag{4.5}$$

where w is a sub-unit weight expressing the importance of the *structural* relations in the overall coupling measure.

Definition 11. The **aggregated coupling of a software entity** e within a software system $\mathcal{S}$, denoted by $ACE(e)$, is defined as

$$ACE(e) = \frac{\sum_{\substack{e' \in \mathcal{S} \\ e \neq e'}} AC(e, e')}{p} \tag{4.6}$$

where $p = |\{e' | e' \in \mathcal{S}, e' \neq e\}|$ is the number of the software entities from $\mathcal{S}$ that are different from e.

4.2.3 Comparison between ACE and other coupling measurements

To answer RQ1 we applied SOMs to visualize the software systems represented using different coupling measures, while to answer RQ2 we performed a correlation analysis.

As experimental case studies, we used Common DBUtils,a reinforcement learning framework and Apache Collections.

The SOMss revealed different mappings, while the correlations analysis resulted in small Pearson and Spearman correlations between ACE and the other coupling measures, which both confirm that ACE differs from existing coupling measures.

4.2.4 Change impact analysis using ACE

4.2.4.1 Data sets

With the major goal of answering our third research question RQ3, we discarded the reinforcement learning framework, since it has only one version, but we added all the available versions in SVN repositories for DButils and Apache Collections.

The following table summarizes their contents.

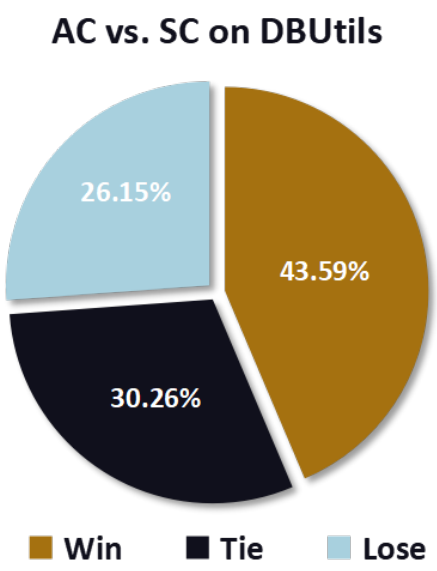
Software system	Number of versions	Total number of commits	Total number of changed classes
DbUtils	8	54	366
Apache Collections	10	2953	9026

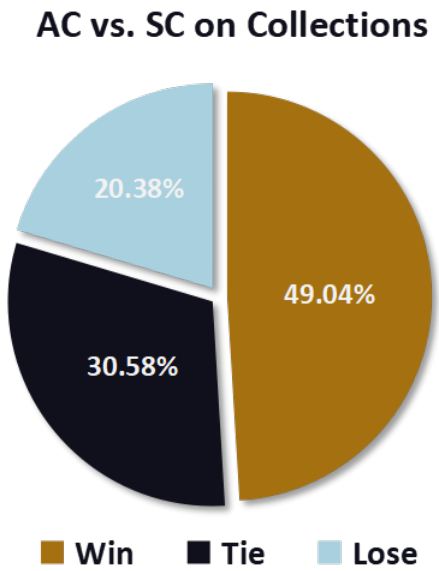
4.2.4.2 Experimental goal

The practical intuition behind the experiment is that by analyzing a commit, one can asses the impact of a change. On the other hand, when a software developer makes a change, he will usually look for the impact of that change based on direct (or structural) coupling (in all the classes that use or define the changed method). So, our goal comes down to showing that guiding the exploration based on the aggregated coupling is more efficient, in the sense that it reduces the number of classes which must be analyzed to identify all those that are impacted.

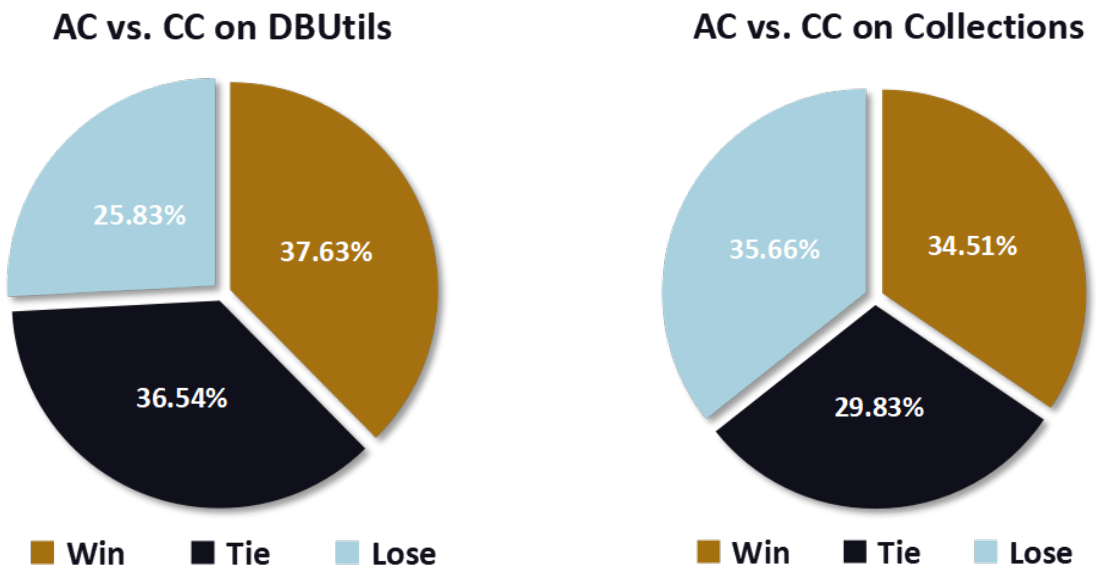
4.2.5 Results

Overall, AC outperformed or tied the structural coupling measures in about 74% of the cases on DBUtils. For Collections, we obtained an even higher percentage to our advantage, of almost 80%.





We have also compared AC with the conceptual coupling measure based on LSI [91], in terms or their efficacies in assessing the impact of the changes. AC proved a higher performance on DBUtils, but was slightly outperformed on Collections.



4.3 Conclusions and future work

We have presented in this chapter our original proposals for new coupling measures.

We have approached ESC by introducing a new ML based conceptual coupling measure, named ConC, and a novel aggregated coupling measure, named ACE, which aggregates structural and conceptual couplings.

The experimental results emphasize that ConC captures other dimensions of coupling than the structural measures and is also more useful for software restructuring at package level and that ACE generally outperforms the classical direct dependency measures, as well as a related conceptual coupling measure, in indicating the software entities that are impacted by software changes.

ConC and ACE can be used for both procedural and object-oriented software systems. However, the experimental case studies have been performed on object-oriented software systems. Therefore, the first direction of further work is to extend the experimental evaluations on other open-source case studies, corresponding to different programming paradigms.

We also aim to investigate using a method level granularity, instead of the class level granularity currently used in our proposal, as well as new ways to aggregate different types of coupling measures.

Chapter 5

Software Coupling as support for Software Defects Prediction

The current chapter is concerned with our original contributions towards using coupling as support for identifying defective software components. These contributions consist of proposing a new hybrid software defects predictor, based on conceptual features learned from the source code [73] and introducing a new conceptual coupling based metrics suite as support for Software Defects Prediction (SDP) [77]. So, as the last chapter of this thesis, it links the two Software Engineering (SE) problems approached, SDP and Estimation of Software Coupling (ESC).

Section 5.1 introduces our original approach to SDP in which a new variation of our hybrid classification model combining Gradual Relational Association Rules (GRARs) mining with Artificial Neural Networks (ANNs), named DePSem [73], detects defective software entities based on conceptual features learned from the source code. The section also includes DePSem's evaluation on 7 SDP data sets corresponding to open source software systems.

The new conceptual coupling based metrics suite for supporting SDP, named COMET [77], is presented in Section 5.2. The metrics suite is experimentally compared with conventional software metrics, in terms of the relevance for identifying defective software components.

Section 5.3, the ending section of this chapter, draws the chapter's conclusions and gives directions for future research.

The approaches presented in the current chapter are original research works published in [73] and [77].

5.1 DePSem: A hybrid software defects predictor based on conceptual features learned from the source code

This section opens the investigation on the relevance of conceptual coupling in assessing the proneness to software defects. It presents a hybrid classification model combining GRARs with ANNs that we have proposed for detecting the defective software entities based on semantic (or conceptual) features automatically learned from the source code [73].

5.1.1 Motivation

Software coupling strongly relates to the quality of the software design and software defects are effects of poor software quality, including poor design. So, design flaws make a software being defects-prone [33]. In other words, most defects are caused by violated dependencies, software coupling indicating technical dependencies. Consequently, there is likely to be a relationship

between software coupling and software defects. However, there are only few approaches in the literature that consider non-structural coupling as support for SDP [18, 52].

In these conditions, we investigated the interplay between conceptual coupling and software defects. Through this, we also followed the second research direction encountered in Software Defects Prediction (SDP) literature, which is designing new features encoding relevant characteristics of software systems [109], in addition to proposing novel and improved Machine Learning (ML) algorithms (HyGRAR [75] and HyGRAR* [69]), which are based on widely used, manually engineered software metrics.

5.1.2 Methodology

The DePSem approach involves two main steps: an unsupervised **data collection** step and a supervised **classification** step.

During data collection, the source code afferent to the software entities is transformed into numeric vectors of conceptual features extracted using Doc2Vec [56] and Latent Semantic Indexing (LSI) [34].

The **classification** step of DePSem is divided into three successive phases:

1) **mining** sets of interesting GRARs that discriminate between defective and non-defective software entities,

2) **training** Multilayer Perceptrons (MLPs) to detect defective entities based on the GRARs mined at step 1),

3) **testing** if a software instance is defective or non-defective using the classification model built at step 2).

Comparing DePSem with HyGRAR and HyGRAR*, DePSem uses semantic features automatically extracted from the source code, while the other two classifiers are based on widely used software metrics. On the other hand, ANNs are not employed here to learn gradual relations - we predefined them, instead - but they are still employed to learn the classification rule, as in HyGRAR*.

5.1.3 Experiments and results

5.1.3.1 Data sets

For evaluating DePSem we used seven SDP data sets (JEdit 3.2, JEdit 4.0, JEdit 4.1, JEdit 4.2, JEdit 4.3, Ant 1.7 and Tomcat 6.0) corresponding to object-oriented software systems.

For our experiments, we have transformed the number of bugs into a binary feature denoting if the entity is defective or not ant we replaced the software metrics with the semantic features learned during data collection, after a minimal pre-processing of the source code.

5.1.3.2 Evaluation

For testing DePSem's classification performance, we used the same evaluation methodology as for the other hybrid defect predictors. In addition, we have visualized the data, represented as conceptual vectors, using the t-SNE technique.

5.1.3.3 Results

Figures 5.1, 5.2, 5.3 and 5.4 give the t-Distributed Stochastic Neighbour Embedding (t-SNE) [107] visualizations of the data corresponding to Ant and Tomcat systems as well as to JEdit versions 3.2 and 4.3, when represented using Doc2Vec.

It is interesting to note that many defective classes, coloured in yellow, are located in the same sparser regions , while in very crowded portions there are no, or there are very few, defective classes. This could indicate that the software components that are highly coupled with others that are also highly coupled among themselves are less likely to be defective. We have also found support for this observation in the literature [18]. And the intuition is that as consistently inter-coupled software components emerge developers become aware of them, so they know how changes propagate and where to look in order to reduce the probability of introducing defects elsewhere.

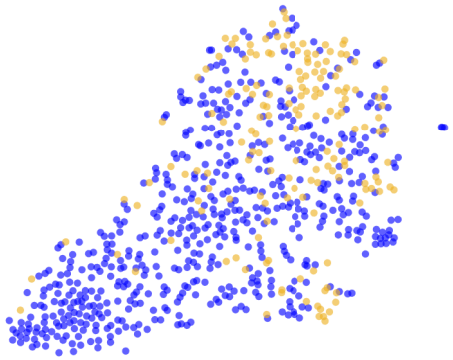


Figure 5.1: t-SNE representation for Ant 1.7 using the conceptual vectors learned by Doc2Vec

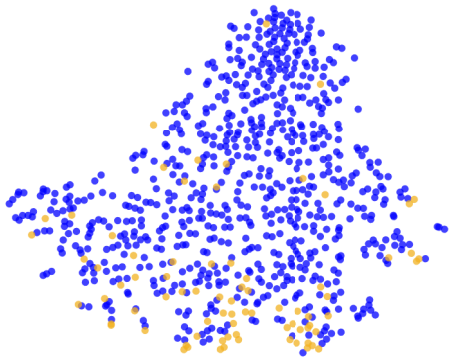


Figure 5.2: t-SNE representation for Tomcat 6.0 using the conceptual vectors learned by Doc2Vec

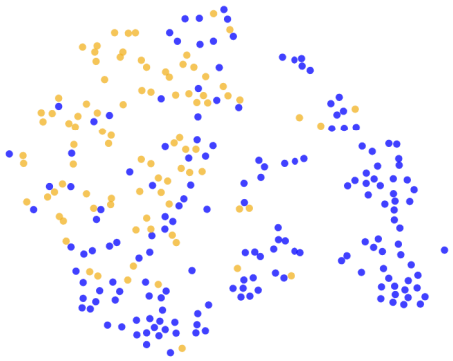


Figure 5.3: t-SNE representation for JEdit 3.2 using the conceptual vectors learned by Doc2Vec

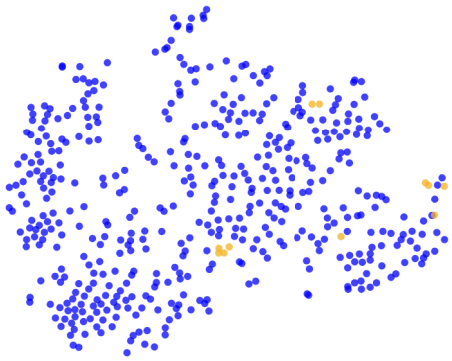


Figure 5.4: t-SNE representation for JEdit 4.3 using the conceptual vectors learned by Doc2Vec

When comparing using Doc2Vec, LSI or both of them for extracting the conceptual features, as we can observe in Figure 5.5, for most of the considered software systems, the combined representation provided the highest Area Under the Receiver Operating Characteristics (ROC) Curve (AUC) values.

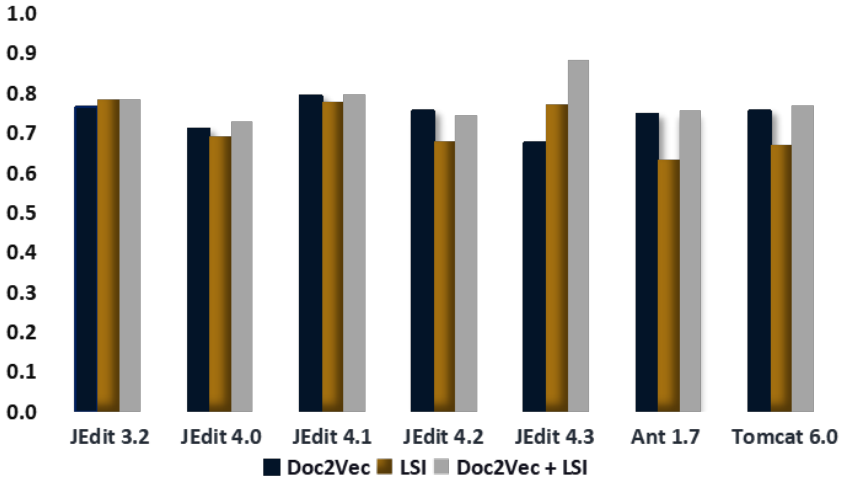


Figure 5.5: Comparative AUC values obtained when using Doc2Vec, LSI or both of them

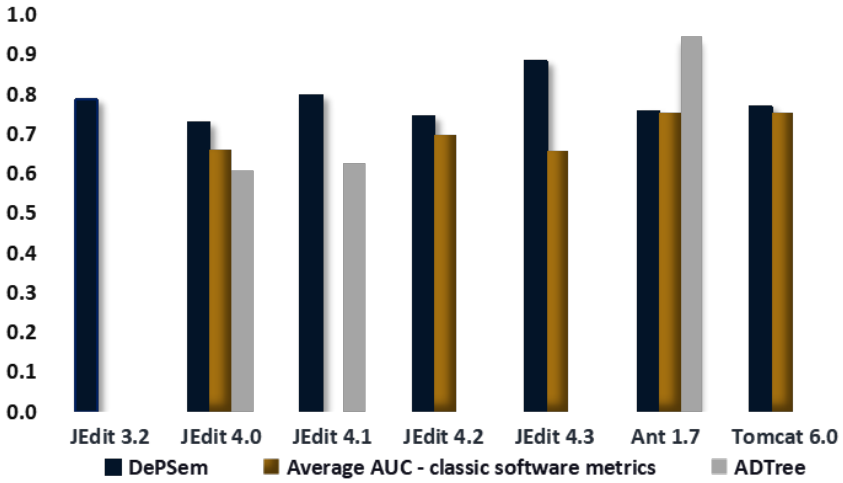


Figure 5.6: DePSem's comparison with related work

5.1.4 Comparison to related work

The SDP literature contains numerous approaches for detecting software faults using ML based classifiers, but most of these use conventional metrics including ones for expressing structural relationships between software components. There are only few approaches to SDP that involve other types of coupling [18, 52, 51]. They are related to DePSem only in the sense that it exploits dependencies different than the ones expressed by the widely used traditional coupling measures. The main difference consists in the fact that the semantic features we use are automatically learned from the source code, whereas the change coupling measures are manually designed based on historical data.

As depicted in Figure 5.6, compared to the defect predictors which are using software metrics [75], DePSem outperforms the average AUC on all case studies. DePSem also outperforms the approach based on semantic features extracted from the programs' Abstract Syntax Trees (ASTs), introduced [109], on JEdit 4.0 and JEdit 4.1. We mention that, unlike in [109], we have not reduced the difficulty of the data sets by eliminating instances with possibly incorrect labels and that the missing bars in Figure 5.6 are due to the fact that no results were reported in the literature for those case studies.

5.2 COMET: A conceptual coupling based metrics suite for software defects prediction

This section briefly presents our new conceptual coupling based metric suite, called COMET, we proposed as support for SDP.

5.2.1 Motivation

The choice of the software metrics to be used as features for SDP turned out to have a higher impact on the prediction performance than the selection of the classification technique, at least when it comes to simple, popular classification models [92]. Yet, many of the existing software defect predictors are based on the widely used but also criticized Promise [97] software metrics [41]. Given the impact of choosing the features for SDP and the experimental results presented in the previous section, we aim to propose new conceptual coupling based software metrics to support defects prediction.

5.2.2 The COMET metrics suite

The steps performed for defining COMET metrics suite are the following:

1. Firstly, conceptual features are unsupervisedly learned from the source code using Doc2Vec [56] and LSI [20].

2. The second step is to compute the conceptual coupling between each pair of software entities, as the similarity between their conceptual vectors. For expressing the closeness of two high dimensional vectors, we are using both *cosine* and *euclidean* similarities.

3. For each software entity a numerical sequence is built considering the entity's conceptual coupling values computed with respect to: (1) all the other entities from the software system; (2) the other software entities that are *defective* and (3) the other software entities that are *non-defective*.

4. The last step performed is to define the software metrics from the COMET suite as the *mean*, *maximum* and *standard deviation* descriptive statistic measures computed for the sequences built at step 3 for each software entity.

Following the previously introduced methodology (steps 1-4), 36 ($2 \cdot 2 \cdot 3 \cdot 3$) software measures are obtained. These 36 measures compose the COMET suite of software metrics suitable for SDP.

5.2.3 Experiments and results

The experiments aim to compare the COMET metrics with the Promise software metrics [97].

5.2.3.1 Data sets

We experimented on the same 7 software systems used for evaluating DePSem, but here, for each of the Promise data sets, we have built an additional data set in which each software entity from the original data set is represented as an 36-dimensional numerical vector whose components are the COMET conceptual coupling based metrics.

5.2.3.2 Correlation based analysis

In order to assess the relevance for SDP of the COMET metrics, we started by comparatively analyzing the correlation between COMET versus Promise metrics and the presence of software defects.

The analysis empirically highlighted that, on the data sets used in our experiments, the COMET metrics are more correlated with the defect proneness than the classical Promise metrics. The overall Pearson correlation between the COMET metrics and the defectiveness of the software entities is 19.67% higher than the overall correlation between the Promise metrics and the presence of defects (0.238 vs 0.199). In the case of the Spearman correlation, this relative percentage is 7.34% (0.237 vs 0.221).

5.2.3.3 Unsupervised learning based analysis

Secondly, we compared COMET suite and Promise metrics set, in terms of their relevance for SDP, through unsupervised learning. For this, we used t-Distributed Stochastic Neighbour Embedding (t-SNE) [107]. We strengthened the strictly visual comparison with a numerical comparison based on the difficulty measure.

The overall classification difficulty [116] is reduced for 6 of the 7 case studies when replacing the Promise metrics with the COMET ones, while the difficulty of distinguishing the defective instances is significantly reduced for all the case studies. These numerical results confirm that, according to the unsupervised learning based analysis involving t-SNE, the COMET metrics suite is more relevant than the Promise metrics set for discriminating between defective and non-defective software components.

5.2.3.4 Supervised learning based analysis

The comparison between the COMET and the Promise metrics has been completed by an experimental study performed from a supervised SDP perspective. We selected 3 different ML methods for supervised classification: Logistic Regression, k-Nearest Neighbors (kNN) and Artificial Neural Network (ANN). The evaluation methodology we applied is Leave-One-Out (LOO).

Overall, when using all the 36 COMET metrics instead of the 20 widely used Promise metrics as features for SDP, the AUC is increased with more than 11%.

So, the supervised learning based comparison reinforces the conclusion that the COMET metrics suite is relevant for SDP, outperforming, on the 7 case studies considered, the widely used Promise software metrics.

5.2.4 Comparison to related work

Figure 5.7 compares the performance of the COMET based SDP models with the performances obtained, in the literature, using the Promise metrics and with the DePSem's performance.

The comparison reveales that the average performance obtained when considering COMET metrics exceeds both the DePSem's performance and the average performance of the Promise metrics based approaches.

5.3 Conclusions and future work

The current chapter presented our novel approach to SDP based on conceptual features extracted automatically from the source code and a conceptual coupling based metrics suite for

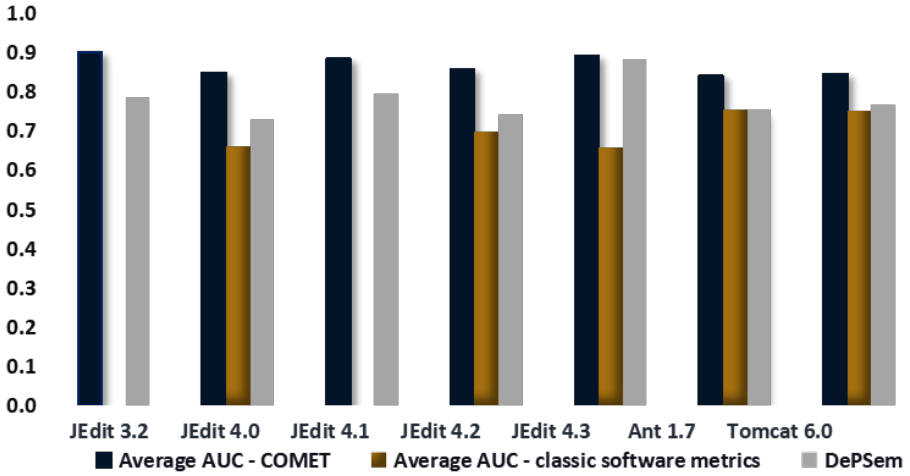


Figure 5.7: Comparison with related work

SDP. Both aimed at emphasizing the relevance conceptual coupling has for predicting defective software components.

The experiments performed have confirmed that conceptual coupling impacts the proneness to software defects.

The conceptual coupling based software metrics have been compared with well known SDP metrics, from three different perspectives: one of a correlation analysis, one of unsupervised learning and one of supervised learning. All these comparisons pointed out that the metrics derived from conceptual coupling are superior, in terms of the predictive performance, to the standard software metrics widely used in the SDP literature.

As a first direction of further work, we aim to extend the experimental evaluation by considering different granularities (e.g. method-level defect prediction) and additional programming languages. As a second direction, future work will be conducted in order to extend COMET for capturing the structural coupling as well. This research direction is motivated by the complementarity of structural and conceptual coupling [18], while aggregated coupling [32] turned out to be effective in expressing software dependencies.

Conclusions

This thesis have presented our conceptual contributions to Machine Learning (ML) along with our contributions to the applied research in Software Engineering (SE).

We have approached two SE problems, namely Software Defects Prediction (SDP) and Estimation of Software Coupling (ESC), both being essential to effectively ensure software quality. Software coupling, for instance, supports software restructuring and change impact analysis, thus facilitating good structuring and understanding. In this regard, we have introduced new ML based conceptual and aggregated coupling measures. Moreover, coupling may affect the defect proneness of software components, which we confirmed by successfully proposing new coupling based features to be used as support for SDP. Through this, we followed one of the two main research directions in the very active research area of SDP, which is proposing new relevant software features. The other direction is developing new and improved classification models and we followed it too, by proposing novel hybrid approaches that combine Gradual Relational Association Rules (GRARs) mining with Artificial Neural Networks (ANNs).

Our hybrid approaches are, at the same time, original fundamental contributions to ML, as well as introducing GRARs mining, together with adaptive and dynamic versions of the mining algorithm. GRARs mining generalizes Relational Association Rules (RARs) mining by enriching it with more expressive power and increased stability to noise, while the adaptive and dynamic versions of the GRARs mining algorithm make the GRARs discovery process significantly more efficient when mining dynamic data. GRARs mining is integrated in original hybrid classifications models which also employ ANNs to learn gradual relations (HyGRAR and HyGRAR*) and / or to classify an instance based on interesting GRARs characterizing the classes (HyGRAR* and DePSem).

As already mentioned, we have capitalized our fundamental contributions so as to also contribute to the applied research in SE. So, first we have approached SDP by applying HyGRAR and HyGRAR* hybrid models as intra-project or inter-projects software defects predictors that proved to have an excellent classification performance for software systems developed either in object-oriented or non-objected-oriented programming languages. Then, we approached ESC by defining new ML based conceptual and aggregated coupling measures which have been validated in terms of their utility for software restructuring and software change impact analysis. Eventually, we have linked SDP and ESC by successfully using conceptual coupling based software features as foundations for predicting defective software entities, while also involving a hybrid model that use GRARs mining in conjunction with ANNs (DePSem).

To sum up, our work contributed to both ML and Search Based Software Engineering (SBSE). The applicability of our conceptual contributions is not limited to the SBSE research area but so far we have proven it in this area. From a different perspective, we have shown that SBSE can benefit from our conceptual contributions, in particular from the reunited power of GRARs mining and ANNs, in the form of hybrid ML models.

The SBSE problems we have addressed, mainly SDP and ESC but also software restructuring and change impact analysis that we have investigated as applications for the newly proposed coupling measures, can be framed in a more general problem, namely software quality assessment, which is essential for ensuring high-quality software. The center of our contributions in SBSE

remains, however, SDP since ultimately both our hybrid ML models and software coupling have been availed for predicting software defects.

In addition to the future research directions indicated separately for each chapter, we also identify other directions of further investigation which could positively impact software quality, particularly through SDP. We present them in the following.

A first direction of further research would be to define new ML based cohesion measures and to investigate their usefulness in assessing software quality, including through SDP. For instance, we aim to define conceptual cohesion measures based on Doc2Vec representations of the source code and to include them, along with structural cohesion measures, in an aggregated cohesion measure.

Since coupling and cohesion are complementary in assessing software design quality, a second future research idea is to unite their power by developing a new extensive metrics suite for SDP, composed of new metrics derived from aggregated coupling and aggregated cohesion.

Regarding the classification models involved in predicting software defects, further research might explore applying HyGRAR in an one-class learning (or anomaly detection) scenario [94], which is particularly suitable when dealing with imbalanced data. This would imply to learn only the interesting GRARs that characterize the majority class (in our case, the non-defective one) and to base the classification exclusively on them.

Furthermore, to align with the dynamic nature of the software development process and thus to bring our solutions closer to the industrial reality, we aim to approach dynamic SDP by incorporating Dynamic Gradual Relational Association Rules Miner (DynGRAR) into HyGRAR. Thus, the set of interesting GRARs on which the classification is based would be adapted dynamically and efficiently as the software system evolves.

Bibliography

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

[2] Abdi, H., and Williams, L. J. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics 2*, 4 (2010), 433–459.

[3] AFZAL, W., TORKAR, R., and FELDT, R. Resampling methods in software quality classification. *International Journal of Software Engineering and Knowledge Engineering 22*, 02 (2012), 203–223.

[4] Agarwal, S. *Data mining: Data mining concepts and techniques*. Morgan Kaufmann Publishers Inc., 2014.

[5] Agrawal, R., and Srikant, R. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases* (1994), Morgan Kaufmann Publishers Inc., pp. 487–499.

[6] Ajienka, N., and Capiluppi, A. Understanding the interplay between the logical and structural coupling of software classes. *Journal of Systems and Software 134*, Supplement C (2017), 120–137.

[7] Akbarinasaji, S., Soltanifar, B., Çağlayan, B., Bener, A. B., Miranskyy, A., Filiz, A., Kramer, B. M., and Tosun, A. A metric suite proposal for logical dependency. In *Proceedings of the 7th International Workshop on Emerging Trends in Software Metrics - WETSoM* (2016), ACM, pp. 57–63.

[8] Arisholm, E., Briand, L. C., and Foyen, A. Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering 30*, 8 (2004), 491–506.

[9] Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., and De Lucia, A. An empirical study on the developers perception of software coupling. In *Proceedings of the 2013 International Conference on Software Engineering* (2013), IEEE Press, pp. 692–701.

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

[11] Borzecka, H. Multi-criteria decision making using fuzzy preference relations. *Operations Research and Decisions 3* (2012), 5–21.

[12] Briand, L., Wust, J., and Lounis, H. Using coupling measurement for impact analysis in object-oriented systems. In *Proceedings of IEEE International Conference on Software Maintenance - ICSM. 'Software Maintenance for Business Change'* (1999), IEEE Computer Society, pp. 475–482.

[13] Briand, L. C., and Daly, J. W. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering 25*, 1 (1999), 91–121.

[14] Broomhead, D. S., and Lowe, D. Multivariable functional interpolation and adaptive networks. *Complex Systems 2* (1988), 321–355.

[15] Campan, A., Serban, G., and Marcus, A. Relational association rules and error detection. *Studia Universitatis Babes-Bolyai Informatica LI*, 1 (2006), 31–36.

[16] Canfora, G., Lucia, A. D., Penta, M. D., Oliveto, R., Panichella, A., and Panichella, S. Multi-objective cross-project defect prediction. In *Proceedings of the 6th International Conference on Software Testing, Verification and Validation* (2013), pp. 252–261.

[17] Catal, C., Sevim, U., and Diri, B. Software fault prediction of unlabeled program modules. In *Proceedings of the World Congress on Engineering* (2009), pp. 1–6.

[18] Cataldo, M., Mockus, A., Roberts, J. A., and Herbsleb, J. D. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering 35*, 6 (2009), 864–878.

[19] Chang, H. Y., Lin, J. C., Cheng, M. L., and Huang, S. C. A novel incremental data mining algorithm based on FP-growth for big data. In *Proceedings of the International Conference on Networking and Network Applications - NaNA* (2016), pp. 375–378.

[20] Chen, H., Martin, B., Daimon, C., and Maudsley, S. Effective use of latent semantic indexing and computational linguistics in biological and biomedical applications. *Frontiers in Physiology 4* (2013), 8.

[21] Chidamber, S. R., and Kemerer, C. F. Towards a metrics suite for object oriented design. *Special Interest Group on Programming Languages Notices 26*, 11 (1991), 197–211.

[22] Clark, B., and Zubrow, D. How good is a software: a review on defect prediction techniques. In *Software Engineering Symposium* (Carneige Mellon University, 2001), pp. 1–35.

[23] Cojocar, G. S., and Şerban, G. On some criteria for comparing aspect mining techniques. In *Proceedings of the 3rd Workshop on Linking Aspect Technology and Evolution - LATE* (2007), Association for Computing Machinery, p. 1–5.

[24] Czibula, G., Bocicor, M. I., and Czibula, I. G. Promoter sequences prediction using relational association rule mining. *Evolutionary Bioinformatics*, 8 (2012), 181–196.

[25] Czibula, G., Czibula, I. G., Miholca, D.-L., and Crivei, L. M. A novel concurrent relational association rule mining approach. *Expert Systems with Applications 125* (2019), 142 – 156.

[26] Czibula, G., Czibula, I. G., Sîrbu, A.-M., and Mircea, I.-G. A novel approach to adaptive relational association rule mining. *Applied Soft Computing 36* (2015), 519–533.

[27] Czibula, G., Marian, Z., and Czibula, I. G. Software defect prediction using relational association rule mining. *Information Sciences 264* (2014), 260 – 278.

[28] Czibula, G., Marian, Z., and Czibula, I. G. Detecting software design defects using relational association rule mining. *Knowledge and Information Systems 42*, 3 (2015), 545–577.

[29] Czibula, I., Czibula, G., Miholca, D., and Marian, Z. Identifying hidden dependencies in software systems. *Studia Universitatis Babeș-Bolyai Informatica 62*, 1 (2017), 90–106.

[30] Czibula, I. G., and Czibula, G. Improving systems design using a clustering approach. *IJCSNS International Journal of Computer Science and Network Security 6*, 12 (2006), 40–49.

[31] Czibula, I. G., Czibula, G., and Miholca, D. L. Enhancing relational association rules with gradualness. *International Journal of Innovative Computing, Information and Control 13*, 1 (2017), 289–305.

[32] Czibula, I. G. I., Czibula, G., Miholca, D.-L. D.-L., and Onet-Marian, Z. An aggregated coupling measure for the analysis of object-oriented software systems. *Journal of Systems and Software 148* (2019), 1–20.

[33] D'Ambros, M., Bacchelli, A., and Lanza, M. On the impact of design flaws on software defects. In *Proceedings of the 10th International Conference on Quality Software* (2010), pp. 23–31.

[34] Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41* (1990), 391–407.

[35] Dhanabhakyam, M., and M., P. An efficient market basket analysis based on adaptive association rule mining with faster rule generation algorithm. *The SIJ Transactions on Computer Science Engineering and its Applications - CSEA 1*, 3 (2013), 1–6.

[36] e Abreu, F. B. The MOOD metrics set. In *Proceedings of the European Conference on Object-Oriented Programming - ECOOP* (1995), pp. 150–152.

[37] Fawcett, T. An introduction to ROC analysis. *Pattern Recognition Letters 27*, 8 (2006), 861–874.

[38] Francisco, L., Armando, B., Fernando, G., Carlos, C., and Antonio, M. Fuzzy association rules for biological data analysis: A case study on yeast. *BMC Bioinformatics 9*, 1 (2008), 107.

[39] Gagolewski, M., and Lasek, J. The use of fuzzy relations in the assessment of information resources producers' performance. In *Advances in Intelligent Systems and Computing* (2015), vol. 323, pp. 289–300.

[40] Gosain, A., and Bhugra, M. A comprehensive survey of association rules on quantitative data in data mining. In *Proceedings of the 2013 IEEE Conference on Information and Communication Technologies - ICT* (2013), pp. 1003–1008.

[41] Grady, R. B. *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall Press, 1992.

[42] Grossman, D. A., and Frieder, O. *Information Retrieval: Algorithms and Heuristics*, vol. 461. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[43] Hall, T., Beecham, S., Bowes, D., Gray, D., and Counsell, S. A systematic review of fault prediction performance in software engineering. *IEEE Transactions on Software Engineering 38*, 6 (2011), 1276–1304.

[44] Harman, M., McMinn, P., de Souza, J. T., and Yoo, S. Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical Software Engineering and Verification: International Summer Schools, LASER 2008-2010, Revised Tutorial Lectures* (2012), Springer, pp. 1–59.

[45] Hryszko, J., and Madeyski, L. Cost effectiveness of software defect prediction in an industrial project. *Foundations of Computing and Decision Sciences 43*, 1 (2018), 7 – 35.

[46] hua Chang, R., Mu, X., and Zhang, L. Software defect prediction using non-negative matrix factorization. *Journal of Software - JSW 6*, 11 (2011), 2114–2120.

[47] Hüllermeier, E. Association rules for expressing gradual dependencies. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery* (2002), pp. 200–211.

[48] Kagdi, H., Gethers, M., Poshyvanyk, D., and Collard, M. L. Blending conceptual and evolutionary couplings to support change impact analysis in source code. In *2010 17th Working Conference on Reverse Engineering* (2010), pp. 119–128.

[49] Kaski, S., and Kohonen, T. Structures of welfare and poverty in the world discovered by self-organizing maps. In *Neural Networks in Financial Engineering. Proceedings of the Third International Conference on Neural Networks in the Capital Markets* (1996), World Scientific, pp. 498–507.

[50] Khanesar, M. A., and Teshnehlab, M. Direct fuzzy model reference controller for siso nonlinear plants using observer. *International Journal of Innovative Computing, Information and Control 6*, 1 (2010), 297–306.

[51] Kirbas, S., Caglayan, B., Hall, T., Counsell, S., Bowes, D., Sen, A., and Bener, A. The relationship between evolutionary coupling and defects in large industrial software. *Software: Evolution and Process 29*, 4 (2017), 1–19.

[52] Kirbas, S., Sen, A., Caglayan, B., Bener, A., and Mahmutogullari, R. The effect of evolutionary coupling on software defects: An industrial case study on a legacy system. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2014), ACM, pp. 1–7.

[53] Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics 43*, 1 (1982), 59–69.

[54] Ksantini, M., Hammami, M. A., and Delmotte, F. On the global exponential stabilization of Takagi-Sugeno fuzzy uncertain systems. *International Journal of Innovative Computing, Information and Control 11*, 1 (2015), 281–284.

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

[56] Le, Q. V., and Mikolov, T. Distributed representations of sentences and documents. *Computing Research Repository (CoRR) abs/1405.4* (2014), 1–9.

[57] Lee, Y. S., Liang, B. S., Wu, S. F., and Wang, F. J. Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow. In *Proceedings of the International Conference of Software Quality - ICSQ* (1995), pp. 81–90.

[58] Levy, O., Goldberg, Y., and Dagan, I. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics - TACL 3* (2015), 211–225.

[59] Li, W., and Henry, S. OO metrics which predict maintainability. *Journal of Systems and Software 23*, 2 (1993), 111–122.

[60] Li, Y., Zhang, Z.-H., Chen, W.-B., and Min, F. TDUP: an approach to incremental mining of frequent itemsets with three-way-decision pattern updating. *International Journal of Machine Learning and Cybernetics 8*, 2 (2017), 441–453.

[61] Liu, Y., and Milanova, A. Static analysis for dynamic coupling measures. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research - CASCON* (2006), pp. 10–21.

[62] L.P.F.a, G., de Carvalho A.C.P.L.F.a, and A.C.b, L. Effect of label noise in the complexity of classification problems. *Neurocomputing 160* (2015), 108–119.

[63] Ma, B., Dejaeger, K., Vanthienen, J., and Baesens, B. Software defect prediction based on association rule classification. In *Proceedings of the 2010 International Conference on E-Business Intelligence* (2010), Atlantis Press, pp. 396–402.

[64] Malhotra, R. A defect prediction model for open source software. In *Proceedings of the World Congress on Engineering* (2012), vol. II, pp. 1–5.

[65] Malhotra, R. Comparative analysis of statistical and machine learning methods for predicting faulty modules. *Applied Soft Computing 21* (2014), 286–297.

[66] Marcus, A., Maletic, J. I., and Lin, K.-I. Ordinal association rules for error identification in data sets. In *Proceedings of the tenth International Conference on Information and Knowledge Management - CIKM* (2001), ACM, pp. 589–613.

[67] Marian, Z., Mircea, I., Czibula, I., and Czibula, G. A novel approach for software defect prediction using fuzzy decision trees. In *Proceedings of the 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing - SYNASC* (2016), pp. 240–247.

[68] Marin, M., Deursen, A. V. A. N., and Moonen, L. Identifying crosscutting concerns using fan-in analysis. *ACM Transactions on Software Engineering and Methodology 17*, 1 (2007), 1–37.

[69] Miholca, D. An improved approach to software defect prediction using a hybrid machine learning model. In *Proceedings of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing - SYNASC* (2018), pp. 443–448.

[70] Miholca, D., Czibula, G., Zsuzsanna, M., and Czibula, I. G. An unsupervised learning based conceptual coupling measure. In *Proceedings of the 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing - SYNASC* (2017), pp. 247–254.

[71] Miholca, D. L. An adaptive gradual relational association rules mining approach. *Studia Universitatis Babes-Bolyai Series Informatica 63*, 1 (2018), 94–110.

[72] Miholca, D.-L., and Czibula, G. DynGRAR: A dynamic approach to mining gradual relational association rules. In *Proceedings of the 23th Knowledge-Based and Intelligent Information and Engineering Systems - KES* (2019), pp. 10 – 19.

[73] Miholca, D.-L., and Czibula, G. Software defect prediction using a hybrid model based on semantic features learned from the source code. In *Proceedings of the International Conference on Knowledge Science, Engineering and Management - KSEM* (2019), Springer International Publishing, pp. 262–274.

[74] Miholca, D.-L., Czibula, G., and Crivei, L. M. A new incremental relational association rules mining approach. In *Proceedings of the 22nd Knowledge-Based and Intelligent Information and Engineering Systems - KES* (2018), pp. 126 – 135.

[75] Miholca, D.-L., Czibula, G., and Czibula, I. A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks. *Information Sciences 441* (2018), 152 – 170.

[76] Miholca, D. L., Czibula, G., Mircea, I. G., and Czibula, I. G. Machine learning based approaches for sex identification in bioarchaeology. In *Proceedings of the 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing - SYNASC* (2016), pp. 311–314.

[77] Miholca, D.-L., Czibula, G., and Tomescu, V. COMET: A conceptual coupling based metrics suite for software defect prediction. In *Proceedings of the 24th Knowledge-Based and Intelligent Information and Engineering Systems International Conference - KES* (2020), pp. 1 – 10. accepted for publication.

[78] Miholca, D. L., and Onicaş, A. Detecting depression from fMRI using relational association rules and artificial neural networks. In *Proceedings of the 2017 IEEE 13th International Conference on Intelligent Computer Communication and Processing - ICCP* (2017), pp. 85–92.

[79] Mitchell, T. M. *Machine Learning*, 1 ed. McGraw-Hill, Inc., 1997.

[80] Muthukumaran, K., Srinivas, S., Malapati, A., and Neti, L. B. M. Software defect prediction using augmented bayesian networks. *Advances in Intelligent Systems and Computing 614*, 1 (2018), 279–293.

[81] Nam, J., and Kim, S. Heterogeneous defect prediction. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (2015), ACM, pp. 508–519.

[82] Nath, B., Bhattacharyya, D. K., and Ghosh, A. Incremental association rule mining: A survey. *Interdisciplinary Reviews: Data Mining and Knowledge Discovery 3*, 3 (2013), 157–169.

[83] Nuñez-varela, A. S., Pérez-gonzalez, H. G., Martínez-perez, F. E., and Soubervielle-montalvo, C. Source code metrics : A systematic mapping study. *The Journal of Systems and Software 128*, Supplement C (2017), 164–197.

[84] Ogunde, A. O., Folorunso, O., and Sodiya, A. S. The design of an adaptive incremental association rule mining system. In *Proceedings of the World Congress on Engineering 2015, Volume I* (2015), pp. 1–6.

[85] Ougli, A. E., and Tidhaf, B. Optimal type-2 fuzzy adaptive control for a class of uncertain nonlinear systems using an LMI approach. *International Journal of Innovative Computing, Information and Control 11*, 3 (2015), 551–563.

[86] P. Tang M. Steinbach, A. K., and Kumar, V. *Introduction to data mining, 2nd Edition.* Addison-Wesley Longman Publishing Co., Inc., 2019.

[87] Panichella, A., Oliveto, R., and Lucia, A. D. Cross-project defect prediction models: L'union fait la force. In *Proocedings of the IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering - CSMR-WCRE* (2014), pp. 164–173.

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

[89] Perepletchikov, M., Ryan, C., Frampton, K., and Tari, Z. Coupling metrics for predicting maintainability in service-oriented designs. In *Proceedings of the Australian Software Engineering Conference - ASWEC* (2007), pp. 329–338.

[90] Perreault L., Berardinelli S., I. C., and J., S. Using classifiers for software defect detection. In *https://www.cs.montana.edu/izurieta/pubs/SEDE_2017.pdf* (2017), pp. 1–8.

[91] Poshyvanyk, D., Marcus, A., Ferenc, R., and Gyimóthy, T. Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering 14*, 1 (2009), 5–32.

[92] Radjenović, D., Heričko, M., Torkar, R., and Živkovič, A. Software fault prediction metrics: A systematic literature review. *Information and Software Technology 55*, 8 (2013), 1397–1418.

[93] Ren, J., Li, W., Wang, Y., and Zhou, L. Graph-mine: A key behavior path mining algorithm in complex software executing network. *International Journal of Innovative Computing, Information and Control 11*, 2 (2015), 541–553.

[94] Rodionova, O. Y., Oliveri, P., and Pomerantsev, A. L. Rigorous and compliant approaches to one-class classification. *Chemometrics and Intelligent Laboratory Systems 159* (2016), 89 – 96.

[95] Rojas, R. *Neural Networks: A Systematic Introduction.* Springer-Verlag, 1996.

[96] Salam, A., and Khayal, M. S. H. Mining top-k frequent patterns without minimum support threshold. *Knowledge and Information Systems 30*, 1 (2012), 57–86.

[97] Sayyad, S., and Menzies, T. The PROMISE repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2015.

[98] Schwenker, F., Kestler, H. A., and Palm, G. Three learning phases for radial-basis-function networks. *Neural Networks 14*, 4-5 (2001), 439–458.

[99] Selvi, C. S. K., and Tamilarasi, A. Association rule mining with dynamic adaptive support thresholds for associative classification. In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications - ICCIMA* (2007), IEEE Computer Society, pp. 76–80.

[100] Serban, G., Câmpan, A., and Czibula, I. G. A programming interface for finding relational association rules. *International Journal of Computers, Communications and Control I*, S. (2006), 439–444.

[101] Serban, G., Czibula, I. G., and Câmpan, A. Medical diagnosis prediction using relational association rules. In *Proceedings of the International Conference on Theory and Applications of Mathematics and Informatics - ICTAMI* (2008), pp. 339–352.

[102] Simon, F., Löffler, S., and Lewerentz, C. Distance based cohesion measuring. In *proceedings of the 2nd European Software Measurement Conference - FESMA* (1999), pp. 69–83.

[103] Somervuo, P., and Kohonen, T. Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters 10*, 2 (1999), 151–159.

[104] Song, A., Ding, X., Li, M., Cao, W., and Pu, K. A novel binary bat algorithm for association rules mining. *ICIC Express Letters 9*, 9 (2015), 2387–2394.

[105] Stevens, W. P., Myers, G. J., and Constantine, L. L. Structured design. *IBM Systems Journal 13*, 2 (1974), 115–139.

[106] Tobergte, D. R., and Curtis, S. Data mining - practical machine learning tools and techniques. *Journal of Chemical Information and Modeling 53*, 9 (2013), 1689–1699.

[107] Van Der Maaten, L., and Hinton, G. Visualizing data using t-SNE. *Journal of Machine Learning Research 9* (2008), 2579–2605.

[108] Varade, S. Hyper-quad-tree based k-means clustering algorithm for fault prediction. *International Journal of Computer Application 76*, 5 (2013), 6–10.

[109] Wang, S., Liu, T., and Tan, L. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering* (2016), ACM, pp. 297–308.

[110] White, L., Jaber, K., Robinson, B., and Rajlich, V. Extended firewall for regression testing: An experience report. *Journal of Software Maintenance and Evolution 20*, 6 (2008), 419–433.

[111] Xuan, X., Lo, D., Xia, X., and Tian, Y. Evaluating defect prediction approaches using a massive set of metrics: An empirical study. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (2015), Association for Computing Machinery, p. 1644–1647.

[112] Yang, T., Asanjan, A. A., Faridzad, M., Hayatbini, N., Gao, X., and Sorooshian, S. An enhanced artificial neural network with a shuffled complex evolutionary global optimization with principal component analysis. *Information Sciences 418-419*, Supplement C (2017), 302–316.

[113] Yu, L., and Mishra, A. Experience in predicting fault-prone software modules using complexity metrics. *Quality Technology and Quantitative Management 9*, 4 (2012), 421–433.

[114] Yu-Dong, G., Sheng-Lin, L., Yong-Zhi, L., Zhao-Xia, W., and Li, Z. Large-scale dataset incremental association rules mining model and optimization algorithm. *International Journal of Database Theory and Application 9*, 4 (2016), 195–208.

[115] Zhang, C., Wu, X., Shyu, M.-L., and Peng, Q. Adaptive association rule mining for web video event classification. In *14th International Conference on Information Reuse and Integration* (2013), IEEE, pp. 618–625.

[116] Zhang, D., Tsai, J., and Boetticher, G. Improving credibility of machine learner models in software engineering. In *Advances in Machine Learning Applications in Software Engineering*. 2007, pp. 52–72.

[117] Zheng, J. Predicting software reliability with neural network ensembles. *Expert Systems with Applications 36*, 2 (2009), 2116–2122.