UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Heterogeneous information network to support the bug report resolution process

Jacson Rodrigues Barbosa

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Jacson Rodrigues Barbosa

Heterogeneous information network to support the bug report resolution process

Thesis submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Márcio Eduardo Delamaro

USP – São Carlos
January 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

Barbosa, Jacson Rodrigues
B238h Heterogeneous information network to support the bug report resolution process / Jacson Rodrigues Barbosa; orientador Márcio Eduardo Delamaro. -- São Carlos, 2021.
112 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2021.

1. Heterogeneous information network. 2. Bug report resolution process. 3. Software documents mining. 4. Data model representation. 5. Machine learning. I. Delamaro, Márcio Eduardo, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Jacson Rodrigues Barbosa

Rede de informações heterogênea para apoiar o processo de resolução de relatórios de incidentes

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Márcio Eduardo Delamaro

USP – São Carlos
Janeiro de 2022

This work is dedicated to the God who always
sustained me throughout this arduous journey.
Also to my beautiful family (Cris, Fefo, and Nono) who always supported me.

ACKNOWLEDGEMENTS

Firstly, I thank God for the daily sustenance and wisdom given along this journey. To my family (Cris, Fefo, and Nono) for their patience and support.

To professor Márcio Eduardo Delamaro for his teachings throughout this journey. To professors Auri M. R. Vincenzi, Ricardo Marcondes Marcacini, Ricardo Britto and Solange Rezende for their co-orientation and support.

To the Instituto de Ciências Matemáticas e de Computação for training, especially to the employees, teachers, and friends of LABES and LABIC. To the Universidade Federal de Goiás for the support. Finally, this work is the result of joint work; I thank everyone who contributed to the realization of this work.

“For the LORD gives wisdom;
from his mouth come knowledge and understanding.”
(Proverbs 2:6)

RESUMO

BARBOSA, J. R. **Rede de informações heterogênea para apoiar o processo de resolução de relatórios de incidentes**. 2022. 112 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Contexto. Ao longo do ciclo de vida de um software, inúmeros documentos (por exemplo, relatórios de incidentes e código fonte) são produzidos por stakeholders. Os relatórios de incidentes (RI) são os principais documentos de insumo para apoiar as atividades (predição da severidade de relatórios de incidentes e recomendação de responsáveis pela correção do software) do processo de resolução de relatórios de incidentes (RRI). Já o código fonte combinado com os relatórios de incidentes são insumos para apoiar atividades de localização de defeitos. A Automação dessas atividades do processo RRI requer uma preocupação em como obter uma representação semanticamente representativa. Tradicionalmente utiliza-se Bag-of-Word (BoW) para representar os documentos de software para apoiar a execução automática dessas atividades por meio de algoritmos de aprendizado de máquina. **Lacuna.** No entanto, pouca atenção foi dada para representações baseadas em redes de informações heterogêneas (RIH), que permitem representar redes complexas respeitando os relacionamentos entre diferentes objetos. **Contribuição.** Esta tese de doutorado contribui para o avanço do estado da arte no que se refere aos modelos de representação de informações para apoiar a execução automática de atividades do processo RRI. Também avança na investigação de (i) algoritmos semissupervisionados que utilizam redes heterogêneas bipartidas para apoiar a predição de severidade de RI, (ii) um método que combina a representação BoW e redes de informações heterogêneas para apoiar a atividade de localização de defeitos, e (iii) uma abordagem holística que reutilizar uma rede de informações heterogêneas para apoiar atividades de RRI. **Resultados.** Os resultados demonstram que redes de informações heterogêneas podem ser uma alternativa promissora para apoiar a automação do processo RRI. **Conclusões.** Um processo RRI automático numa perspectica holística utilizando rede de informações heterogêneas apresentou resultados promissores ao serem comparados com representações do estado da arte.

Palavras-chave: Rede de informações heterogêneas, Processo de resolução de relatórios de incidentes, Mineração de documentos de software, Aprendizado de máquina.

ABSTRACT

BARBOSA, J. R. **Heterogeneous information network to support the bug report resolution process**. 2022. 112 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Context. Throughout a software's lifecycle, numerous documents (e.g., bug reports and source code) are produced by stakeholders. Bug reports (BR) are the primary input documents to support the activities (bug report severity prediction and fixer recommendation) of the bug report resolution (BRR) process. Source code combined with bug reports are resources to support troubleshooting activities. Automation of these activities of the BRR process requires a concern with obtaining a semantically representative representation. Traditionally, Bag-of-Word (BoW) represents software documents to support the automatic execution of these activities through machine learning algorithms. **Gap.** However, little attention has been paid to representations based on heterogeneous information networks (HEN), which allow representing complex networks respecting the relationships between different objects. **Contribution.** This doctoral thesis contributes to advancing state of the art regarding information representation models to support the automatic execution of activities in the BRR process. It also advances in the investigation of (i) semi-supervised algorithms that use bipartite heterogeneous networks to support the bug report severity prediction, (ii) a method that combines the BoW representation and heterogeneous information networks to support the bug localization activity, and (iii) a holistic approach that reuses a heterogeneous information network to support BRR activities. **Results.** The results demonstrate that heterogeneous information networks can be a promising alternative to support the automation of the BRR process. **Conclusions.** An automatic BRR process using a heterogeneous information network in a holistic perspective presented promising results compared with state-of-the-art representations.

Keywords: Heterogeneous information network, Bug report resolution process, Software documents mining, Machine learning.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

BoW	Bag-of-Words
BTS	Bug Tracking System
BULNER	BUg Localization with word embeddings and NEtwork Regularization
CNN	Convolutional Neural Network
DBRNN-A	Deep Bidirectional Recurrent Neural Network with Attention
GRU	Gated Recurrent Unit
HEN	Heterogeneous Information Network
HENBUR	HEterogeneous information Network to support the BUg report Resolution
rVSM	revised Vector Space Model
SEVERIS	SEVERity ISsue assessment
SG	Stacked Generalization
SO	Stack Overflow
VSM	Vector Space Model
WEmb	Word Embeddings

CONTENTS

CHAPTER

1

INTRODUCTION

Many different artifacts (e.g., bug reports, source code files) are created during software development and maintenance of one software project. Program failures or defects are recorded on bug reports by different stakeholders (e.g., end-users, developers, or testers) (ZHU *et al.*, 2021).

The typical steps to handle bugs in software projects are as follow: i) bugs are logged in a bug report management tool (e.g., Jira); ii) someone with the right competence analyzes the logged bugs and define their severity levels (bug report severity prediction task) (LAMKANFI *et al.*, 2010); iii) the bugs are prioritized, and developers are assigned to fix the prioritized bugs (fixer recommendation task) (ČUBRANIĆ, 2004); iv) to fix a bug, it is necessary to identify the source code files that are most likely related to the logged bugs (bug localization task) (LAM *et al.*, 2017). These tasks belong to the bug report resolution (BRR) process.

According to Xia *et al.* (2015), the cost of manual fault correction represents the software maintenance process's highest cost. For example, the Eclipse project received more than 547,200 bug reports until May 14th, 2019 (WANG *et al.*, 2020). Moreover, many methodologies, techniques, and tools have been defined and proposed to minimize fault correction costs.

Combined with the development of tools and definition of automation strategies for BRR activities (e.g., bug localization), the previous studies investigate the applicability of artificial intelligence techniques in this context. For example, using machine learning algorithms to define predictive models to support bug report resolution activities.

Machine learning algorithms learn from data examples (e.g., bug reports) with a specifically structured data representation (e.g., Bag-of-Words). Many previous studies used this representation to support BRR tasks. However, Bag-of-Words has high dimensionality and high sparsity because of the high number of words on textual documents. Then, in general, this context impact negatively machine learning algorithms performance (ROSSI, 2015; YU; LIU, 2004).

The bug report resolution activities are interconnected, but most approaches handle those

activities in an isolated and disconnected way. In this perspective, software managers do not have a global view of BRR activities. Then in an isolated and disconnected way, they could have difficulty in decision making in planning maintenance activities. Moreover, automatic BRR activities would be much increase productivity (ZHOU *et al.*, 2016).

Zhang *et al.* (2016a) proposed a framework of bug report severity prediction and semi-automatic fixer recommendation. It handles these two tasks in an interconnected way. Also, it used Bag-of-Words to represent term occurrence from bug reports (ZHANG *et al.*, 2016a). This doctoral thesis investigates a heterogeneous information network's impact on representing data from different sources.

Xiao *et al.* (2020) defined a novel deep neural network (named $HIND_{BR}$) that used a heterogeneous information network (HEN) to detect similar duplicate bug reports. The experimental results suggest that the $HIND_{BR}$ method is better than the deep learning-based approach (XIAO *et al.*, 2020). This thesis investigates unified HEN representation to support three BRR activities (bug report severity prediction, fixer recommendation, and bug localization) holistically.

Overall, in the context of automation of bug report resolution tasks, there are some associated challenges to obtaining satisfactory predictive models:

- How to represent bug reports and other software artifacts to obtain an effective predictive model for specific tasks?
- How to define a unified representation that can be reused to support a set of tasks?

1.1 Research Question and Objectives

Motivated by the above challenges for the automation bug report resolution process, this doctoral thesis intends to answer the following research question:

How should software engineers represent software information to support the automatic bug report resolution process from a holistic multimodal perspective?

In this thesis, the holistic multimodal perspective refers to global information representation to support the automatic bug report resolution process. Also, it is a unified representation that uses software information from different software data sources (e.g., bug reports and sources code).

Based on this research question, the main objective of this doctoral thesis is *to investigate methods to support the automatic bug report resolution process in the context of open software*. Therefore, this thesis defined the following specific objectives to answer the research question and also achieve the main objective:

(i) evaluate the performance of semi-supervised network-based algorithms over the supervised algorithms to bug report severity prediction;

(ii) make a comparison between Bag-of-Words (BoW) representation and Heterogeneous Information Network (HEN) representation to support bug localization; and

(iii) investigate the use of heterogeneous information networks to support the automatic bug report resolution process in a holistic multimodal perspective.

1.2 Contributions

From the objectives of this doctoral thesis and the development of the proposals, we have the following scientific contributions:

- Proposal for a new use of semi-supervised algorithms based on heterogeneous information network to support the bug report severity prediction.
- Definition of a data representation based on a heterogeneous information network to support the bug localization activity.
- Extending, standardizing, pre-processing, and making available a set of collections of open source bug reports with the corresponding software metrics from impacted source code files after the resolution of the bug reports.
- Definition of a unified data representation based on a heterogeneous information network to support the bug report resolution process (bug report severity prediction, fixer recommendation, and bug localization).
- Dissemination of scientific results directly related to this doctoral thesis:
  - Published Article: BARBOSA, J. R.; MARCACINI, R. M.; BRITTO, R.; SOARES, F.; REZENDE, S. O.; VINCENZI, A. M. R.; DELAMARO, M. E. . BULNER: BUg Localization with word embeddings and NEtwork Regularization. In: VII Workshop on Software Visualization, Evolution and Maintenance (VEM), 2019, Salvador - BA. Anais do VII Workshop on Software Visualization, Evolution, and Maintenance (VEM). Porto Alegre - RS: SBC, 2019. p. 21-28.
  - Published Article: BARBOSA, J. R.; MATSUNO, I. P. ; GUIMARAES, E. H. R. ; REZENDE, S. O. ; VINCENZI, A. M. R. ; DELAMARO, M. E. . Mineração de Textos para Apoiar a Predição de Severidade de Relatórios de Incidentes: um Estudo de Viabilidade. In: XVI Simpósio Brasileiro de Qualidade de Software, 2017, Rio de Janeiro. Anais do XVI Simpósio Brasileiro de Qualidade de Software. Porto Alegre-RS: SBC, 2017. p. 89-103.

  - Article submitted: BARBOSA, J. R.; MATSUNO, I. P.; MARCACINI, R. M.; BRITTO, R.; REZENDE, S. O.; VINCENZI, A. M. R.; DELAMARO, M. E. . Improving predicting the severity of bug reports with semi-supervised learning based on the heterogeneous networks. In: Journal of Software Engineering Research and Development, 2021.
  - Article submitted: BARBOSA, J. R.; MARCACINI, R. M.; BRITTO, R.; REZENDE, S. O.; VINCENZI, A. M. R.; DELAMARO, M. E. . A framework to support the bug report resolution process with heterogeneous information network. In: Journal of Automated Software Engineering, 2021.

- Dissemination of scientific results related to other research areas:

  - Chapter of published book: BARBOSA, J. R.; VINCENZI, A. M. R. . Ferramentas de Gerenciamento da Qualidade de Software. In: José Carlos Maldonado, Márcio Eduardo Delamaro, Auri Marcelo Rizzo Vincenzi. (Org.). Automatização de teste de software com ferramentas de software livre. 1ed. Rio de Janeiro, RJ: Elsevier, 2018, p. 205-224.
  - Chapter of published book: VINCENZI, A. M. R.; BARBOSA, J. R.; FREITAS, E. N. A. . Ferramentas de Execução Automática de Casos de Teste. In: José Carlos Maldonado, Márcio Eduardo Delamaro, Auri Marcelo Rizzo Vincenzi. (Org.). Automatização de teste de software com ferramentas de software livre. 1ed. Rio de Janeiro, RJ: Elsevier, 2018, p. 21-58.
  - Published Article: BARBOSA, J. R.; VALLE, P.; MALDONADO, J. C.; VINCENZI, A. M. R.; DELAMARO, M. E. . An Experimental Evaluation of Peer Testing in the Context of the Teaching of Software Testing. In: XIX International Symposium on Computers in Education (SIIE 2017), 2017, Lisboa- Portugal. International Symposium on Computers in Education, 2017.

1.3 Thesis Outline

Chapter 2 presents the main concepts related to this doctoral thesis.

Chapter 3 addresses a quasi-experiment to evaluate semi-supervised learning methods to support bug report severity prediction. The results are reported in Barbosa *et al.* (2017). Also, this chapter evaluates how term weighting (binary or frequency) in text representation impacts performance in bug report severity prediction.

Chapter 4 presents a quasi-experiment to evaluate the impact of model representation combination (Bag-of-Words and Word Embeddings) to support bug localization, named BUg Localization with word embeddings and NEtwork Regularization (BULNER). The results are reported in Barbosa *et al.* (2019).

Chapter 5 describes a quasi-experiment that analyzes HEterogeneous information Network to support the BUg report Resolution (HENBUR) framework. The results have been submitted to publication in a scientific journal.

Finally, Chapter 6 provides contributions, conclusions, and limitations of the doctoral thesis. Also, it suggests some future work and provides data and source code availability.

CHAPTER

2

BACKGROUND

In this chapter are presented concepts and works related to this doctoral thesis. Section 2.1 presents machine learning concepts and methods used in this thesis to automate bug report resolution (BRR) tasks. Section 2.2 presents some data model representation options to support automatic BRR tasks. Section 2.3 presents an overview of the bug report and main bug report attributes. Finally, Section 2.4 presents three tasks (bug report severity prediction, fixer recommendation, and bug localization) from the BRR process that are researched in the following chapters of this doctoral thesis.

2.1 Machine Learning

Machine learning (ML) methods can be divided into supervised, unsupervised, and semi-supervised (LANTZ, 2013). The main difference among them is the set of examples for training. In supervised learning, the examples of the training set are labeled. That is, they are classified beforehand. New occurrences will be classified based on what was learned in the training set. There is no predefined label for the training set examples in unsupervised learning. The advantage of this type of learning is that it does not depend on labeled information, but the errors are more significant than the first method. In general, in supervised learning, accuracy is higher, but there are scenarios where it is challenging to have a sufficient number of labeled examples to generate a proper classification model (YUGOSHI, 2018).

Semi-supervised learning also considers a set of labeled examples. However, the number of examples in which the labels are known is much lower, and in this type of learning, specific methods are necessary to address this scenario. The goal of semi-supervised learning is to make use of unlabeled examples to improve classification performance. The way unlabeled examples are treated in semi-supervised learning may result in a better classification than supervised learning, considering the same number of labeled examples, or equivalent performance, considering a smaller number of labeled examples (ZHU; GOLDBERG, 2009; CHAPELLE; SCHLKOPF;

ZIEN, 2006).

2.1.1 Supervised Learning

Supervised learning methods can be divided into the following approaches: probabilistic, statistical learning, decision trees, and distance. Next, a few methods are mentioned.

- ***Naïve Bayes (NB)*** (RISH, 2001): this method is a probabilistic classifier. It is based on Bayes' theorem (KOCH, 1990) to identify the class in which example x has the highest probability of being associated, as given by Equation 2.1.

$$y = arg\,max_i P(y_i|x) \tag{2.1}$$

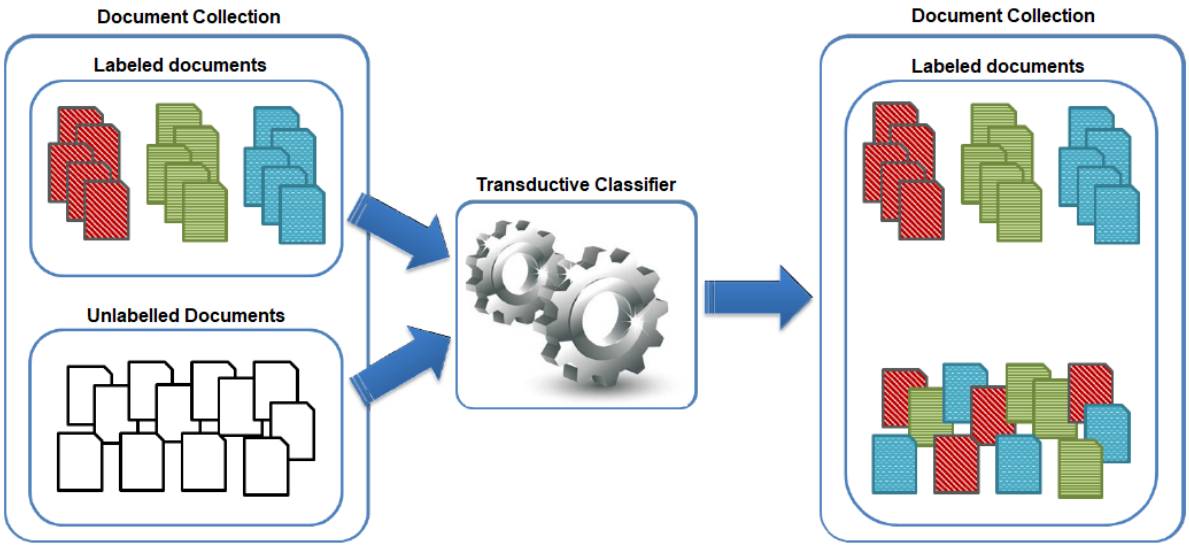
  For each term x, the probability of a given category y_i is calculated. This probability $P(y_i|x)$ is calculated from the occurrences of the term x in training documents where categories are already known. When all these probabilities are calculated, a new document can be classified according to the sum of the probabilities for each category of each term occurring within the document. $arg\,max_i$ returns the class most likely to be associated with the term x. The presence or absence of a term in a textual document can determine the prediction of the category.
- ***Multinomial Naïve Bayes (MNB)***: this method is a probabilistic classifier. It is based on the previous method, but the category is determined by the presence or absence of a term in the document and the number of occurrences of the terms in the document (KIBRIYA *et al.*, 2005).
- ***Support Vector Machines (SVM)***: this method is based on statistical learning developed by Vapnik (1995) that establishes a series of principles that must be followed in obtaining classifiers with good generalization capacity. The result of this classifier is hyperplanes between attribute vectors that divide space into several categories.
- ***J48***: C.45 algorithm (QUINLAN, 1993), this method is based on decision trees that use the *divide-and-conquer* strategy to recursively segment the search space into subspaces. Each subspace is adjusted using different models. Decision trees are simple to understand and interpret. However, they may create biased trees if some classes dominate other classes.
- ***k-nearest neighbor (kNN)***: this method is based on distances among objects. The classification of a new object is done considering the examples of the training set closest to it. The variation in this algorithm is the number of neighbors K to be considered.

2.1.2 Semi-Supervised Learning

Semi-supervised learning algorithms are divided into two types: (i) transductive and (ii) inductive. Transductive learning algorithms (see Figure 1) examine the entire set of unla-

beled examples, then classify the examples directly without the need to induce a classification model (ROSSI, 2015; CHAPELLE; SCHLKOPF; ZIEN, 2006).

Figure 1 – Automatic document classification through semi-supervised transductive learning.



Source: Adapted from Rossi (2015).

On the other hand, semi-supervised inductive learning algorithms use labeled and unlabeled examples to induce a classification model (see Figure 2) in two phases: (i) definition of the labels of unlabeled training documents through of transductive learning algorithm and (ii) extraction of a classification model from combining previously labeled documents with documents labeled in the first phase (ROSSI, 2015).

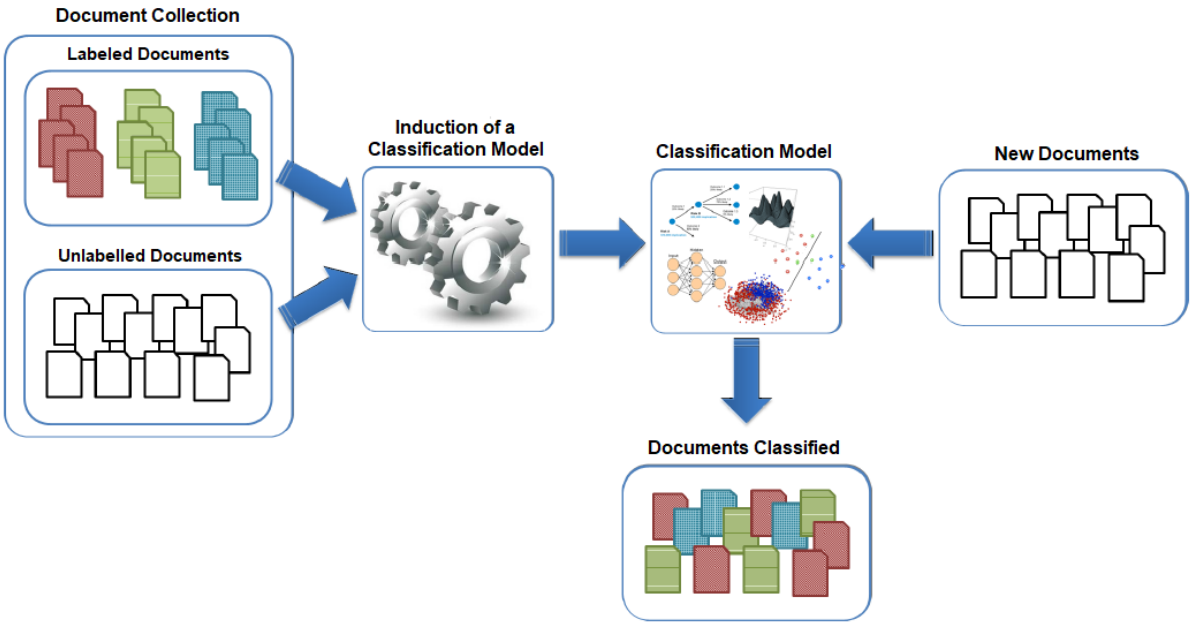
This thesis investigates the contribution of semi-supervised transductive learning algorithms based on networks (GNetMine, Label Propagation based on Bipartite Heterogeneous Network (LPBHN), Tag-based Model (TM), Transductive Classification based on Bipartite Heterogeneous Network (TCBHN)) and of a semi-supervised transductive learning algorithm based on the Vector Space Model (Expectation Maximization (EM)) to support the bug report severity prediction activity.

2.1.2.1 Network-based transductive learning

Considering that for a given problem, a network that represents the dataset (through relationships between objects) and that few objects are labeled. Regularization is an alternative to solve this problem. Regularization is a type of strategy used to implement semi-supervised transductive classification. (ROSSI, 2015).

The semi-supervised learning methods based on graph regularization seek to regularize the network considering two assumptions: (i) two objects connected in the network tend to be classified with the same label, and (ii) the object labels should be close to the real class information (training set). This section presents the algorithms to perform graph regularization

Figure 2 – Automatic document classification through semi-supervised inductive learning.



Source: Adapted from Rossi (2015).

on bipartite networks used in this thesis, with their respective graph regularization functions. In each regularization function, the first term is related to the first assumption and, analogously, the second term describes the second assumption (ROSSI, 2015; YUGOSHI, 2018).

A regularization framework can model these two assumptions (ZHU, 2005). The following cost function generically expresses this framework (DELALLEAU; BENGIO; ROUX, 2005):

$$Q(\mathbf{F}) = \frac{1}{2} \sum_{o_i, o_j \in \mathcal{O}} w_{o_i,o_j} \Omega(\mathbf{f}_{o_i}, \mathbf{f}_{o_j}) + \mu \sum_{o_i \in \mathcal{O}^L} \Omega'(\mathbf{f}_{o_i}, \mathbf{y}_{o_i}) \tag{2.2}$$

where o_i and o_j are examples of objects existing in the network that represent the problem domain. On the other hand, $\Omega(.)$ and $\Omega'(.)$ are functions of distances or similarities. For each pair of related objects in the network, $\Omega(.)$ calculates the proximity between the vectors of class information ($\mathbf{f}_{o_i}$, $\mathbf{f}_{o_j}$). While $\Omega'(.)$ calculates the proximity between the object class information and the respective real class information ($\mathbf{y}_{o_i}$). Finally, w_{o_i,o_j} defines the weight of the relation between the objects, and μ is the regularization parameter that defines the importance of the class information during the propagation of the labels (ROSSI, 2015; YUGOSHI, 2018).

Several variations of algorithms are based on regularization, differentiating concerning the parameter μ and the distance or similarity function. Below are some variations of algorithms:

- ***GNetMine*** (JI *et al.*, 2010): this is the extension of the algorithm *Learning with Local and Global Consistency* (LLGC) (ZHOU *et al.*, 2003). The regularization function to be

minimized by GNetMine is:

$$Q(\mathbf{F}) = \sum_{\mathcal{O}_i,\mathcal{O}_j \in \mathcal{O}} \lambda_{\mathcal{O}_i,\mathcal{O}_j} \sum_{o_k \in \mathcal{O}_i} \sum_{o_l \in \mathcal{O}_j} w_{o_k,o_l} \left|\left| \frac{\mathbf{f}_{o_k}(\mathcal{O}_i)}{\sqrt{\sum_{o_m \in \mathcal{O}_j} w_{o_k,o_m}}} - \frac{\mathbf{f}_{o_l}(\mathcal{O}_j)}{\sqrt{\sum_{o_m \in \mathcal{O}_i} w_{o_l,o_m}}} \right|\right|^2 + \sum_{o_j \in \mathcal{O}^L} \alpha_{o_j} (\mathbf{f}_{o_j} - \mathbf{y}_{o_j}) \tag{2.3}$$

in which $0 \leq \alpha_{o_j} \leq 1$ gives the importance of an object initially labeled by o_j in Equation 2.3.

- ***Label Propagation based on Bipartite Heterogeneous Network*** **(LPBHN)** (ROSSI; LOPES; REZENDE, 2014): this algorithm is an extension of the Gaussian Fields and Harmonic Function (GFHF) algorithm (ZHU; GHAHRAMANI; LAFFERTY, 2003) to bipartite heterogeneous networks. This is a parameter-free algorithm to perform semi-supervised learning on bipartite networks. The regularization function to be minimized by LPBHN is:

$$Q(\mathbf{F}) = \sum_{\mathcal{O}_j,\mathcal{O}_l \subset \mathcal{O}} \frac{1}{2} \sum_{o_i \in \mathcal{O}_j} \sum_{o_k \in \mathcal{O}_l} w_{o_i,o_k} (\mathbf{f}_{o_i} - \mathbf{f}_{o_k})^2 + \lim_{\mu \to \infty} \mu \sum_{o_i \in \mathcal{O}^L} (\mathbf{f}_{o_i} - \mathbf{y}_{o_i})^2 \tag{2.4}$$

There is a restriction that $\mathbf{f}_{o_i} = \mathbf{y}_{\mathbf{o_i}}$, so the second term of Equation 2.4 has a value tending to infinity.

- ***Tag-based Model*** **(TM)** (YIN *et al.*, 2009): this algorithm was initially proposed to classify web objects connected to social tags. In our context, the regularization function to be minimized by TM is:

$$Q(\mathbf{F}) = \left(\beta \sum_{o_i \in \mathcal{O}^L} ||\mathbf{f}_{o_i} - \mathbf{y}_{o_i}||^2 + \gamma \sum_{o_i \in \mathcal{O}^U} ||\mathbf{f}_{o_i} - \mathbf{y}_{o_i}||^2\right) + \left(\sum_{o_i \in \mathcal{O}_i} \sum_{o_j \in \mathcal{O}_j} w_{o_i,o_j} ||\mathbf{f}_{o_i} - \mathbf{f}_{o_j}||^2\right) \tag{2.5}$$

in which the parameters β and γ control the importance given to the term of Equation 2.5.

- ***Transductive Classification based on Bipartite Heterogeneous Network*** **(TCBHN)** (ROSSI; LOPES; REZENDE, 2016): this algorithm performs optimization and label propagation to minimize the following regularization function:

$$Q(\mathbf{F}) = \frac{1}{2} \left(\sum_{c_k \in \mathcal{C}} \left(\sum_{t_i \in \mathcal{T}^U} f_{t_i,c_k} - \sum_{b_j \in \mathcal{T}} w_{t_i,b_j} \cdot f_{b_j,c_k} \right) \right)^2 + \frac{1}{2} \left(\sum_{c_k \in \mathcal{C}} \left(\sum_{t_i \in \mathcal{T}^L} y_{t_i,c_k} - \sum_{b_j \in \mathcal{B}} w_{t_i,b_j} \cdot f_{b_j,c_k} \right) \right)^2 \tag{2.6}$$

2.1.2.2 Transductive learning based on the Vector Space Model

Another algorithm considered is ***Expectation Maximization (EM)*** for textual data classification presented in Nigam *et al.* (2000). In this algorithm, it is necessary to define the parameter λ (weight of unlabeled examples during semi-supervised learning) and the number of components for each class.

2.2 Data Model Representation

Machine learning algorithms learn from data examples with a specifically structured data representation. Consequently, there are different data model representations. This section presents basic concepts essential for understanding two types of data model representation. Firstly, it summarizes the characteristics of Bag-of-Words based on the Vector space model. Finally, it introduces the Heterogeneous Information Network representation. Both data model representations are used in this doctoral thesis to support three BRR tasks: bug report severity prediction, fixer recommendation, and bug localization.

2.2.1 Bag-of-Words Based on Vector Space Model

Bag-of-words (BoW) representation uses terms (e.g., keywords) extracted from texts as features in a vector space model. BoW is a document-term matrix (see Figure 3), where each row represents a document, each column (predictive attributes) represents a term (word) present in the document collection. The last column (target attribute) defines the document class when an document example has a label. The machine learning algorithms use predictive attributes to predict target attribute (MOREIRA; CARVALHO; HORVATH, 2018).

Each cell contains each term's weight in documents that could be one of two measures: (i) zero or one that indicates the presence or absence of the word in the document and (ii) frequency that contains the number of times of the word in the respective document. An example is presented in Figure 3.

Figure 3 – Structured text representations with Bag-of-words.

	$Term_1$	$Term_2$	...	$Term_{n-2}$	$Term_{n-1}$	$Term_n$	Class
Doc_1	1	1	...	0	0	0	C_1
Doc_2	0	0	...	1	1	0	C_2
Doc_3	0	1	...	1	0	0	...
...							...
Doc_m	1	1	...	0	1	0	C_k

Source: Adapted from Yugoshi (2018).

BoW ignores the order in which the terms appear in the documents, thereby losing the documents' semantic features. Furthermore, in BoW, only the weight of each term in the

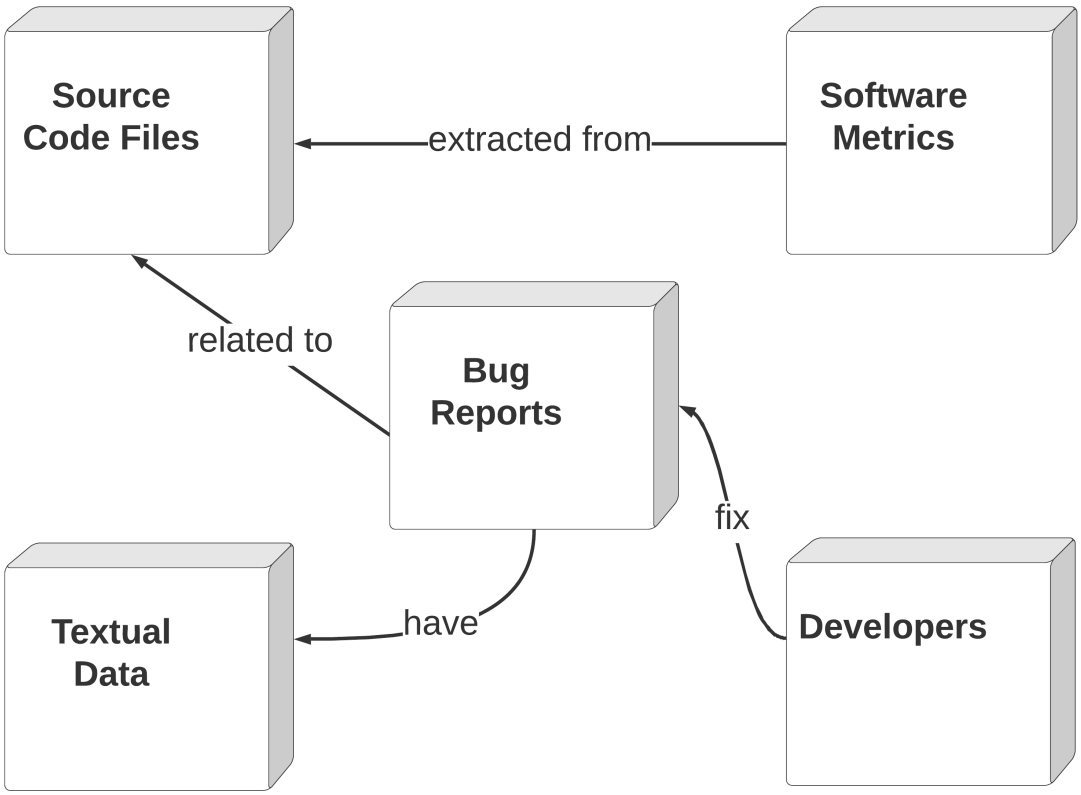
document is registered, i.e., the independence of the terms is lost (MANNING; RAGHAVAN; SCHüTZE, 2008).

BoW representation has as main characteristics of high dimensionality and sparsity. These characteristics affect the performance of machine learning algorithms negatively. Furthermore, machine learning algorithms are not able to infer relations between one term and other (ROSSI; LOPES; REZENDE, 2016).

2.2.2 Heterogeneous Information Network

Many problems can naturally be represented as a network. For example, a network can represent a bug report resolution process (see Figure 4). In this figure, the network has different node sets (e.g., bug reports) that represent distinct objects, and the edges represent the relationships between the objects. This network with different object types is called a heterogeneous information network (HEN).

Figure 4 – Heterogeneous information network.



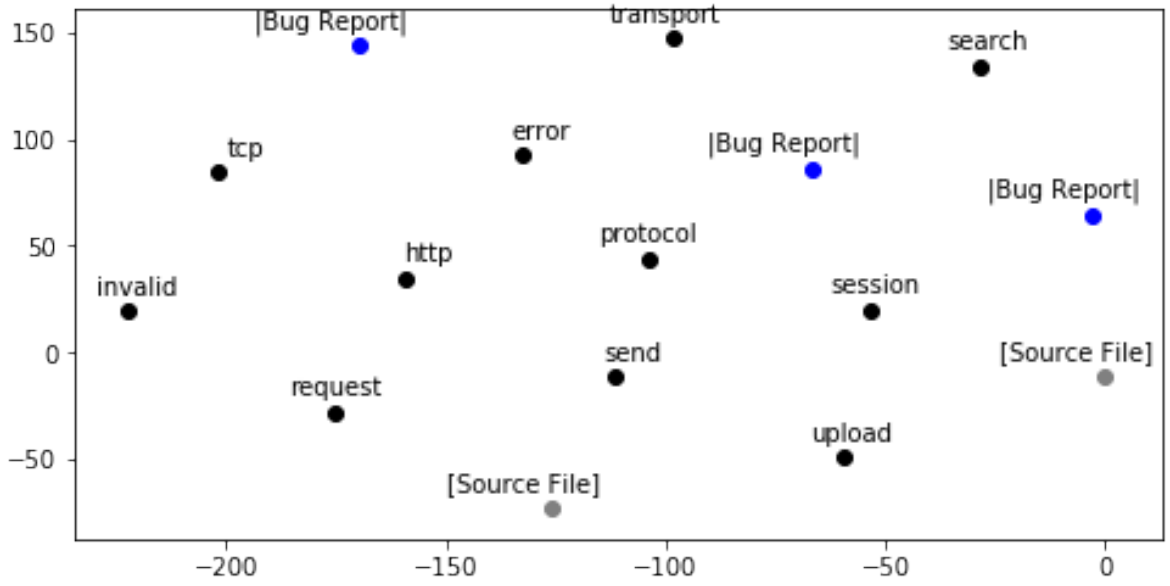
Source: Elaborated by the author.

Depending on the network's complexity, it is challenging to represent the relationships between different objects through nodes (Cui *et al.*, 2019). Network embedding is an alternative

to solve this challenge. It creates a low-dimensional feature representation for nodes. It preserves the network structure and side information from the original network (CHANG *et al.*, 2015; Cui *et al.*, 2019). Then, network embedding is a way to learn from HEN.

All nodes and edges from the original network are encoded into its embedding vector in the network embedding space created. From these vectors, it is possible to calculate the similarity between the network's different nodes and perform other operations in a simplified way (Cui *et al.*, 2019).

Figure 5 – Projection of a sample of network nodes from the embedding network model.



Source: Elaborated by the author.

Figure 5 illustrates the projection of some heterogeneous information network of the Tomcat software project in two-dimensional space using the t-SNE method (MAATEN; HINTON, 2008). Data points indicate different nodes in the network, such as terms, bug reports, and source code files. As discussed in this section, the distances between objects in vector space can be explored for different tasks related to mining software repositories. Note that it is possible to visually identify the proximity of bug reports, terms, and source files. Analogously, it is possible to identify similar or even duplicate bug reports. Thus, this new dimensional space can even be used for training other machine learning methods to support the automatic bug report resolution process.

2.3 Bug Report

According to the IEEE Standard 29119-1-2013, an Incident Report is the "documentation of the occurrence, nature, and status of an incident" (ISO/IEC/IEEE, 2013). This doctoral thesis

uses the term “bug report” to refer to an “incident report” since it is especially interested in incidents that correspond to software faults, popularly *bugs*.

Figure 6 – An example of Eclipse bug report 4023.

Source: Adapted from Eclipse Foundation (2021).

A Bug Tracking System (BTS) is a tool that supports the management of bug reports. Software faults are registered as bug reports using this type of tool (e.g., Bugzilla[1]). Figure 6 shows a screenshot of the bug report *Bug 4023* from the Bugzilla of the Eclipse [2]. The type of registered information depends on the used BTS. For example, to register a bug report using Bugzilla, one needs to provide the following information:

- **Summary** - the title definition (see information *1* in Figure 6).
- **Product** - the product in which it is originated the bug report (see information *2* in Figure 6).

[1] Bugzilla: <https://www.bugzilla.org/>
[2] Eclipse: <https://www.eclipse.org/>

- **Component** - the component related to the occurrence (see information *3* in Figure 6).
- **Priority** - the information defines the priority of correction of a fault concerning the others, where P1 is considered the highest priority and P5 is the lowest (see information *4* in Figure 6).
- **Severity** - the information describes the impact of the fault (see information *5* in Figure 6). Table 1 shows possible values. The blocker type is the most severe, and the trivial type is the least severe. Blocker bug reports are interdependent with other bug reports. Because of this, it is a complex process of solution of other bug reports (XIA *et al.*, 2015).
- **Description** - the detailed description of the fault (see information *6* in Figure 6).

Table 1 – Bug report severity type.

Severity	Description
blocker (BL)	Blocks the development and activity of software testing
critical (CR)	Causes data loss, crashes, or memory impairment
major (MA)	Increased loss of functionality
normal (NO)	Causes loss of functionality under specific situations
minor (MI)	It results in less loss of functionality
trivial (TR)	Elementary problems, such as misspelling

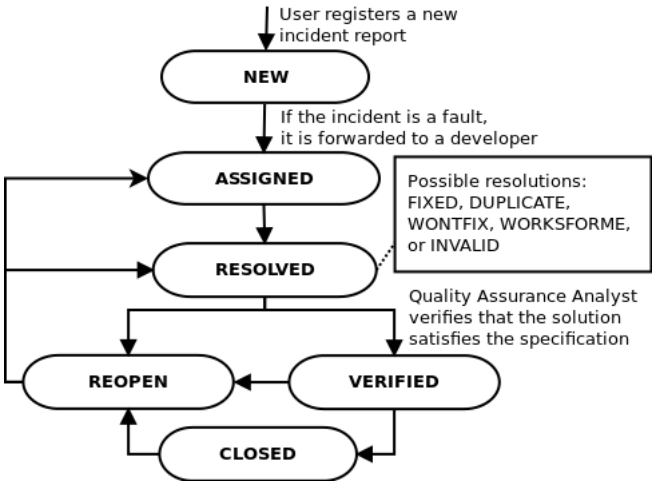
Source: Adapted from Saha *et al.* (2015).

Figure 7 presents different states of a bug report in Bugzilla projects. In this cycle, the bug report's severity level is manually verified by a triager. After registering a new bug report in the BTS, its state is defined as **NEW**. Then, the triager assigns the bug report to a developer, and the bug report's state is changed to **ASSIGNED**. When the assignee finishes the bug fixing task, the bug report's state is changed to **RESOLVED**. Finally, if the bug report is fixed entirety, its state is changed to **CLOSED**. Otherwise, its state is changed to **REOPEN** (ZHANG *et al.*, 2016a).

2.4 Bug Report Resolution Process

Bug reports are important software artifacts used to support various bug report resolution (BRR) activities (ZHANG *et al.*, 2016b). According to Zhang *et al.* (2016b), in the BRR manual process, a new bug report generally needs to go through three phases to be fixed: bug understanding, bug triage, and bug fixing. Figure 8 shows the interrelationship between these phases and their respective tasks. First, in the bug understanding phase, the software test manager needs to understand the new bug report for the classifier by severity level; second, the software test manager needs to identify an appropriate developer to fix the bug report in the bug triage phase. Finally, a developer needs to locate the bug in the source code, and for that, it is sometimes necessary to analyze old bug reports.

Figure 7 – Bug report life cycle in Bugzilla.



Source: Adapted from Zhang *et al.* (2016a).

This section presents some previous studies related to the three main tasks in the bug report resolution process: bug report severity prediction, fixer recommendation, and bug localization. This doctoral thesis presents solutions based on machine learning methods to problems related to these three tasks. The following subsections present previous work related to these three tasks.

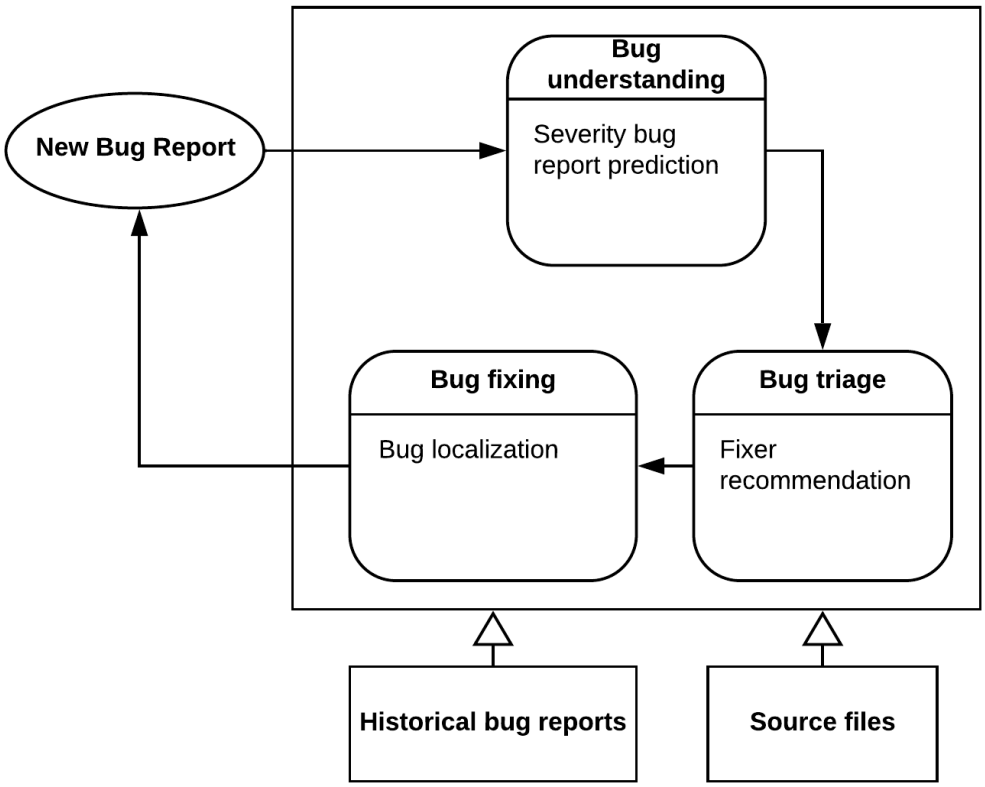
2.4.1 Bug Report Severity Prediction

The automatic classification of a bug report concerning the severity is also known as bug report severity prediction. There are two types of severity prediction models: binary (e.g., severe or non-severe) (LAMKANFI *et al.*, 2010) and non-binary (e.g., blocker, critical, major, minor, or trivial) (TIAN; LO; SUN, 2012).

Considering that there are different types of bug report severity, Tian, Lo and Sun (2012) proposed a non-binary severity prediction model that considers the five major severity types. Tian, Lo and Sun (2012) applied *BM25* document similarity function (ROBERTSON; ZARAGOZA; TAYLOR, 2004; ZARAGOZA *et al.*, 2004) and its extension ($BM25_{ext}$) (SUN *et al.*, 2011) combined with Vector Space Model (VSM) for bug report representation. They used k nearest neighbors algorithm to decide the severity type for a new bug report. The proposed approach presented better results than the SEVERity ISsue assessment (SEVERIS) method defined by Menzies and Marcus (2008).

Garcia and Shihab (2014) proposed a binary severity prediction model (blocking or non-blocking). They used 14 factors (e.g., description text, comment text, and reporter name) related to the textual bug report information to calculate a Bayesian score using a Naïve Bayes classifier. Then, the Bayesian score was used to classify bug reports in either blocking or non-blocking (GARCIA; SHIHAB, 2014).

Figure 8 – Bug report resolution process.



Source: Adapted from Zhang *et al.* (2016b).

Zhou *et al.* (2016) proposed a non-binary classification approach that predicts three levels of severity: high, middle, and low. A combination of text mining techniques and data mining is applied to analyze bug report summaries in the study. The authors used a Multinomial Naïve Bayes classifier and outperformed another approach (Logistic Regression model) presented by Antoniol *et al.* (2008).

Ramay *et al.* (2019) proposed a binary severity prediction model based on a deep neural network classifier. First, they used Senti4SD (CALEFATO *et al.*, 2018) to compute an emotion score for each bug report. Second, they used word embedding to transform each word from the bug report into a fixed-dimensional vector. Finally, they used these vectors and emotion scores as input to a deep learning-based classifier (Convolutional Neural Network) (Ramay *et al.*, 2019). The results suggest that their study outperforms two machine learning algorithms (Multinomial Naïve Bayes (LAMKANFI *et al.*, 2011; KIBRIYA *et al.*, 2005) and Random Forest (BREIMAN, 2001)) and a deep learning algorithm (Long Short-Term Memory) (HOCHREITER; SCHMIDHUBER, 1997; GERS; SCHMIDHUBER; CUMMINS, 1999).

Tan *et al.* (2020) defined a method based on logistic regression to support a non-binary severity prediction model. They collect question-and-answer pairs from Stack Overflow (SO)

related to bug repositories. After, they use *BM25* document similarity function to enhance bug reports with SO posts. Finally, they train a model using logistic regression classifier to automated severity prediction. The results suggest that their study outperforms three algorithms: Naïve Bayes, *k*-nearest neighbor, and a deep learning algorithm Long Short-Term Memory (TAN *et al.*, 2020).

2.4.2 Fixer Recommendation

Typically, automatic fixer recommendation is defined as a classification problem. The objective is to identify developers qualified (bug fixers) to solve a given bug report (MANI; SANKARAN; ARALIKATTE, 2019).

Mani, Sankaran and Aralikatte (2019) developed a bug fixer recommendation approach that used Deep Bidirectional Recurrent Neural Network with Attention (DBRNN-A) (PHAM *et al.*, 2014) for bug report representation. In the study, for each word presented in the bug report vocabulary, a fixed-dimensional vector is learned using *word2vec* (MANI; SANKARAN; ARALIKATTE, 2019). The authors compared their approach with a baseline BoW-based approach. As a result, their approach outperformed the baseline.

Lee *et al.* (2017) proposed an approach (CNN Triager) that combines word embedding and a Convolutional Neural Network (CNN) to recommend bug fixers. The authors showed that their approach outperformed state-of-the-art classifiers (Support Vector Machine, Naive Bayes, C.48) in open source projects (LEE *et al.*, 2017). They also evaluated their approach in an industrial case. They used CNN Triager to integrate with the issue tracking system as an assistant for human triagers. As a result, CNN Triager recommends hints for human triagers to support bug triage activity.

Xi *et al.* (2018) defined a framework (SeqTriage) that uses Recurrent Neural Networks with neurons Gated Recurrent Unit (GRU) to recommender potential fixers. SeqTriage is a sequence to sequence-based model that explores from the first potential fixer until the last fixer related to each bug report (XI *et al.*, 2018). They compared SeqTriage with the following baselines: DBRNN-A (MANI; SANKARAN; ARALIKATTE, 2019), CNN Triager (LEE *et al.*, 2017), SVM+BoW (ANVIK; HIEW; MURPHY, 2006) and TopicMinerMTM (Xia *et al.*, 2017). SeqTriage improved the accuracy compared with the baselines chosen.

Jonsson *et al.* (2016) proposed a bug fixer recommendation approach that combines uses ensemble learner Stacked Generalization (SG). They used BoW to represent the input data for their approach. In addition, they combined the following machine learning techniques: Bayes Net, Naive Bayes, Support Vector Machines, *k*-nearest neighbor, and Decision Tree (JONSSON *et al.*, 2016). The authors evaluated their approach and showed that SG outperformed individual classifiers to recommend bug fixers in industrial applications.

Zaidi and Lee (2021) developed a fixer recommendation approach that used a graph

neural network (GNN) based on Yao, Mao and Luo (2019)'s work. Zaidi and Lee (2021) used a heterogeneous graph to represent the word-to-word relation and word-to-bug report relation. They evaluated their approach with Lee *et al.* (2017)'s and Mani, Sankaran and Aralikatte (2019)'s methods. Their approach outperformed them.

2.4.3 Bug Localization

To fix a fault reported in a bug report, it is necessary to identify where the bug is located in a software product. This activity is known as bug localization. A common way to automate this activity is through information retrieval (IR), where a bug report is treated as a query and the related sources code files are handled as documents (Saha *et al.*, 2013).

Ye, Bunescu and Liu (2014) proposed an approach that combines the IR model with the learning-to-rank (LR) technique to localize bugs. They used BoW to represent the words from bug reports (summary and description) and source files (comment and code). Also, they defined a ranking model that used features weighted combination from relationships between source code file and bug report (YE; BUNESCU; LIU, 2014). Finally, they compared the LR approach against two state-of-the-art systems: BugLocator (ZHOU; ZHANG; LO, 2012) and BugScout (NGUYEN *et al.*, 2011). Their approach significantly outperformed BugLocator and BugScout.

Wen, Wu and Cheung (2016) developed an approach (LOCUS) based on IR that locates bugs from software changes. LOCUS combines three different models (the natural language (NL), code entity names (CE), and Boosting) to compute a final similarity score between a change and a bug report (Wen; Wu; Cheung, 2016). LOCUS outperforms three state-of-the-art approaches: BRTracer (Wong *et al.*, 2014), BLUiR (Saha *et al.*, 2013), and AmaLgam (WANG; LO, 2014).

Lam *et al.* (2017) proposed an approach named DNNLoc that combines deep learning (DL) with a revised Vector Space Model (rVSM) to localize bugs automatically (LAM *et al.*, 2017). DNNLOC outperformed Naive Bayes (NB) (KIM *et al.*, 2013), LR (YE; BUNESCU; LIU, 2014), and BugLocator (ZHOU; ZHANG; LO, 2012).

Wang *et al.* (2020) defined a multi-dimension deep learning model, denoted by MD-CNN. It captures the complex and non-linear correlation of five features (text similarity, structural information, collaborative filtering, bug fixing history, and class name similarity) from relationships between source code files and bug reports (WANG *et al.*, 2020). MD-CNN outperformed LR (YE; BUNESCU; LIU, 2016), BugLocator (ZHOU; ZHANG; LO, 2012), Vector Space Model method (MANNING; RAGHAVAN; SCHüTZE, 2008), and Deep Neural Networks (HINTON; SALAKHUTDINOV, 2006).

Zhu *et al.* (2021) developed a new framework using a deep multimodal model for bug localization (DEMOB). Firstly, they mapped individual representation of the bug report and

source code files after coordinating each representation into a multimodal space. This framework narrows the gap between natural language (bug reports) and programming language (source code files). As a result, DEMOB outperformed BugLocator (ZHOU; ZHANG; LO, 2012), BLUiR (Saha *et al.*, 2013), DNNLoc (LAM *et al.*, 2017), NP-CNN (HUO; LI; ZHOU, 2016), and LS-CNN (HUO; LI, 2017).

2.5 Final Remarks

Regarding how data is represented to be used as input by prediction models, most related papers use BoW (YE; BUNESCU; LIU, 2014; LAMKANFI *et al.*, 2010; GARCIA; SHIHAB, 2014; ZHOU *et al.*, 2016). A few studies have used word embedding for data representation (LEE *et al.*, 2017; MANI; SANKARAN; ARALIKATTE, 2019; Ramay *et al.*, 2019). To the best of our knowledge, few studies have explored representing software documents by a heterogeneous information network in this BRR process context (bug report severity prediction, fixer recommendation, and bug localization).

In the context of bug report severity prediction and fixer recommendation, a few studies use different software artifacts to subsidize the prediction model. In general, it used a bug report history only to support the task automation process. A few studies also explored a holistic way of predicting these three activities of the BRR process (ZHANG *et al.*, 2016a).

This doctoral thesis fills the existing gaps by proposing: (i) an application of semi-supervised classifier and data representation based on two-way heterogeneous networks in severity prediction of bug reports (presented in Chapter 3); (ii) a BUg Localization with word embeddings and Network Regularization (presented in Chapter 4); and (iii) a general framework to support tasks (bug report severity prediction, fixer recommendation, and bug localization) in the bug report resolution process using a unified heterogeneous information network (presented in Chapter 5).

CHAPTER

3

IMPROVING PREDICTING THE SEVERITY OF BUG REPORTS WITH SEMI-SUPERVISED LEARNING AND HETEROGENEOUS NETWORKS

It is common to identify faults at all stages of the software development life cycle; 50-80% of the total cost of software maintenance is associated with the cost of fault correction (XIA *et al.*, 2015). Many software projects use Bug Tracking Systems (BTS) to support the management of these bug reports. Examples of BTS are Bugzilla (Mozilla Foundation, 2021), Jira (Atlassian, 2021), and Mantis (MantisBT Development Team, 2021).

In general, human beings record bug reports after identifying unexpected behavior of a software system. When registering a bug report, two types of information are essential and often define how fast the bug report should be fixed: priority and severity. A bug report's severity refers to the impact that a bug has on the successful execution of the software (LAMKANFI *et al.*, 2010). The priority attribute represents the degree of urgency with which a bug report must be fixed (LAMKANFI *et al.*, 2010).

Different people often conduct the analysis of faults subjectively, and the severity and priority of bug reports are often under or overestimated. This situation makes it challenging to prioritize the correction of bugs. Furthermore, due to many bug reports produced daily, there is an enormous waste of human resources that must be allocated to manually redefine/correct the priority and severity of bug reports.

To address the aforementioned issue, several automatic techniques have been proposed. These techniques include the assignment of bug report severity/priority (Ramay *et al.*, 2019), duplicate bug report detection (NEYSIANI; BABAMIR; ARITSUGI, 2020), and fault correction time prediction (XIA *et al.*, 2015).

In the context of bug report severity prediction, most studies use supervised machine learning (SML) (GOMES; TORRES; CôRTES, 2019). In general, SML has better performance with many labeled samples (e.g., severity bug reports defined). However, when one software project (e.g., a new software project) has few labeled instances, SML may perform worst than semi-supervised machine learning (ZHU; GOLDBERG, 2009; YUGOSHI, 2018).

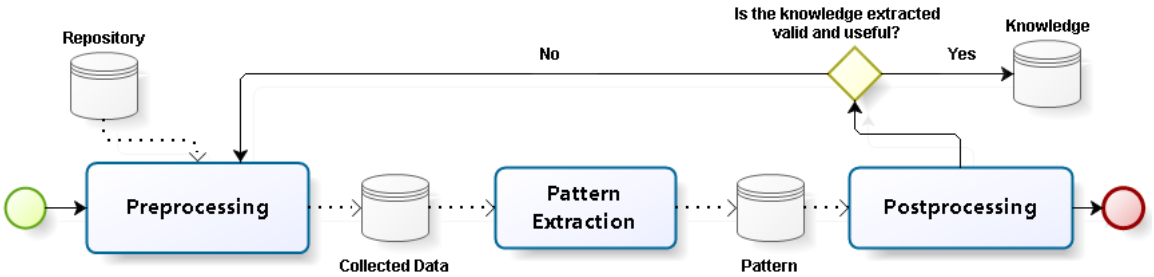
In the present chapter, an approach is proposed to enable the prediction of bug report severity from text mining techniques. This approach is based on text classification, which uses sample bug reports labeled in different types of severity to construct a classification model. As a result, the severity of new bug reports can automatically be classified (or suggested), dramatically reducing the human effort related to manual analysis. To evaluate this approach, an experiment was conducted to evaluate several classification methods and compare the main techniques to obtain the structured representation of texts. The experimental results are promising and potentially helpful in guiding the choice of classification methods and their parameters for new studies involving the prediction of severity in bug reports.

This chapter is organized as follows. Section 3.1 contains relevant concepts associated with bug report severity prediction. Section 3.2 presents the experimental design, while Section 3.3 presents the results. Section 3.4 discusses the results. Section 3.5 presents the threats to the validity of this study. Finally, Section 3.6 presents the conclusions and view on future work.

3.1 Mining Bug Tracking System Repository

Figure 9 presents the main activities and objects of the Mining Bug Tracking System Repository (MBTSR). The activities correspond to the rounded rectangles and objects to the cylinders.

Figure 9 – Mining BTS repository general flow.



Source: Elaborated by the author.

The pre-processing activity corresponds to the first phase of MBTSR, and one or more types of BTS repositories can be provided as input to enable data collection (bug reports) (JUNG; LEE; WU, 2012). After the data collection, pre-processing activities are conducted to transform the data into a suitable representation for the pattern extraction. For example, when pre-processing

bug reports, which is natural language data in text format, it is necessary to conduct the following steps:

- **Tokenization** - break the text into tokens. Numeric characters and punctuation marks are removed from the text.
- **Removal of stopwords** - preposition, adverbs, and other structures (defined as stop-words) commonly used to support the construction of sentences in the human language (a, from, to, up), generally do not aggregate information in the context of bug report mining algorithms (LAMKANFI *et al.*, 2011). Because of this, from a set of defined stop-words, all stop-words in the text being processed are removed.
- **Stemming** - reducing each word of the text to its radical correspondent (minimum and unambiguous denotation of the term). For example, the words "argue", "arguing" and "argus" can be reduced to "argu".
- **Definition of data representation** - an acceptable format of textual documents needs to be defined to enable the analysis. In most cases, a bag-of-words (TAN; STEINBACH; KUMAR, 2005) (representation in the vector space model) is constructed to meet this objective. Each row of the bag-of-words represents a document (bug report), and each column represents an attribute.

Machine learning (ML) methods must be used to extract bugs reports' patterns.. The process of assigning a predefined label to a data instance is known as automatic document classification, which is an important data mining sub-area (LAMKANFI *et al.*, 2011).

The representation of the document classification function is defined as follows:

$$f : Document \rightarrow \{r_1, \cdots, r_q\} \tag{3.1}$$

in the context of this chapter, the document corresponds to an example of a bug report and $\{r_1, ..., r_q\}$ to the predefined labels, that is, the type of severity. This process of classification of a bug report concerning the kind of severity is also known as bug report severity prediction.

Finally, the acquired knowledge is evaluated according to specific metrics in the post-processing activity. If the acquired knowledge is not suitable for use, the pre-processing activities will be resumed to improve the quality of the knowledge.

3.2 Experiment Design

Experiments were conducted to evaluate the application of pre-processing techniques (text representations) and classification methods on nine large-scale open source bug repositories

in the context of bug report severity prediction. The goal is to verify whether semi-supervised methods perform better than supervised ones. In addition, the experiment measures how term weighting influences performance in bug report severity prediction.

The overview of the experiment is presented in this section.

3.2.1 Definition of Research Question

In particular, this study seeks to answer the following research questions:

- **RQ1**: Do semi-supervised learning methods perform better to support bug report severity prediction than supervised learning methods?
- **RQ2**: Does term weighting (binary or frequency) in text representations impact performance in bug report severity prediction?

In addition, the experiment can be defined as follows (WOHLIN *et al.*, 2012):

- analyze *classification methods with different text representations*,
- for the purpose of *evaluating bug report severity prediction*,
- with respect to *model performance*,
- from the point of view of the *researcher*,
- in the context of *open source projects*.

3.2.2 Preparation and Planning

This section presents the experiment planning steps executed: sample selection, and the three activities presented in Figure 9 on Page 50 (pre-processing, pattern extraction, and post-processing) are instantiated in our experiments as presented in the following subsections.

3.2.2.1 Sample Selection

In this experiment, we crawled bug reports from the Bugzilla repository provided by Lamkanfi, Pérez and Demeyer (2013) and bug reports from the Apache Jira repository (Apache Software Foundation, 2021). Only the detailed descriptions of the bug reports were considered data to be processed. The information from the collections of texts used in the experiments are summarized and presented in Tables 2 and 3. The description of the software and the total number of documents of the datasets are presented. The number of documents per class type of each collection is also presented. This information is also important in assessing whether the number of labeled examples and/or unbalanced classes can impact

the classification performance. Pre-processed texts and other information are available at <http://sites.labic.icmc.usp.br/ipm/msr-sbrp>.

Table 2 – Descriptions of collections of texts from Bugzilla repository (Severity Type: BL = Blocker, CR = Critical, MA = Major, NO = Normal, MI = Minor, TR = Trivial).

Software	# Docs	Distribution of Classes by Severity Type					
		BL	*CR*	*MA*	*NO*	*MI*	*TR*
Eclipse-CDT	5640	78	166	490	4547	275	84
Eclipse-JDT	10814	94	274	1000	8306	781	359
Eclipse-PDE	5655	47	117	476	4693	208	114
Eclipse-Platform	24775	415	989	2718	18891	1088	674
Mozilla-Bugzilla	4616	275	176	506	2478	766	415
Mozilla-Thunderbird	19237	65	1894	2982	12429	1415	452

Source: Research data.

Table 3 – Descriptions of collections of texts from Apache Jira repository (Severity Type: BL = Blocker, CR = Critical, MA = Major, MI = Minor, TR = Trivial).

Software	# Docs	Distribution of Classes by Severity Type				
		BL	*CR*	*MA*	*MI*	*TR*
Apache-Abdera	180	7	8	124	34	7
Apache-Ambira	11592	798	2873	7633	245	43
Apache-Open-JPA	2440	72	144	1750	428	46

Source: Research data.

3.2.2.2 Pre-processing

In pre-processing, the following tasks are performed: document preparation, term extraction and attribute selection, and generation of a structured representation of the appropriate document collection for the pattern extraction methods. For the document preparation, the following tasks were carried out: standardization, removal of stopwords, stemming, and selection of unigrams as terms. To answer the second research question, two measures were used to evaluate the weight of the terms: (i) **binary** which considers only if the term is present or not in each document (ii) **frequency** which considers the number of times that the term is present in each document. Only terms with frequencies greater than two were considered.

Regarding the representation of the text collection, two representations were used: (i) vector space model, the bag-of-words (BoW) (TAN; STEINBACH; KUMAR, 2005) and (ii) bipartite heterogeneous network (ROSSI; LOPES; REZENDE, 2016). Bag-of-words is a document-term matrix, where each row represents a document, each column represents a term (word) present in the document collection, and each cell contains a measure. In this work, two measures: (i) zero or one, that indicates presence or absence of the word in the document, and (ii)

frequency that contains the number of times of the word in the respective document. An example is presented in Figure 10 (a).

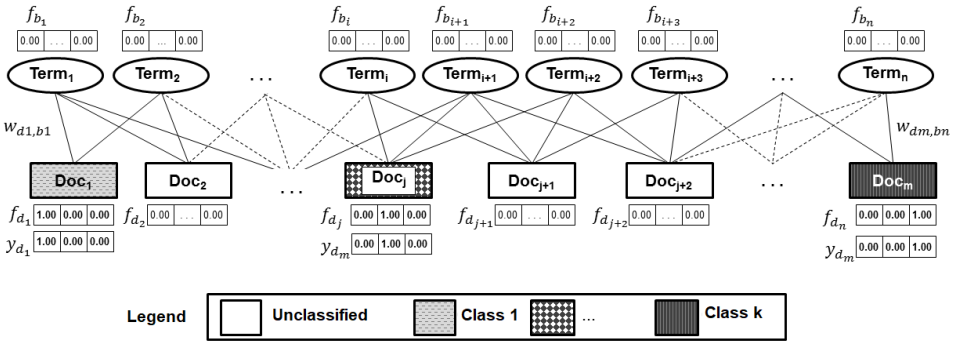
Formally, a bipartite heterogeneous network can be defined as $N = (\mathcal{O}, \mathcal{E}, \mathcal{W})$, in which $\mathcal{O}$ represents two sets of network objects (also called vertices or nodes), $\mathcal{E}$ represents the set of connections (also called relations or links) which occurs just from objects of one set to another set, and $\mathcal{W}$ represents the weights of the connections. In this study, $\mathcal{O}$ is composed of the sets of documents $\mathcal{D}$ and set of terms $\mathcal{T}$. $\mathcal{E}$ is composed of e_{ij} connections that represent the presence of the term t_j in the d_i, $0 < i \leq |\mathcal{D}|$ and $0 < j \leq |\mathcal{T}|$, and the weights of the connections are binary or frequency.

Figure 10 (b) presents an example of the representation using a bipartite heterogeneous network model. Each description of the bug reports extracted from the BTS repositories is considered a document in this study. Each word is considered a term. The weight is considered binary (zero or one) or the frequency of the terms.

Figure 10 – Structured text representations.

	$Term_1$	$Term_2$	...	$Term_{n-2}$	$Term_{n-1}$	$Term_n$	Class
Doc_1	1	1	...	0	0	0	C_1
Doc_2	0	0	...	1	1	0	C_2
Doc_3	0	1	...	1	0	0	...
...							...
Doc_m	1	1	...	0	1	0	C_k

(a) *Bag-of-words* (*BoW*)



(b) Bipartite heterogeneous network

Source: Adapted from Rossi (2015), Yugoshi (2018).

The text classification algorithms based on BoW ignore the dependence among the documents or terms (ROSSI; LOPES; REZENDE, 2016). Thus, in general, these algorithms present worst performance than algorithms based on networks that use label propagation to disseminate labels from labeled objects to other objects by network connections (ROSSI; LOPES; REZENDE, 2016; ROSSI; LOPES; REZENDE, 2014; ZHOU *et al.*, 2003).

3.2.2.3 Pattern Extraction

In this activity, the main supervised and semi-supervised learning methods were used. Regarding the supervised learning methods, the following approaches were used: probabilistic (NB and MNB), statistical learning (SVM), decision trees (J48), and distance (*k*NN). Regarding the semi-supervised learning methods, the following algorithms were used: GNetMine, LPBHN, TCBHN, TM, and EM.

3.2.2.4 Post-processing

In this activity, an experimental evaluation is carried out, in which the feasibility and impact of the use of semi-supervised learning in the severity prediction of bug report and the impact of the weight of the terms in the text representation are analyzed.

To compare the results of the classification, the measure F^1 representing the harmonic mean of the *Precision* and *Recall* measurements were used, where both measures have the same weight (see Equation 3.2).

$$F^1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.2}$$

Precision and *Recall* were calculated for each class in multi-class assessment. The formula for computing the *Precision* and *Recall* of a class c_i are given in Equations 3.3 and 3.4, respectively:

$$Precision_{c_i} = \frac{TP_{c_i}}{TP_{c_i} + FP_{c_i}} \tag{3.3}$$

$$Recall_{c_i} = \frac{TP_{c_i}}{TP_{c_i} + FN_{c_i}} \tag{3.4}$$

where TP (True Positive) means the number of test documents correctly assigned to class c_i, FP (False Positive) means the number of test documents from class c_j ($c_j \neq c_i$) but assigned to class c_i, and FN (False Negative) is the number of test documents from class c_i but assigned to class c_j ($c_j \neq c_i$).

The *Precision* measure returns the percentage of documents correctly classified as c_i considering all documents classified as c_i. The *Recall* measure returns the percentage of documents correctly classified as c_i considering all documents which actually belong to class c_i.

Two strategies to summarize the results of precision and recall computed for each class of a text collection are: (i) ***micro-averaging*** and ***macro-averaging*** (SOKOLOVA; LAPALME, 2009). The ***micro-averaging*** strategy performs a sum of the terms of the evaluation measures.

Therefore, the *precision* and *recall* using the micro-averaging strategy are defined by Equations 3.5 and 3.6, respectively.

$$Precision^{Micro} = \frac{\sum_{c_i \in \mathcal{C}} TP_{c_i}}{\sum_{c_i \in \mathcal{C}} (TP_{c_i} + FP_{c_i})}, \tag{3.5}$$

$$Recall^{Micro} = \frac{\sum_{c_i \in \mathcal{C}} TP_{c_i}}{\sum_{c_i \in \mathcal{C}} (TP_{c_i} + FN_{c_i})}. \tag{3.6}$$

The ***macro-averaging*** strategy performs an average over the evaluations measures for each class. Therefore, the *precision* and *recall* using macro-averaging strategy are:

$$Precision^{Macro} = \frac{\sum_{c_i \in \mathcal{C}} Precision_{c_i}}{|\mathcal{C}|}, \tag{3.7}$$

$$Recall^{Macro} = \frac{\sum_{c_i \in \mathcal{C}} Recall_{c_i}}{|\mathcal{C}|}. \tag{3.8}$$

Micro-averaging scores are dominated by the number of TP. Therefore, large classes dominate small classes in micro-averaging scores. On the other hand, macro-averaging gives equal weight to each class. In this case, the number of TP in small classes is emphasized in macro-averaging scores. These two strategies present different scores and are complementary to each other. F^1 computed through micro-averaging of precision and recall is denoted by *Micro-*F^1, and through macro-averaging by *Macro-*F^1.

Firstly carried out a 10-fold cross-validation process to obtain *Micro-*F^1 and *Macro-*F^1. For each training set (9 folds), ten runs were carried out to induce a classification model considering N randomly selected labeled documents in each run. In this study was considered $N = \{1\%, 10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 90\%\}$. This variation in the number of labeled documents allowed us to demonstrate better the behavior of the algorithms for different numbers of labeled documents, a trade-off between the number of labeled documents and classification performance, and the differences among the inductive supervised learning algorithms and semi-supervised learning algorithms when increasing the number of labeled documents. The remaining training examples were considered as unlabeled examples for semi-supervised learning algorithms. Thus, 100 executions were carried out, and an accuracy value was obtained in each execution. The final *Micro-*F^1 and *Macro-*F^1 values presented in the next section were an average of the 100 values obtained in the 10-fold cross-validation.

3.3 Operation of the Experiment

The inductive supervised learning algorithms, quoted in Subsection 3.2.2.3, were performed for analysis and comparison with semi-supervised learning. This comparison also allows

analyzing whether using unlabeled documents improves classification performance. In this study was used the implementations available in the Weka tool to evaluate the execution of the supervised algorithms (FRANK; HALL; WITTEN, 2016). The parameters and considerations of the inductive algorithms of supervised learning are listed below.

- ***Naïve Bayes (NB)***: default setting.
- ***Multinomial Naïve Bayes (MNB)***: default setting.
- ***Support Vector Machine (SVM)***: three types of kernel were considered: Linear, Polynomial (exponent = 2) and RBF (Radial Basis Function). Since the parameter C is real and positive, some authors set these values as 10^Y. For each type of kernel was considered $Y = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$.
- ***J48***: in this decision tree algorithm, the value 0.25 was used for the parameter *confidence factor*.
- ***k-NN***: in this algorithm was considered $k = \{7, 17, 37, 57\}$ (ROSSI *et al.*, 2014). Also, it was considered *k-NN* algorithm without and with a weighted vote, which gives for each of the nearest neighbors a weighted vote equal to $(1 - s)$, where s is a similarity measure among neighbors. Cosine was adopted as a similarity measure.

Semi-supervised learning algorithms based on the bipartite heterogeneous network model were used. The algorithms and their parameters are in Rossi, Lopes and Rezende (2016). All the iterative solutions were used for all the iterative solutions algorithms (GNetMine, LPBHN, TCBHN and TM). The maximum number of iterations was set to 1000. The parameters and considerations of the semi-supervised algorithms used in these experiments are defined as follows:

- **GNetMine**: it used $\alpha = \{0.1, 0.3, 0.5, 0.7, 0.9\}$.
- ***Label Propagation using Bipartite Heterogeneous Networks*** **(LPBHN)**: this is a parameter-free semi-supervised learning algorithm.
- ***Tag-based Model*** **(TM)**: it used $\beta = \{0.1, 1, 10, 100, 1000\}$, and $\gamma = \{0.1, 1, 10, 100, 1000\}$.
- ***Transductive Categorization based on Bipartite Heterogeneous Networks*** **(TCBHN)**: the iterative solution of TCBHN used is presented in Rossi *et al.* (2014). This iterative solution has two parameters η (error correction rate) and ε (minimum squared error). This algorithm used $\eta = \{0.01, 0.05, 0.1, 0.5\}$, $\varepsilon = 0.01$, 10 as the maximum number of global iterations and 100 as maximum number of local iterations, which gives a total of 1000 iterations.
- ***Expectation Maximization (EM)***: in this algorithm, it is necessary to define the parameter λ (weight of unlabeled examples during semi-supervised learning) and the number of

component for each class. This algorithm used $\lambda = \{0.1, 0.3, 0.5, 0.7, 0.9\}$ and 1, 2, 5, 10 components for each class.

3.4 Analysis and Discussion of the Results

The analysis of the experimental results considers two aspects: (i) the effectiveness of semi-supervised classification methods, *i.e.*, the effect of the number of labeled examples for the classification of severity of bug reports (related to the RQ1), and (ii) the overall impact of the term weighting strategy (binary or frequency) to represent the texts about the severity of bug reports (related to the RQ2). These aspects are related to the research questions presented in Section 3.2.1.

3.4.1 Answer to RQ1: effectiveness of classifiers

Regarding the first aspect, this study uses the best configurations of each method for each dataset by varying the number of labeled examples. The graphs of Figure 11 present the results obtained for the classification of severity type using the representation of texts considering the presence or absence of terms (binary) in the documents. Figure 12 presents the results obtained using the representation of texts considering the frequency of the terms in the documents. In Figures 11 and 12, the solid lines indicate the results of the supervised methods, and the dashed lines indicate the semi-supervised methods.

For most methods, the number of labeled examples has a limited impact, *i.e.*, a slight improvement in classification performance often does not justify the effort to use a more significant number of labeled examples. In general, the experimental results suggest the use of 20% of labeled samples was sufficient to stabilize classification performance for most classifiers (see Figures 11 and 12). Supervised classifiers, especially SVM and J48, were exceptions since they significantly improved the classification of bug report severity as the number of examples increased. However, it is important to argue that, in practical scenarios, there is not a large set of labeled examples due to the (human) effort to validate the training set. Thus, the next steps of the experimental analysis are based on 20% of labeled examples.

A statistical analysis was performed to compare the significance of the classification performance (*Micro-F*^{1} and *Macro-F*^{1}) of the best configurations of each classifier. Tables 4, and 6 present the results of *Micro-F*^{1} for each classifier in each dataset. Also, Tables 5, and 7 present the results of *Macro-F*^{1} for each classifier in each dataset. In Tables 4, 5, 6, and 7 cells with bold values indicate the best performance for a given dataset. The non-parametric Friedman test with Nemenyi post-hoc tests was used to compare multiple classifiers over multiple datasets. This procedure is considered the most robust strategy to compare several classifiers (DEMsAR, 2006; GARCIA; HERRERA, 2008), and the results were analyzed with 95% confidence level ($\alpha = 0.05$).

Table 4 – *Micro-F*[1] classification results for each classifier in each dataset with binary-based term weighting.

Dataset	J48	KNN	MNB	NB	SVM	EM	GNetMine	LPBHN	TagBased	TCBHN
Eclipse-CDT	0.166	0.196	0.194	0.231	0.303	0.756	**0.804**	**0.804**	**0.804**	0.766
Eclipse-JDT	0.177	0.178	0.179	0.203	0.311	0.736	**0.767**	0.766	**0.767**	0.740
Eclipse-PDE	0.102	0.202	0.171	0.212	0.335	0.784	**0.828**	**0.828**	**0.828**	0.798
Eclipse-Platform	0.219	0.175	0.180	0.211	0.246	0.735	**0.761**	**0.761**	**0.761**	0.742
Mozilla-Bugzilla	0.254	0.224	0.244	0.267	0.272	0.513	0.558	0.533	**0.566**	0.501
Mozilla-Thunderbird	0.172	0.222	0.223	0.241	0.250	0.604	0.647	0.643	**0.648**	0.642
Apache-Abdera	0.736	0.373	0.443	0.453	**0.762**	0.377	0.662	0.665	0.665	0.562
Apache-Ambira	0.224	0.254	0.239	0.255	0.373	0.619	**0.658**	**0.658**	**0.658**	0.630
Apache-OpenJPA	0.350	0.261	0.260	0.299	0.316	0.687	0.726	0.719	**0.738**	0.708

Source: Research data.

Table 5 – *Macro-F*[1] classification results for each classifier in each dataset with binary-based term weighting.

Dataset	J48	KNN	MNB	NB	SVM	EM	GNetMine	LPBHN	TagBased	TCBHN
Eclipse-CDT	0.198	0.210	0.209	0.211	**0.262**	0.201	0.160	0.159	0.182	0.228
Eclipse-JDT	0.200	0.211	**0.212**	0.211	0.211	0.197	0.153	0.147	0.171	0.204
Eclipse-PDE	0.191	0.208	0.208	0.210	**0.218**	0.191	0.159	0.158	0.175	0.192
Eclipse-Platform	0.200	0.217	0.217	0.216	**0.224**	0.220	0.159	0.151	0.210	0.222
Mozilla-Bugzilla	0.297	0.263	0.282	0.317	0.323	0.313	0.277	0.167	0.340	**0.347**
Mozilla-Thunderbird	0.229	0.242	0.246	0.249	0.256	0.250	0.210	0.133	0.221	**0.257**
Apache-Abdera	0.203	0.183	0.193	0.188	**0.231**	0.196	0.206	0.191	0.204	0.208
Apache-Ambira	0.232	0.254	**0.259**	0.258	0.255	0.230	0.174	0.162	0.212	0.254
Apache-OpenJPA	0.288	0.316	0.309	0.317	0.411	0.313	0.300	0.250	0.375	**0.425**

Source: Research data.

Table 6 – *Micro-F*[1] classification results for each classifier in each dataset with frequency-based term weighting.

Dataset	J48	KNN	MNB	NB	SVM	EM	GNetMine	LPBHN	TagBased	TCBHN
Eclipse-CDT	0.117	0.185	0.193	0.227	0.262	0.753	**0.804**	**0.804**	**0.804**	0.767
Eclipse-JDT	0.177	0.182	0.177	0.214	0.276	0.731	**0.767**	**0.766**	**0.767**	0.739
Eclipse-PDE	0.110	0.194	0.171	0.218	0.280	0.781	**0.828**	**0.828**	**0.828**	0.798
Eclipse-Platform	0.219	0.175	0.186	0.189	0.300	0.732	**0.761**	**0.761**	**0.761**	0.742
Mozilla-Bugzilla	0.253	0.223	0.244	0.257	0.250	0.511	0.558	0.533	**0.566**	0.501
Mozilla-Thunderbird	0.201	0.211	0.220	0.231	0.289	0.592	0.647	0.643	**0.648**	0.641
Apache-Abdera	**0.736**	0.370	0.449	0.436	0.730	0.368	0.662	0.665	0.665	0.557
Apache-Ambira	0.224	0.247	0.241	0.249	0.338	0.609	0.658	0.658	**0.659**	0.629
Apache-OpenJPA	0.350	0.263	0.264	0.300	0.320	0.685	0.726	0.719	**0.737**	0.708

Source: Research data.

Figure 11 – Results with binary term weighting.

(a) *Eclipse-CDT* (bin)

(b) *Eclipse-JDT* (bin)

(c) *Eclipse-PDE* (bin)

(d) *Eclipse-Platform* (bin)

(e) *Mozilla-Bugzilla* (bin)

(f) *Mozilla-Thunderbird* (bin)

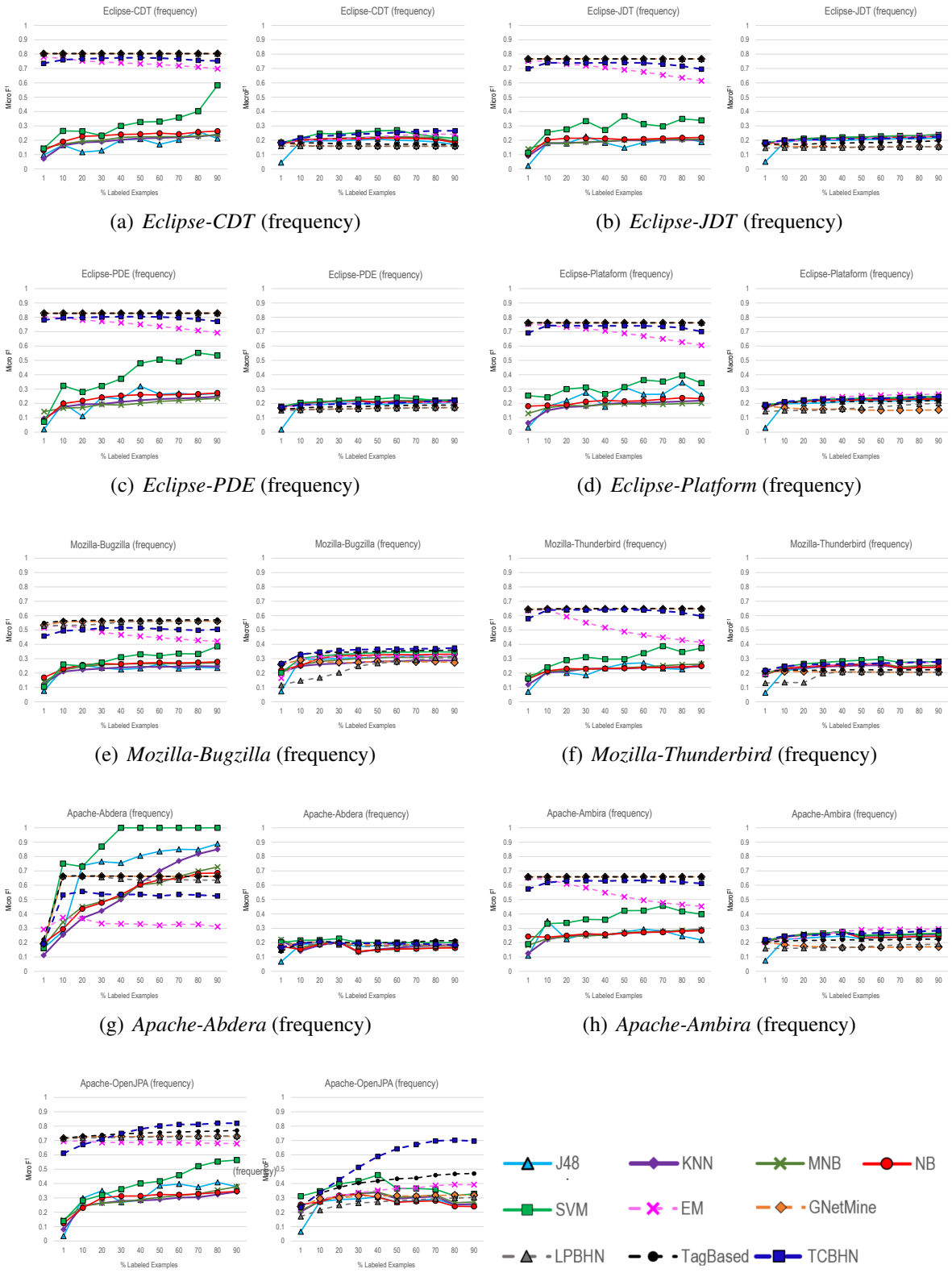
(g) *Apache-Abdera* (bin)

(h) *Apache-Ambira* (bin)

(i) *Apache-OpenJPA* (bin)

Source: Research data.

Figure 12 – Results with frequency term weighting.



(a) *Eclipse-CDT* (frequency)

(b) *Eclipse-JDT* (frequency)

(c) *Eclipse-PDE* (frequency)

(d) *Eclipse-Platform* (frequency)

(e) *Mozilla-Bugzilla* (frequency)

(f) *Mozilla-Thunderbird* (frequency)

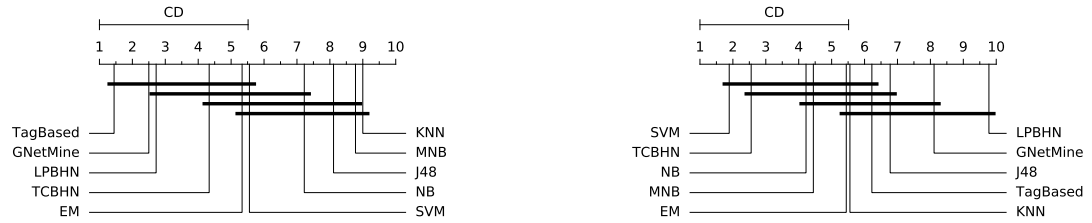
(g) *Apache-Abdera* (frequency)

(h) *Apache-Ambira* (frequency)

(i) *Apache-OpenJPA* (frequency)

Source: Research data.

Figure 13 – Critical difference diagram to compare statistical significance between multiple classifiers for datasets with binary-based term weighting.



(a) Critical difference diagram *Micro-F*^{1} measure (b) Critical difference diagram *Macro-F*^{1} measure

Source: Elaborated by the author.

Table 7 – *Macro-F*^{1} classification results for each classifier in each dataset with frequency-based term weighting.

Dataset	J48	KNN	MNB	NB	SVM	EM	GNetMine	LPBHN	TagBased	TCBHN
Eclipse-CDT	0.196	0.206	0.208	0.208	**0.247**	0.204	0.160	0.159	0.182	0.229
Eclipse-JDT	0.201	0.209	**0.212**	0.210	0.211	0.196	0.154	0.147	0.171	0.203
Eclipse-PDE	0.189	0.208	0.207	0.210	**0.213**	0.191	0.159	0.158	0.174	0.192
Eclipse-Platform	0.200	0.217	0.217	0.215	**0.222**	0.219	0.160	0.151	0.212	0.222
Mozilla-Bugzilla	0.295	0.261	0.280	0.303	0.319	0.312	0.279	0.167	0.339	**0.346**
Mozilla-Thunderbird	0.229	0.237	0.245	0.243	**0.264**	0.249	0.210	0.133	0.220	0.257
Apache-Abdera	0.203	0.182	0.193	0.188	**0.218**	0.198	0.207	0.191	0.207	0.207
Apache-Ambira	0.231	0.253	**0.260**	0.253	0.254	0.238	0.174	0.162	0.216	0.252
Apache-OpenJPA	0.288	0.314	0.310	0.317	0.397	0.311	0.300	0.250	0.374	**0.427**

Source: Research data.

Figures 13 (a), 13 (b), 14 (a), and 14 (b) present a graphical analysis of the statistical test called the critical difference diagram (DEMsAR, 2006). In these diagrams, the classifiers are sorted according to their position in the overall classification performance ranking. If there is no statistically significant difference between two classifiers, then they are connected by a line.

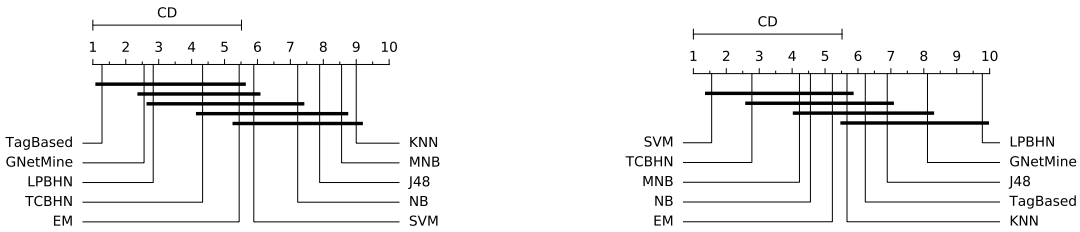
Considering the *Micro-F*^{1} measure, the semi-supervised Tag-based Model (TagBased) network-based algorithm obtained the first position in the ranking. In addition, the TagBased algorithm showed better results than all supervised algorithms with statistically significant differences (see Figures 14 (a)), except SVM (see Figures 13 (a)).

Statistical analysis reveals that the semi-supervised classifiers obtain competitive results concerning the supervised classifiers for both evaluation measures (*Micro-F*^{1} and *Macro-F*^{1}) and term-weighting techniques. This result is promising for research involving the prediction of bug report severity. It indicates that it is appropriate to obtain a small set of expert-labeled examples to predict a large set of bug reports. The transductive classifiers TCBHN and TagBased proved to be efficient considering both labeled examples and unlabeled examples during the classifier learning process.

Thus, in relation to the first research question (*Do semi-supervised learning methods perform better to support Severity Bug Report Prediction than supervised learning methods?*),

the experimental analysis allows us to conclude that semi-supervised classification is more recommended for the task classifying the severity of bug reports. Even without achieving a statistically superior performance in all scenarios, the TCBHN and TagBased classifiers presented a stable classification performance in several datasets. Therefore, its viability is recommended for future research in the area. On the other hand, if there is a requirement to use a supervised classification method, the SVM classifier is recommended, which obtained competing results even with few labeled examples.

Figure 14 – Critical difference diagram to compare statistical significance between multiple classifiers for datasets with frequency-based term weighting.



(a) Critical difference diagram *Micro-F*[1] measure (b) Critical difference diagram *Macro-F*[1] measure

Source: Elaborated by the author.

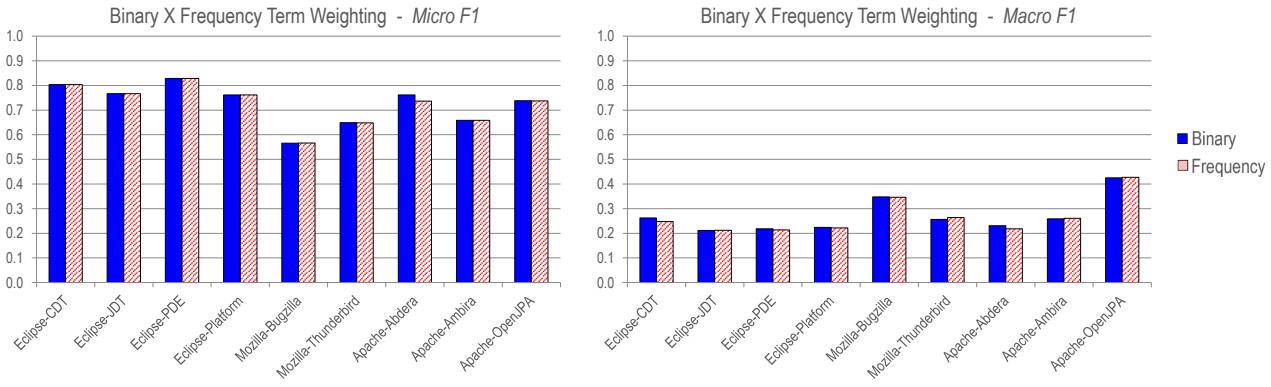
3.4.2 Answer to RQ2: impact of strategies to represent the text

Regarding the second aspect of the experimental evaluation, this study uses the best classification results for each dataset according to the term-weighting technique (binary or frequency). Thus, it is possible to directly evaluate which term-weighting technique is most effective for learning classifiers in the severity of the bug report domain.

Figure 15 presents a general comparison of each term-weighting technique for all datasets considering both F1-Micro and F1-Macro measures, respectively. The two techniques were statistically compared by means of an analysis of variance. For the choice of the most appropriate statistical test, it was verified that the experimental results are not normally (Gaussian) distributed. Therefore, the use of a non-parametric test is recommended. The Wilcoxon test was used, which is distribution free and powerful to compare two techniques over multiple datasets (DEMsAR, 2006; GARCIA; HERRERA, 2008). In this sense, a statistical analysis of these results based on Wilcoxon's non-parametric test reveals no statistically significant difference between the two term weighting techniques (with 95% of confidence).

According to the second research question (*Does term weighting (binary or frequency) in text representations impact on performance in Severity Bug Report Prediction?*), the choice of the term weighting technique can be done based on another requirement, such as interpretability of the representation or the text pre-processing time, since they do not affect the classification performance.

Figure 15 – General comparison of the classification performance for each term weighting technique.



Source: Elaborated by the author.

3.5 Threats to Validity

The threats to validity of the results of this study are classified into four types:

- **Construct validity**: Construct validity is about obtaining the right measure and whether it was defined the proper scope about what is considered an empirical study. It is possibility incorrectly use the data and the treatments in this study but, to minimize this threat, it is necessary to select a traditional dataset and the comparison strategy of ML methods from previous work (LAMKANFI; PÉREZ; DEMEYER, 2013).

- **Internal validity**: As only bug reports with "resolved" status were selected (because they represent bug reports that span the entire lifecycle), the proposed prediction model did not consider an immature bug report, such as those newly created by stakeholders. This restriction may have an impact on the performance of the proposed model.

- **External validity**: The results of this study can not be generalized to proprietary software since only Open Source software projects were analyzed. Since all the bug reports used in this study were extracted from the main repositories (Bugzilla and Jira), the results of the present study can not be generalized to other software projects.

- **Reliability**: About the possibility of replication of this study. All datasets and settings of machine learning algorithms are publicly available.

3.6 Final Remarks

The bug report is an essential source of information to support the process of software evolution. However, because registration and manipulation of bug reports are done manually, and those responsible for registration are not experts, it is only natural that misclassification errors

occur. For example, when the users maintain the severity attribute value with the default value offered by the BTS during the registration. This misclassification in the repositories increases the number of unlabelled bug reports.

Because of this problem, this chapter presents a comparative study between the approaches of supervised and semi-supervised learning in the context of the prediction of bug report severity.

As a summary, our contributions in this chapter are as follows:

- To the best of our knowledge, this experimental study is the first work on the application of semi-supervised classifier and data representation based on two-way heterogeneous networks in severity prediction of bug reports.
- 20% of the labeled samples were sufficient to stabilize the performance of most classifiers. A large number of labels caused a slight improvement in the performance of the classifiers.
- Semi-supervised classifiers obtained competitive results relative to supervised classifiers regardless of the number of labeled examples.
- Regarding the term weighting techniques, there is no statistically significant difference between the two techniques. Therefore, they do not impact on performance classifier.

This chapter considered the impact of semi-supervised methods only on the non-binary severity bug report prediction. One possibility of future work is evaluating semi-supervised methods' performance in the binary severity bug report prediction (severe or non-severe).

Considering this context, as future work, the impact of the following activities on the performance of binary classification could be investigated:

1. evaluate the use of class balancing techniques;
2. use techniques to enrich text representations; and
3. uses proprietary product data to compare the behavior of prediction models used in another context.

CHAPTER

4

BULNER: BUG LOCALIZATION WITH WORD EMBEDDINGS AND NETWORK REGULARIZATION

Bug localization (BL) from bug reports is an expensive step in the software life cycle because of the manual localization process. For example, the Mozilla project receives almost 300 bug reports per day, and each one needs a manual triage. Also, often, a bug report (84-93% of bugs) impacts one or a few files (THUNG *et al.*, 2014). A family of bug localization techniques uses Information Retrieval (IR). IR-based bug localization tool suggests defective parts of a software system by automatically relating a bug report's vocabulary and associated source code files. IR often uses Vector Space Model (VSM) but, due to VSM limitations, recent studies apply distributional semantics of words (RAHMAN; ROY, 2018).

The high severity levels (i.e., Blocker, Critical or Major) correctly assigned to bug reports impact major functionalities of an application. In general, when IR-based bug localization tools are applied on high severity bugs, they tend to find the exact faulty source files. That happens because such bugs often contain highly relevant words in their description (LE; THUNG; LO, 2017). Also, the severity of a bug report defines how quickly the developers need to identify where the bug is located in a software product to be fixed (TAN *et al.*, 2020).

In this chapter, BULNER is presented, an IR-based bug localization method, which stands for BUg Localization with word embeddings and NEtwork Regularization. Barbosa *et al.* (2019) reported the content of this chapter.

BULNER considers both word embedding features of bug reports and features extracted from project source file metrics. These features were combined in an information network proposed in BULNER. It is presented a network regularization-based machine learning method that obtains a more appropriate representation model for identifying potential buggy files from bug report texts. This research answers the following research questions: **RQ1** - How effective

is BULNER? **RQ2** - What is the contribution of each model? It was carried out an experimental evaluation using three well-known real-world datasets. BULNER is competitive with two other state-of-the-art methods. The experimental results indicate that combining different representation models, such as information networks and the vector-space model, is a promising method.

This chapter is organized as follows. Section 4.1 contains relevant concepts associated with bug localization data model representation. Also, this section presents the state of the art of bug localization. Section 4.2 presents the proposed method (BULNER), while Section 4.3 presents the experimental evaluation. Section 4.4 discusses our results and presents the threats to the validity of this proposed method. Finally, Section 4.5 presents our conclusions.

4.1 Bug Localization Data Model Representation

Bug localization has been modeled as an information retrieval task, where the bug report is treated as a query, and sources code files that conform to the system are documents. The goal is to select the files that better match the query based on a defined similarity measure. The effectiveness of the similarity measure depends on the text representation model of bug reports, often based on Bag-of-Words (BoW) and Word Embeddings (WEmb).

In bug localization context, each source code is defined as weights' vector in BoW, and they use cosine similarity function to identify closely related vector. Zhou, Zhang and Lo (2012) define BugLocator, an IR-based bug localization method based on the revised Vector Space Model. From the initial bug report, BugLocator applies textual similarity using similar bugs' information that was fixed before. Then, it ranks all suspicious source files (ZHOU; ZHANG; LO, 2012).

WEmb are a mapping table from words to continuous vectors (e.g., vec("dog") = [0.8, 0.3, 0.1], vec("cat") = [0.7, 0.5, 0.1] , vec("pasta") = [0.2, 0.1, 0.7]). In this example, the first parameter of each word represents some kind of animal. We could calculate the semantic similarity between words by cosine similarity and consequently calculate the similarity between sentences or entire documents. To obtain each above vector, it is used Skip-gram model proposed in the word2vec method for language modeling (MIKOLOV *et al.*, 2013b). It is an unsupervised method that defines each word meaning in its context (e.g., context("dog") = ["Pet," "tail," "smell,"], context("cat") = ["pet," "tail," "home"]). Two sets of context words may also have common concepts. According to the distributional hypothesis, we can estimate how close these two words are to each other by comparing with other words in the same context (MIKOLOV *et al.*, 2013b).

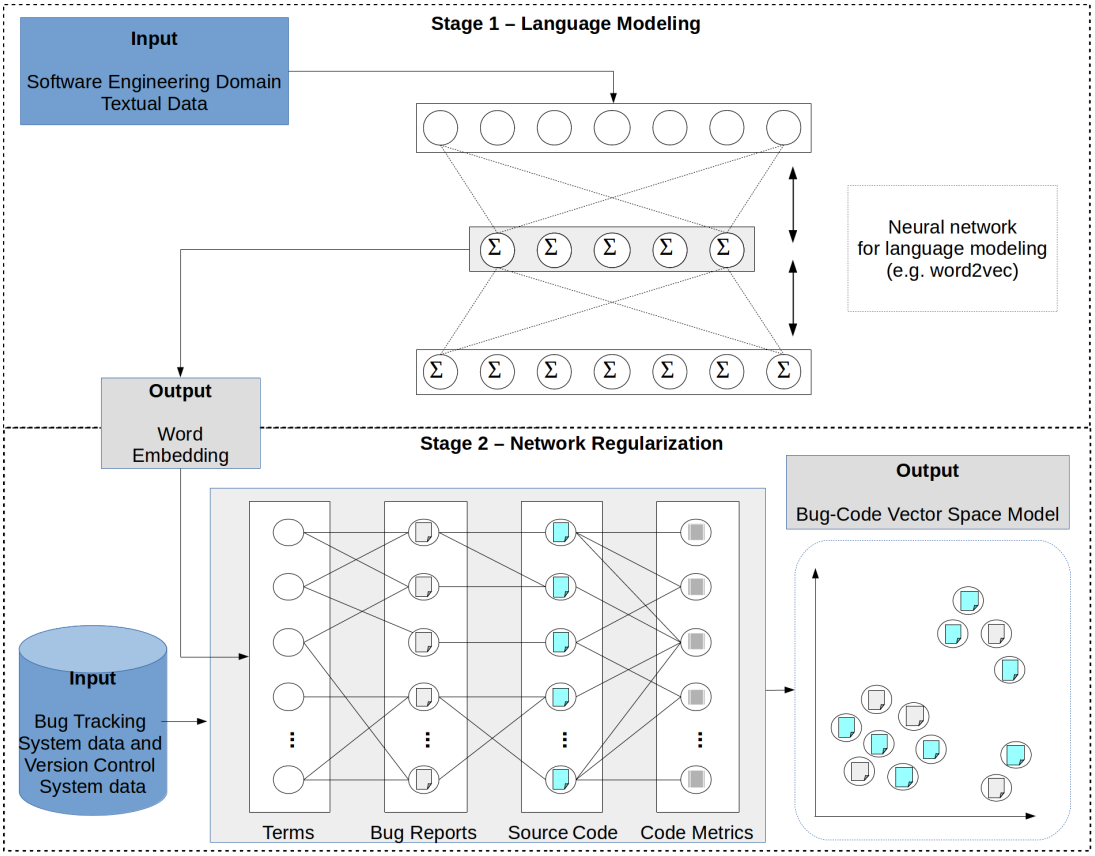
Ye *et al.* (2016) use word embedding to train on software documents (API documents, reference documents, and tutorials). First, they adapt the Skip-gram model and aggregated software documents to estimate their semantic similarities. Then, from an initial bug report

(query document), the bug localization model computes the ranking score for all source code (Ye *et al.*, 2016).

4.2 Proposed Method

In this section, it is introduced a new method for Bug Localization, called BULNER. This method innovates by considering the semantic content of bug reports through language models and source code content through software metrics. Figure 16 shows an overview of the BULNER method. The method has two stages: (1) language modeling and (2) network regularization. Given a new bug report, BULNER identifies the most similar bug reports and the source code files that probably contain the related bug. While the BULNER's first stage enables more accurate computation of the similarity between bug reports, the second performs fine-tuning of the language model by considering relationships between bug reports, source code files, and code metrics.

Figure 16 – An overview of the BULNER.



Source: Barbosa *et al.* (2019).

Unlike existing methods that calculate the similarity between bug reports through keywords or language models, BULNER learns a new vector space model to directly compares bug reports and source code files. We call this new vector space Bug-Code Vector Space Model.

Moreover, our method allows the inclusion of domain information, such as code metrics, during the learning process of this vector space model.

The first stage of the BULNER method uses a neural network. The primary purpose of this stage is to learn word vector representations from a sizeable textual dataset of the software engineering domain. For example, the term 'abort' has the following correlated terms in the language model used in BULNER: 'interrupted,' 'terminate,' 'halt,' 'timed-out,' and 'exit.' The BULNER method uses the language model based on the skip-gram model trained over 15GB of textual data from Stack Overflow posts, as proposed in Efstathiou, Chatzilenas and Spinellis (2018). The output of the skip-gram model is a representation called word embedding, where each term t contains a representation $\mathbf{Y}(t)$ in the d-dimensional vector space, i.e., $\mathbf{Y}(t) \in \mathbb{R}^d$.

The second stage of the BULNER method uses network regularization to perform the fine-tuning of the model obtained in the first stage. Given a bug reports dataset, we propose a heterogeneous network-based representation $N = (O, R, W)$, where O represents a set of objects o_i of the network, R represents a set of relations r_{o_i,o_j} between objects, and W represents a set of weights $w_{r_{o_i,o_j}}$ of the relations. We organize the set of network objects into four different types $O = \{O_B, O_T, O_S, O_M\}$, where O_B are objects that identify each bug report, O_T are terms t extracted from textual data of the bug reports, O_S are source files related to bug reports, and O_M are code metrics (discretized into intervals) computed from the source code files.

The general idea of network regularization is to obtain a new representation $\mathbf{F} \in \mathbb{R}^d$ in the d-dimensional vector space model, which satisfies two assumptions: (1) two objects o_i and o_j that share neighbors in the network must have similar vector representation, i.e., $\mathbf{F}(o_i) \sim \mathbf{F}(o_j)$, and (2) term-type objects in the network must have vector representation similar to the word embedding representation, i.e., $\mathbf{F}(t) \sim \mathbf{Y}(t)$.

It is inspired by the theoretical regularization framework of Ji *et al.* (2010). In Equation 4.1 we propose a regularization function for the BULNER method, where the goal is to minimize the function according to a representation model $\mathbf{F}$ for all objects of the network, given a word embedding $\mathbf{Y}$. Equation 4.1 uses a regularization function similar to LPBHN (Equation 2.4 on Page 37) (ROSSI; LOPES; REZENDE, 2014; ZHU; GHAHRAMANI; LAFFERTY, 2003).

$$Q(\mathbf{F}) = \sum_{O_i,O_j}^{\{O_B,O_T,O_S,O_M\}} \frac{1}{2} \sum_{o_i \in O_i} \sum_{o_j \in O_j} w_{r_{o_i,o_j}} \left|\mathbf{F}(o_i) - \mathbf{F}(o_j)\right|^2 + \lim_{\mu \to \infty} \mu \sum_{t \in O_T} \left|\mathbf{F}(t) - \mathbf{Y}(t)\right|^2 \tag{4.1}$$

The first term of the regularization function is responsible for the first assumption, in which related objects must have similar representations to minimize the distance $w_{r_{o_i,o_j}} \left|\mathbf{F}(o_i) - \mathbf{F}(o_j)\right|^2$. Regarding the second assumption, the proposed regularization function ensures that the d-dimensional representation of a term will remain the same as the word embedding representation, i.e., $\lim_{\mu \to \infty} \mu \sum_{t \in O_T} \left|\mathbf{F}(t) - \mathbf{Y}(t)\right|^2$.

In practical terms, we can minimize the regularization function of Equation 4.1 by using label (information) propagation techniques. In this case, BULNER initializes the representation of the term type objects according to word embedding $\mathbf{Y}$, whereas the representation of the remaining objects of the network can be randomly initialized. In each iteration, BULNER propagates the information of the term objects (i.e., WEmb) to the objects of the bug report type. The information is then propagated from bug reports to source code files objects and then to objects representing code metrics. The information is propagated back and each object $o_i \in O$ adjusts its $\mathbf{F}(o_i)$ representation. This process continues until there are no more significant changes in the $\mathbf{F}$ representation or until it reaches a maximum number of iterations, i.e., until the convergence of the BULNER method in which $\mathbf{F}$ is the learned representation for bug-code vector space model.

After the regularization process, we can directly compute the similarity between a bug report object $o_b \in O_B$ and a source code type object $o_s \in O_S$ as defined in the cosine similarity of Equation 4.2.

$$\cos(o_b, o_s) = \frac{\mathbf{F}(o_b) \cdot \mathbf{F}(o_s)}{||\mathbf{F}(o_b)||\,||\mathbf{F}(o_s)||} \tag{4.2}$$

$$sim(r_{new}, r_{train}) = (1 - \alpha)BOW(r_{new}, r_{train}) + \alpha BULNER(r_{new}, r_{train}) \tag{4.3}$$

A new bug report can be represented in the bug-code space vector through the word embedding of its terms and thus obtain a representation $\mathbf{F}(o_b)$. Equation 4.3 defines a new similarity function for bug localization in BULNER, which is a linear combination of similarity between bug reports in the BoW model and similarity in the Bug-Code Vector Space model since new bug reports contain only textual information. While one can define the BoW function as the cosine similarity between bug report keywords, the BULNER function represents the cosine similarity between the bug report representation $\mathbf{F}(o_b)$ and the source code file $\mathbf{F}(o_s)$ (Equation 4.2). The α parameter is a combination factor that allows defining the weight of the bug-code vector space model in the new similarity function, which can be estimated empirically. We use this new similarity function to calculate the scores for source files potentially related to the new bug reports.

4.3 Experimental Evaluation

We conducted this experiment to evaluate the application of word embeddings and Network Regularization on three open source projects in context of bug localization. The overview of the experiment is presented in this section.

4.3.1 Definition of Research Question

This experiment compares BULNER with two different models based on BoW. The hypothesis is based on the fact that BULNER uses a data representation that values the relationships between different objects through a HEN. The assumption is that the bug localization data representation based on HEN enriches the representational model. Consequently, it offers a performance gain over BoW-based representational models.

In particular, this study seeks to answer the following research questions:

- **RQ1**: How effective is BULNER?
- **RQ2**: What is the contribution of each model?

4.3.2 Experiment Definition

We defined our experiment as follows (WOHLIN *et al.*, 2012):

- analyze *network regularization-based machine learning method*,
- for the purpose of *evaluating bug localization*,
- with respect to *model performance*,
- from the point of view of the *researcher*,
- in the context of *open source projects*.

4.3.3 Dataset

To evaluate our approach, we obtained data from three open source projects: AspectJ, Birt, and Tomcat. We extracted bug report data associated with each project from the Bugzilla repository provided by Ye, Bunescu and Liu (2014), while we mined each project's repository (located in GitHub) to obtain software metrics. For each bug report, we checked out a before-fix version of the source code from Github. Then, we used Understand™[1] to calculate different software metrics (object oriented, volume and complexity metrics). Appendix A presents details from software metrics used.

[1] Understand™: <https://scitools.com/>.

4.3.4 Baselines

We consider two methods in the literature for experimental evaluation. The first uses only the BoW model and cosine similarity to retrieve similar bug reports and related source files (ZHOU; ZHANG; LO, 2012). The second combines the BoW model and WEmb models (Ye *et al.*, 2016).

4.3.5 Evaluation Metrics

We use Mean Average Precision (MAP) as an evaluation criterion. Equation 4.4 calculates the precision in identifying NP buggy files, given a maximum value of k recommendations. Equation 4.5 calculates the precision average, where NPI is the total number of positive instances. Equation 4.6 calculates the MAP, where M is the total of bug reports. We use MAP with $k = \{1, 5, 10\}$, presented as MAP@1, MAP@5 and MAP@10.

$$P(k) = \frac{NP}{k} \qquad (4.4) \qquad AP = \sum_{i=1}^{N} \frac{P(i)}{NPI} \qquad (4.5) \qquad MAP = \frac{1}{M} \sum_{j=1}^{M} AP(j) \qquad (4.6)$$

4.4 Results and Discussion

This section presents the results of the proposed method for each research question and threats to validity.

4.4.1 RQ1: How effective is BULNER?

Table 8 shows the best method's performance (max MAP performance independent of combination factor) for the three datasets. The results suggest that BULNER is the best method for the three datasets. BULNER achieves this result by combining BoW with WEmb and Network Regularization. It receives as input a heterogeneous network (N), and it treats each type of object and link separately. Moreover, BULNER minimizes classification error when preserving consistency for each relation graph (R) by applying graph regularization (JI *et al.*, 2010). However, a statistical analysis of the results (Student's t-Tests with 95% confidence) does not allow us to state that the BULNER method is significantly superior to other methods, mainly due to the few datasets used in the experimental evaluation.

Table 8 – MAP Performance Comparison with the State-of-the-art Methods.

	AspectJ			**Tomcat**			**Birt**		
Methods	MAP@1	MAP@5	MAP@10	MAP@1	MAP@5	MAP@10	MAP@1	MAP@5	MAP@10
BoW+Cosine	0.1185	0.1738	0.1879	0.2121	0.2908	0.3017	0.0900	0.1372	0.1477
Embedding	0.1185	0.1738	0.1811	0.2134	0.2908	0.3001	0.0928	0.1402	0.1504
Bulner	0.1390	0.1913	0.2059	0.2201	0.2952	0.3078	0.0968	0.1420	0.1525

Source: Barbosa *et al.* (2019).

Table 9 presents the comparison between the best Bulner's performance and the approach of Ye, Bunescu and Liu (2014). Bulner uses the extended dataset by Ye, Bunescu and Liu (2014) with software metrics, but it does not outperform the approach of Ye, Bunescu and Liu (2014). Unlike text tokens appearing in bug reports and software metrics, Ye, Bunescu and Liu (2014) correlate all text tokens appearing in bug reports and souce code files (YE; BUNESCU; LIU, 2014).

Table 9 – MAP Performance Comparison with the approach of Ye, Bunescu and Liu (2014).

	AspectJ	**Tomcat**	**Birt**
Methods	MAP	MAP	MAP
Approach of Ye, Bunescu and Liu (2014)	0.25	0.49	0.15
Bulner	0.20	0.31	0.15

Source: Ye, Bunescu and Liu (2014), Barbosa *et al.* (2019).

4.4.2 RQ2: What is the contribution of each method?

We evaluate the contributions of each method by the combination factor (α). In Figure 17, when α=0, all methods have the performance equal to baseline (BoW+Cosine), but when we increment α, each method has different behavior. In general, BULNER has better performance for: AspectJ when $0.15 < \alpha < 0.3$; Birt when $\alpha = 0.1$ and Tomcat when $0.05 < \alpha < 0.1$. These variations between software project occur because each one has its context.

4.4.3 Threats to Validity

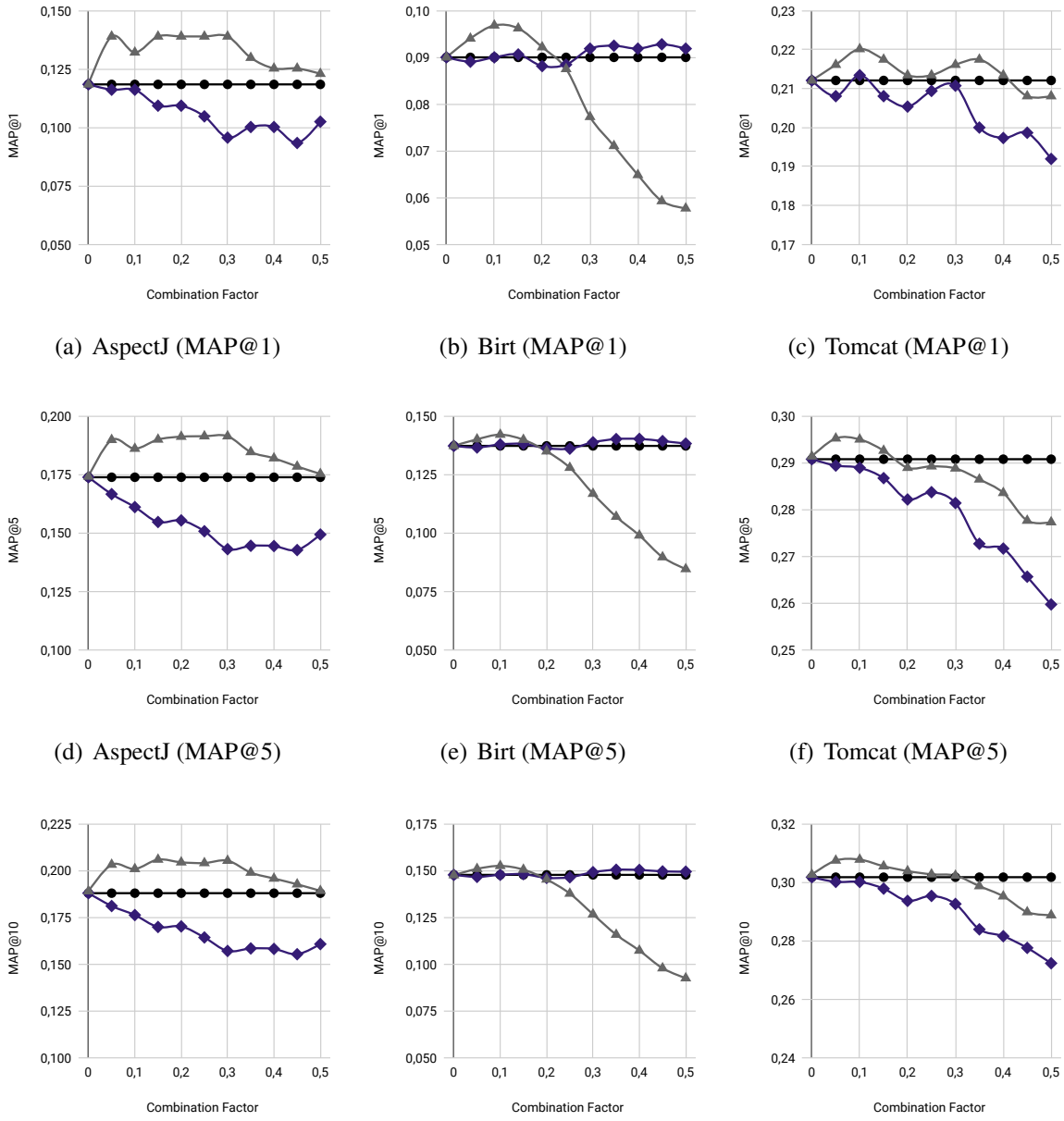
Regarding **Internal validity**, as only bug reports with "resolved" status were selected (because they represent bug reports that span the entire life cycle), the proposed model did not consider an immature bug report, such as those newly created by stakeholders. This restriction may have an impact on the performance of the proposed model. Furthermore, regarding **External validity**, the results of this study can not be generalized to proprietary software since only Open Source software projects were analyzed. Finally, concerning **Conclusion validity**, we choose the dataset provided by Ye, Bunescu and Liu (2014) that is largely used to maximize the quality of the data collected.

4.5 Final Remarks

We proposed a method for BUg Localization with word embeddings and NEtwork Regularization, which locates bugs in terms of source files from bug reports and source code data. Our method is competitive with two other state-of-the-art methods. BULNER is very promising. In one case, it can recommend the 30% suspicious file within top 5 for one bug report.

Bug localization is interconnected with severity bug report prediction and other activities related to the resolution of bug reports (e.g., fixer recommendation). Then, approaches that

Figure 17 – Methods' performance. ▲ BULNER; ● BoW+Cosine; ◆ Embedding.



(a) AspectJ (MAP@1) (b) Birt (MAP@1) (c) Tomcat (MAP@1)

(d) AspectJ (MAP@5) (e) Birt (MAP@5) (f) Tomcat (MAP@5)

(g) AspectJ (MAP@10) (h) Birt (MAP@10) (i) Tomcat (MAP@10)

Source: Barbosa *et al.* (2019).

integrate these activities could help software managers' decisions making in the maintenance process.

For future works, we intend to compare our work with different types of network embedding methods, for example, advanced information preserving network embedding. Additionally, we plan to extend our dataset with source code change genealogy and evaluate the impact on the performance of the methods. Our BULNER source code, as well as the datasets used, are publicly available at https://github.com/jacsonrbinf/bulner.

CHAPTER

5

A FRAMEWORK TO SUPPORT THE BUG REPORT RESOLUTION PROCESS WITH HETEROGENEOUS INFORMATION NETWORK

The manual execution of bug report resolution (BRR) activities takes much time from the stakeholders involved in the development process. It can also generate rework, for example, when the correct severity or bug report fixer does not occur.

To automate BRR activities, software engineers may use machine learning algorithms to automate the severity analysis (Ramay *et al.*, 2019; LAMKANFI *et al.*, 2010), to recommend bug fixers (MANI; SANKARAN; ARALIKATTE, 2019; ČUBRANIć, 2004), and to localize bugs in a code base (LAM *et al.*, 2017). In general, they use textual attributes of software artifacts (e.g., bug reports or source code) as input to machine learning models.

The aforementioned three activities related to the resolution of bug reports are interconnected. However, the existing approaches handle those activities in an isolated and disconnected way. Consequently, software managers and the quality assurance team do not have a global view of BRR activities in this context. Then in an isolated and disconnected way, they could have difficulty in decision making in planning maintenance activities. Moreover, automatic BRR activities increase productivity (ZHOU *et al.*, 2016).

The existing approaches also define machine learning models that use representations with a data source, called a monomodal perspective (e.g., a representation that uses only bug reports). Consequently, such approaches ignore the semantic dependency between different data sources and information related to the domain (Hoang *et al.*, 2019). On the other hand, the multimodal perspective (e.g., representation that using bug reports and program blocks) employs the enrichment of the domain representation with data from different sources (Hoang *et al.*,

2019).

Faced with the gaps mentioned above, this chapter proposes HENBUR (HEterogeneous information Network to support the BUg report Resolution process). This approach proposes a holistic solution for the three activities of the BRR process (bug report severity prediction, fixer recommendation, and bug localization). HENBUR uses a multimodal perspective (representation using bug reports and software metrics) implemented with a heterogeneous information network (HEN).

HEN is an alternative that allows multimodal perspective representation. Heterogeneous information networks represent different objects and their respective relationships in the same structure. HEN has been applied in other domains and has offered competitive results compared to traditional data representations (Shi *et al.*, 2017). Recent studies have shown promising results for text classification through HEN (ROSSI; LOPES; REZENDE, 2016; ROSSI; LOPES; REZENDE, 2014).

The main contributions of this chapter are summarized as follows:

- a holistic multimodal approach to improve the bug resolution process;
- use of the heterogeneous information network as a generator of features for bug resolution process;
- availability of implementations of BRR activities (severity bug report prediction, fixer recommendation, and bug localization) in Python programming language; and
- experimental results of the impact of HEN representation on bug report resolution process.

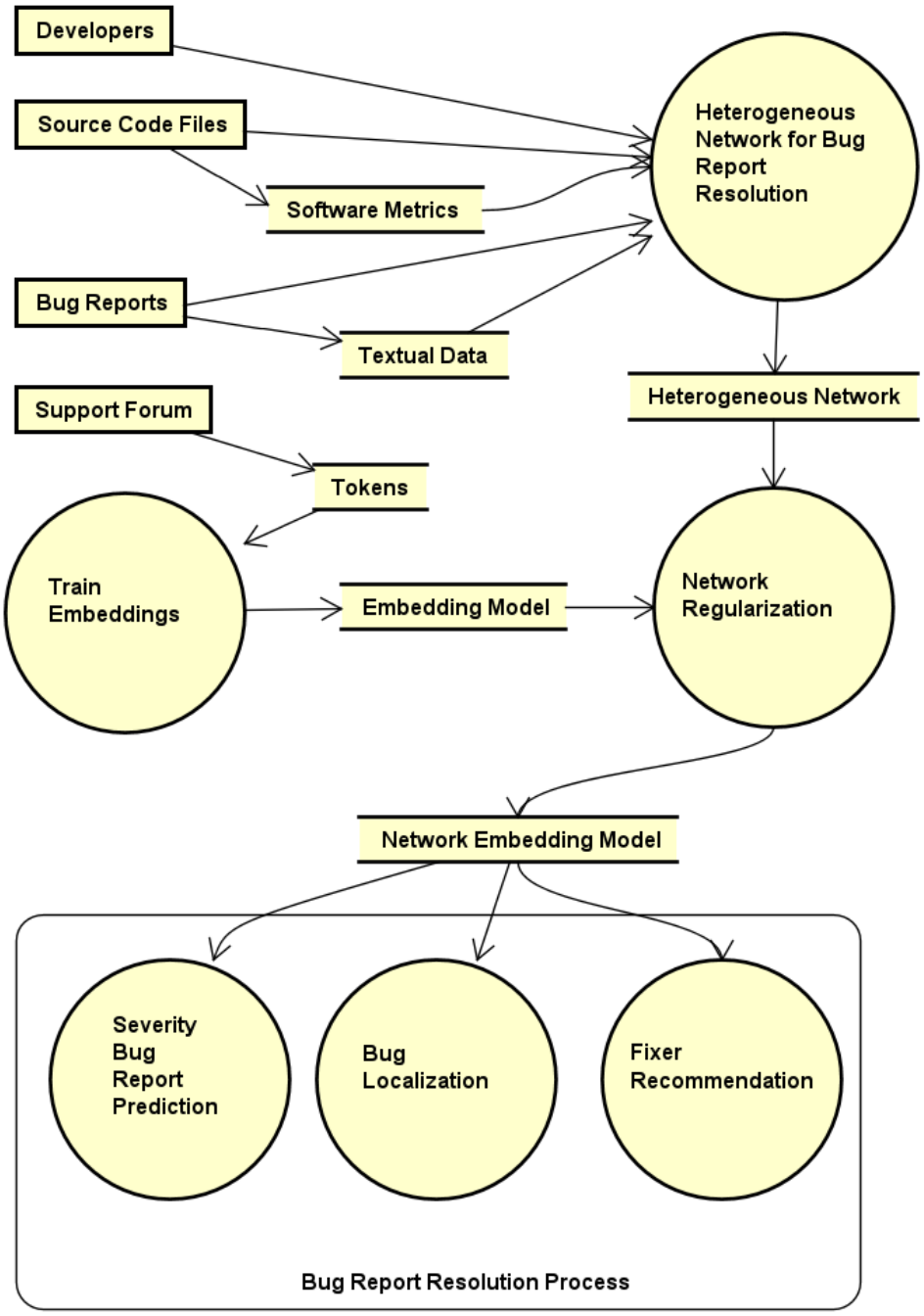
The remainder of this chapter is organized as follows. The overview about HENBUR is presented in Section 5.1. Section 5.2 presents the design of the conducted experiment to validate this approach, while Section 5.3 presents the validation results. Section 5.4 presents a description of the threats to the validity of this investigation. Finally, conclusions are presented in Section 5.5.

5.1 Embedding-based Multimodal Framework with a Heterogeneous Information Network to Support Bug Report Resolution

This section presents a general framework to support bug report resolution called HENBUR (HEterogeneous information Network to support the BUg report Resolution process). As shown in Figure 18, HENBUR is capable of: (i) representing different types of information and relationships between software artifacts from software repositories through nodes and edges in

a heterogeneous network; and (ii) learning an embedding model that maps all network nodes into a unified m-dimensional vector-space representation. We show that HENBUR is useful for generating rich features to be incorporated into traditional classification and ranking methods, thereby improving bug severity prediction models, fixer recommendation, and bug localization models.

Figure 18 – Data flow diagram for HENBUR.



Source: Elaborated by the author.

5.1.1 Heterogeneous Information Network for Bug Report Resolution

First, we introduce our heterogeneous network as a representation model for the different information from software repositories (e.g., Bugzilla and GitHub repositories). In particular, we consider the following objects and relationships:

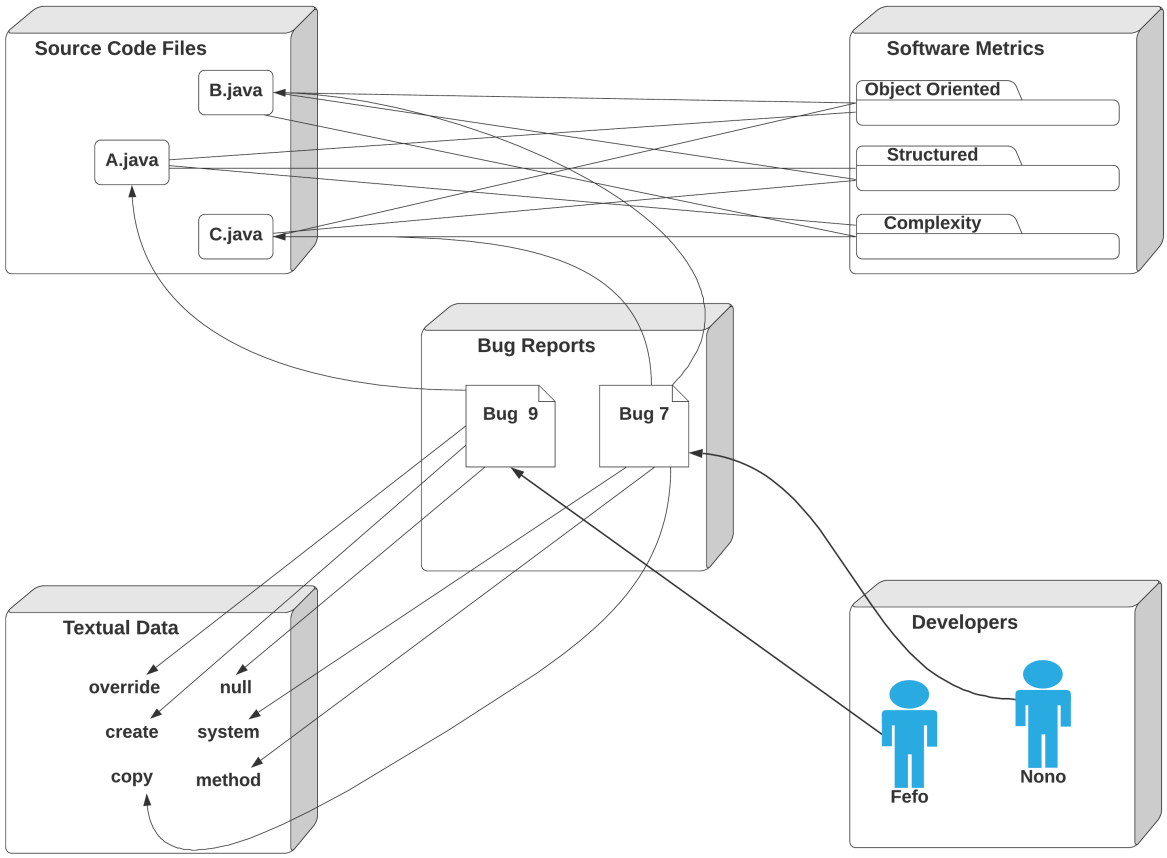
- **Bug reports** – nodes that refer to fixed bug reports.
- **Source code files** – nodes that correspond to each source code file associated with bug report status changes to the status *resolved fixed*, *verified fixed*, or *closed fixed*.
- **Software metrics** – nodes that represent different metrics calculated from each source code file. HENBUR includes the following types of metrics (GÓMEZ *et al.*, 2008): Object Oriented (e.g., Depth of Inheritance Tree), Structured (e.g., Source Lines of Code), and Complexity (e.g., Cyclomatic Complexity). We normalized each metric from numeric measures to label (e.g., lv0, lv1, and lv2).
- **Developers** – nodes that correspond to the last developer that fixed each bug report. We obtained this information from source code repository.
- **Textual data** – nodes that represent terms and expressions extracted from bug reports. Such nodes play an essential role in HENBUR; word vectors of these textual data are propagated to other nodes through a network regularization method proposed herein.

The relationships in the heterogeneous information network have different meanings according to the nodes involved:

- We connect bug reports with textual information to indicate that specific terms and expressions are associated with bug reports.
- Source code files are connected to each bug report to reflect the software maintenance and evolution.
- Source code files are also connected to nodes representing different metrics of software.
- Nodes representing developers are connected to bug reports to indicate the associated developers' engagement during the correction of a bug.

Figure 19 illustrates the conceptual model of the proposed heterogeneous information network. We divide each type of object into layers (represented by box in Figure 19) and add edges relating to objects (e.g., A.java, Bug 9 and Fefo) from different layers.

Figure 19 – Heterogeneous information network for bug report resolution knowledge representation.



Source: Elaborated by the author.

Formally, our heterogeneous network is represented by $N = (O, R, W)$, where O represents a set of objects o_i of the network, R represents a set of relations r_{o_i,o_j} between objects, and W represents a set of weights $w_{r_{o_i,o_j}}$ of the relations.

We organize the set of network objects into five different types $O = \{O_B, O_T, O_S, O_M, O_D\}$, where O_B are objects that identify each bug report, O_T are terms t extracted from textual data of the bug reports, O_S are source code files related to bug reports, O_M are software metrics (discretized into intervals) computed from the source code files, and O_D represents developers.

5.1.2 Train Embedding

This phase of the HENBUR uses a neural network to learn word vector representations from a textual dataset of the software engineering domain. As proposed by Efstathiou, Chatzilenas and Spinellis (2018), HENBUR uses the language model based on the skip-gram model trained over 15GB of clean textual data from the support forum Stack Overflow (posts' data).

Efstathiou, Chatzilenas and Spinellis (2018) use skip-gram model to represent each word (called word embedding) based on the distributional hypothesis (HARRIS, 1954; MIKOLOV *et*

al., 2013a), where each term t contains a representation $\mathbf{Y}(t)$ in the d-dimensional vector space, i.e., $\mathbf{Y}(t) \in \mathbb{R}^d$.

5.1.3 Network Regularization

A heterogeneous information network for bug report resolution is challenging: (i) some data objects could have the label information, and (ii) respect type differences among relationships and objects while mining HEN. Networking regularization is one alternative to address these challenges (JI *et al.*, 2010).

Ji *et al.* (2010) defined a network regularization framework that treats each information (object and relation) separately in a heterogeneous network. Therefore, it preserves different semantic meanings in heterogeneous networks. This framework uses transductive classification to find the hidden structure of the information network (JI *et al.*, 2010).

Based on this regularization framework, we propose a network regularization in which the word vectors obtained from the train embedding phase are defined as vector information in the regularization process. Then, in each iteration of Algorithm 1, network nodes propagate their information vectors to neighboring nodes until the process converges (when there is no significant change in information vectors throughout the network).

The general idea of network regularization is to obtain a new representation $\mathbf{F} \in \mathbb{R}^d$ in the d-dimensional vector space model, which satisfies two assumptions: (i) two objects o_i and o_j that share neighbors in the network must have similar vector representation, i.e., $\mathbf{F}(o_i) \sim \mathbf{F}(o_j)$, and (ii) term-type objects in the network must have vector representation similar to the word embedding representation, i.e., $\mathbf{F}(t) \sim \mathbf{Y}(t)$. In Equation 5.1 we propose the regularization function, where the goal is to minimize the function according to a representation model $\mathbf{F}$ for all objects of the network, given a textual word embedding $\mathbf{Y}$.

$$
\begin{aligned}
Q(\mathbf{F}) = \; & \frac{1}{2} \sum_{o_t \in O_T} \sum_{o_b \in O_B} w_{r_{o_t,o_b}} \left| \mathbf{F}(o_t) - \mathbf{F}(o_b) \right|^2 \\
& + \frac{1}{2} \sum_{o_s \in O_S} \sum_{o_b \in O_B} w_{r_{o_s,o_b}} \left| \mathbf{F}(o_s) - \mathbf{F}(o_b) \right|^2 \\
& + \frac{1}{2} \sum_{o_s \in O_S} \sum_{o_m \in O_M} w_{r_{o_s,o_m}} \left| \mathbf{F}(o_s) - \mathbf{F}(o_m) \right|^2 \\
& + \frac{1}{2} \sum_{o_d \in O_D} \sum_{o_b \in O_B} w_{r_{o_d,o_b}} \left| \mathbf{F}(o_d) - \mathbf{F}(o_b) \right|^2 \\
& + \lim_{\mu \to \infty} \mu \sum_{o_t \in O_T} \left| \mathbf{F}(o_t) - \mathbf{Y}(o_t) \right|^2
\end{aligned}
\tag{5.1}
$$

Given $O_i, O_j \subset \{O_B, O_T, O_S, O_M, O_D\}$, the first four terms of the regularization function are responsible for the first assumption, in which related objects must have similar representations

to minimize the distance $w_{r_{o_i,o_j}} \left|\mathbf{F}(o_i) - \mathbf{F}(o_j)\right|^2$. Regarding the second assumption, the proposed regularization function ensures that the d-dimensional representation of a term will remain the same as the textual embedding model, i.e., as defined in Equation 5.2.

$$\lim_{\mu \to \infty} \mu \sum_{o_t \in O_T} \left|\mathbf{F}(o_t) - \mathbf{Y}(o_t)\right|^2 \tag{5.2}$$

Algorithm 1 – *HENBUR Label Propagation Algorithm*

Input: $O, \mathbf{Y}, \mathbf{W}, \mathbf{D}$
Output: $\mathbf{F}$

```
1  begin
2      P ← (D^-1) · W
3      repeat
4          while O_i, O_j ⊂ {O_B, O_T, O_S, O_M, O_D} do
5              F(O_i, O_j) ← P(O_i, O_j) · F(O_i, O_j)
6              F(O_T) ← Y(O_T)
7          end
8      until convergence or other stop condition;
9  end
10 return F
```

5.2 Experiment Design

To evaluate the approach presented in this chapter, we conducted a quasi-experiment (a type of empirical study where the assignment of treatments to subjects is not random) (WOHLIN *et al.*, 2012). To do so, we followed the guidelines proposed by Wohlin *et al.* (2012). The experimental design is detailed in the remainder of this section.

5.2.1 Definition of Research Question

Our work investigates how to integrate and reuse different software artifacts in a multimodal holistic way to support the BRR process. In this paper, we addressed the following research questions:

- **RQ1**: What is the practical significance of the HENBUR to the bug report resolution process?
- **RQ2**: How can the HEN representation improve bug report severity prediction?
- **RQ3**: How can the HEN representation improve fixer recommendation?
- **RQ4**: How can the HEN representation improve bug localization?

5.2.2 Experiment Definition

We defined our experiment as follows (WOHLIN *et al.*, 2012):

- analyze *HENBUR framework*,
- for the purpose of *evaluating bug report severity prediction, fixer recommendation, and bug localization*,
- with respect to *model performance*,
- from the point of view of the *researcher*,
- in the context of *open source projects*.

5.2.3 Preparation and Planning

This section presents the experiment planning steps executed: sample selection, description of the experimental package, the definition of variables, and general design principles.

5.2.3.1 Sample Selection

To validate our approach, we used data from six popular open source projects: AspectJ, Birt, Eclipse Platform UI (UI), Eclipse JDT (JDT), Eclipse SWT (SWT), and Tomcat. All of them come from the collected dataset by Ye, Bunescu and Liu (2014). Some previous studies also used these benchmark datasets (LAM *et al.*, 2015; Ye *et al.*, 2016; LAM *et al.*, 2017; POLISETTY; MIRANSKYY; BAsAR, 2019).

Table 10 presents the following data for the open source projects included in our experiment: the number of bug reports (Reports), the number of severe bug reports (Severe), the number of fixer bugs (Assignees), the average fixing time by bug report (Fixing time) and the date interval (Duration) by software project.

Table 10 – Bug report data used in our experiments.

Project	# Reports	Severe (# / %)	# Assignees	Fixing time (days)	Duration
AspectJ	593	111 / 18.72	9	53	2002-03-13 ~2013-12-04
Birt	4178	789 / 18.88	74	30.55	2005-06-14 ~2013-11-20
UI	6495	1010 / 15.55	152	99.42	2001-10-10 ~2013-12-17
JDT	6274	630 / 10.04	55	96.96	2001-10-10 ~2014-01-03
SWT	4151	764 / 18.41	42	87.6	2002-02-19 ~2014-01-17
Tomcat	1056	113 / 10.70	20	115.7	2002-07-06 ~2013-08-12

Source: Research data.

5.2.3.2 Experimental Package

In our quasi-experiment, we have three objects:

- bug report severity prediction;
- fixer recommendation; and
- bug localization.

The experiment package is publicly available at GitHub[1]. It has the following documents:

- Description of the benchmark dataset: The experiment package describes all the information used by each object.
- Definition of the objects: We used the Python programming language to define each object. The source code is publicly available on GitHub.
- Description of the hyper-parameters for each Machine learning classification algorithms: We chose six supervised classification algorithms (*k-nearest neighbor* (*k*NN), *Support Vector Machines* (SVM), *Decision tree* (DT), *Random forest* (RF), *Multinomial Naïve Bayes* (MNB) and *Multi-layer Perceptron* (MLP)), which are used in the state of the art to support bug report severity prediction and fixer recommendation.

5.2.3.3 Variables

Data representation is the independent variable (factor) controlled in our experiment. This independent variable assumes two values (HEN and BoW). Treatments are a combination of each data representation with: (i) six supervised classification algorithms to support bug report severity prediction and fixer recommendation tasks and (ii) euclidean distance ranking to support bug localization tasks.

The model performance is the dependent variable that is affected by the treatment. We use two sets of evaluation metrics to compare the model performance of our approach with other techniques: (i) Macro-averaged F1-score (*Macro-F*^{1}) for multi-class classification context to evaluate bug report severity prediction and fixer recommendation tasks and (ii) Mean Average Precision (MAP) to evaluate only bug localization tasks.

Macro-F^1 representing the harmonic mean of the $Precision^{Macro}$ and $Recall^{Macro}$ measurements was used, where both measures have the same weight (see Equation 5.3) (SOKOLOVA; LAPALME, 2009).

$$Macro{-}F^1 = 2 * \frac{Precision^{Macro} * Recall^{Macro}}{Precision^{Macro} + Recall^{Macro}} \tag{5.3}$$

[1] https://github.com/jacsonrbinf/henbur

The *Macro-F*[1] metric uses ***macro-averaging*** strategy. This strategy performs an average over the evaluations measures for each class. Then in Equations 5.4 and 5.5, respectively, we have *precision* and *recall* using macro-averaging strategy (ROSSI; LOPES; REZENDE, 2016):

$$Precision^{Macro} = \frac{\sum_{c_i \in C} Precision_{c_i}}{|C|}, \tag{5.4}$$

$$Recall^{Macro} = \frac{\sum_{c_i \in C} Recall_{c_i}}{|C|}. \tag{5.5}$$

Precision (see Equation 3.3) and *Recall* (see Equation 3.4) were calculated for each class in multi-class assessment.

Macro-averaging gives equal weight to each class. In this case, the number of TP in small classes is emphasized in macro-averaging scores.

To compare bug localization tasks' model performance, we use Mean Average Precision (MAP) as an evaluation criterion. We use MAP (see Equation 4.6) with $k = \{1, 5, 10\}$, presented as $MAP@1$, $MAP@5$ and $MAP@10$.

5.2.3.4 Experimental Design

In this experiment, we have one factor (data representation) and seven treatments (six supervised classification algorithms and euclidean distance ranking algorithm) with crossover (WOHLIN *et al.*, 2012).

Table 11 – Experimental design (Legend: Rep = Representation).

	Baselines (Control)		Treatments	
Bug report resolution tasks	**Rep**	**Method**	**Rep**	**Method**
Bug report severity prediction	BoW	Supervised machine learning	HEN	Supervised machine learning
Fixer recommendation	BoW	Supervised machine learning	HEN	Supervised machine learning
Bug localization	BoW	Euclidean Distance Ranking	HEN	Euclidean Distance Ranking

Source: Elaborated by the author.

Table 11 presents the employed experimental design in our study for each open source project.

5.2.4 Operation of the Experiment

We extended benchmark datasets from chapter 4 with bug fixers information. Also, we replicated the same information extraction techniques for three additional projects provided by Ye, Bunescu and Liu (2014): Eclipse Platform UI, Eclipse JDT, and Eclipse SWT.

We obtained the bug report data related to the aforementioned projects from the Bugzilla repository provided by Ye, Bunescu and Liu (2014). They collected only bug reports with status

marked as resolved fixed, verified fixed, or closed fixed. Furthermore, Ye, Bunescu and Liu (2014) applied Dallmeier and Zimmermann's heuristic to connect bug reports and their associated correction files (DALLMEIER; ZIMMERMANN, 2007).

To obtain software metrics for each correction files, we checked out a before-fix version of each project's source code (from Github). Then, we used Understand™ to calculate different software metrics (object-oriented, structured, and complexity metrics).

5.2.5 Data Analysis

For the analysis, we consider each algorithm's best-case by fold, i.e., we consider the highest model performance value obtained by each algorithm. We use the Wilcoxon (Wilcoxon matched-pairs test) statistical test to compare HENBUR with the baseline's algorithms according to the measure (*Macro-F*[1] or *MAP*) (DEMsAR, 2006). We consider p-values = 0.05 as a threshold, and we used the Wilcoxon test with Bonferroni-Holm correction method (de Oliveira Neto *et al.*, 2019).

Also, we use a critical difference diagram to compare model performance. It corresponds to a graphic illustration of the statistical test result (Friedman's test with Nemenyi's post-test) (DEMsAR, 2006).

de Oliveira Neto *et al.* (2019) argues that the effect size measure is the first step in presenting practical significance for a new proposal for industry professionals (de Oliveira Neto *et al.*, 2019). In the context of software engineering, many studies use the $\hat{A}_{12}$ metric by Arcuri and Briand (2011).

In our study, $\hat{A}_{12}(T,B)$ estimates the probability of the T (treatment) algorithm that uses representation based on heterogeneous networks to obtain a better result than the B algorithm (baseline) (VARGHA; DELANEY, 2000). When $\hat{A}_{12}(T,B) = 0.5$, T and B are equivalent. But when $\hat{A}_{12}(T,B) > 0.5$, for example, $\hat{A}_{12}(T,B) = 0.6$ means that the T algorithm is better than the B algorithm 60% of the time.

According to Hess and Kromrey (2004), when $\hat{A}_{12}(T,B) > 0.5$, we define the effect size's magnitude: negligible (N), small (S), medium (M), large (L).

5.3 Results and Analysis

This section presents the experiment results and analysis concerning the research questions.

5.3.1 RQ1 - Practical Significance

In Tables 12, 13, and 14 present the values of $\hat{A}_{12}(T,B)$ for bug report severity prediction, fixer recommendation and bug localization, respectively. In each tables effect size's magnitude in brackets.

Table 12 – Vargha-Delaney effect size for bug report severity prediction.

Product	DT	*k*NN	MLP	RF	SVM-L	MNB
AspectJ	0.7654 (L)	0.5432 (N)	0.8518 (L)	0.8024 (L)	0.9135 (L)	0.6666 (M)
Birt	0.7037 (M)	0.6172 (S)	0.7654 (L)	0.8148 (L)	1 (L)	0.9876 (L)
Eclipse Platform UI	0.8148 (L)	0.5555 (N)	0.8518 (L)	0.9629 (L)	1 (L)	1 (L)
JDT	0.9382 (L)	0.8148 (L)	0.8888 (L)	1 (L)	1 (L)	0.3333 (M)
SWT	0.6419 (S)	0.3950 (S)	0.7901 (L)	0.9259 (L)	0.9876 (L)	1 (L)
Tomcat	1 (L)	0.7654 (L)	0.7654 (L)	0.9259 (L)	0.9629 (L)	0.8395 (L)

Source: Research data.

The Vargha–Delaney $\hat{A}_{12}(T,B)$ effect size shows large or medium differences in 31 out of 36 comparisons (i.e., 86.11% of the cases) in favour of our heterogeneous networks representation for bug report severity prediction (see Table 12).

Table 13 – Vargha-Delaney effect size for fixer recommendation.

Product	DT	*k*NN	MLP	RF	SVM-L	MNB
AspectJ	0.6172 (S)	0.6543 (S)	0.4567 (N)	0.5617 (N)	0.5246 (N)	0.5061 (N)
Birt	0.5308 (N)	0.4691 (N)	0.6296 (S)	0.9382 (L)	1 (L)	0.5185 (N)
Eclipse Platform UI	0.5308 (N)	0.3086 (M)	0.6913 (M)	0.5432 (N)	1 (L)	0.4074 (S)
JDT	0.6666 (M)	0.3950 (S)	0.6543 (S)	0.8271 (L)	1 (L)	0.3209 (M)
SWT	0.6049 (S)	0.5432 (N)	0.5061 (N)	0.9135 (L)	1 (L)	0.3333 (M)
Tomcat	0.5925 (S)	0.5432 (N)	0.6666 (M)	0.5370 (N)	0.5925 (S)	0.5308 (N)

Source: Research data.

The Vargha–Delaney $\hat{A}_{12}(T,B)$ effect size shows large or medium differences in 13 out of 36 comparisons (36.11%) in favor of heterogeneous networks representation for fixer recommendation (see Table 13). Only in 7 out of 36 comparisons (19.44%) the BoW representation obtains greater values than HEN.

Table 14 – Vargha-Delaney effect size for bug localization.

Product	K=1	K=5	K=10
AspectJ	0.6296 (S)	0.6049 (S)	0.5925 (S)
Birt	0.5740 (S)	0.5925 (S)	0.5802 (S)
Eclipse Platform UI	0.5123 (N)	0.5555 (N)	0.5432 (N)
JDT	0.5432 (N)	0.5555 (N)	0.5555 (N)
SWT	0.5308 (N)	0.4938 (N)	0.4938 (N)
Tomcat	0.4938 (N)	0.5555 (N)	0.5061 (N)

Source: Research data.

For bug localization task, we obtained a greater effect size in 15 out of 18 comparisons in favor of heterogeneous networks representation (see Table 14). On the other hand, only in 3 out of 18 comparisons (16.66%) the BoW representation obtains greater values than HEN.

Answer: Our HENBUR framework performs better than traditional monomodal representation (BoW) for three bug report resolution activities.

5.3.2 RQ2 - Bug Report Severity Prediction

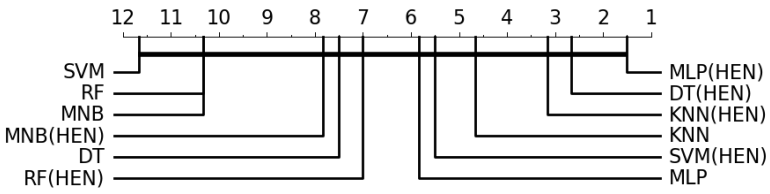
In Tables 15, 16 and 17, the symbol ▲ indicates that HENBUR has the best result, with statistical significance; △ indicates that HENBUR has the best result, without statistical significance; ▼ indicates that HENBUR has the worst result, with statistical significance; ▽ indicates that HENBUR has the worst result, without statistical significance.

The representation based on heterogeneous networks shows higher performance (according to the measure *Macro-F*[1]) than the BoW representation for bug report severity prediction (Table 15). For most projects, the classification algorithms showed statistically higher performance results in the HEN representation. The HEN representation obtained a lower performance for the JDT (MNB classifier) and SWT (KNN classifier) projects.

According to Gomes, Torres and Côrtes (2019), in most studies that used machine learning algorithms to support bug report severity prediction, the MNB classifier with BoW performed better. However, the MLP classifier based on heterogeneous representation networks (MLP(HEN)) performed better in our experiment. This can be seen in Figure 20.

Figure 20 illustrates the average ranking of each algorithm, with the MLP (HEN), DT (HEN), and KNN (HEN) algorithms being in the first, second, and third positions of the ranking, respectively. The SVM algorithm based on BoW is in the last position. Therefore, the algorithms connected by the same line do not present statistically significant differences.

Figure 20 – Critical difference diagram to compare statistical significance between multiple classifiers for bug report severity prediction.



Source: Elaborated by the author.

Answer: HEN representation combined with MLP classifier is the leader ranking for bug report severity prediction, but without statistical significance in our study.

5.3.3 RQ3 - Fixer Recommendation

The classification algorithms based on heterogeneous networks also obtained higher performance (according to the measure *Macro-F*[1]) with the BoW-based algorithms for the

Table 15 – Results of the statistical tests for bug report severity prediction.

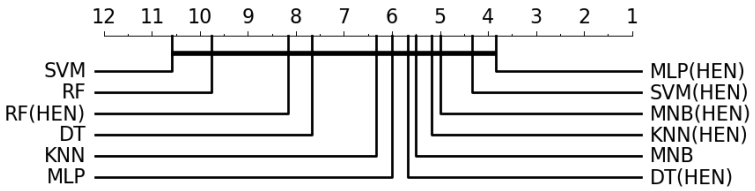
	DT		*k*NN		MLP		RF		SVM-L		MNB	
Project	BoW	HEN	BoW	HEN	BoW	HEN	BoW	HEN	BoW	HEN	BoW	HEN
AspectJ	0.1959	▲ 0.2653	0.2083	△ 0.2124	0.1801	▲ 0.2326	0.1679	▲ 0.2191	0.1196	▲ 0.1799	0.1598	△ 0.1822
Birt	0.2383	▲ 0.2621	0.2391	△ 0.2503	0.2156	▲ 0.2727	0.1383	▲ 0.1543	0.1386	▲ 0.185	0.1388	▲ 0.1884
UI	0.2062	△ 0.2318	0.2382	△ 0.2464	0.2137	▲ 0.2902	0.1476	▲ 0.1664	0.1447	▲ 0.2132	0.1471	▲ 0.2111
JDT	0.1765	▲ 0.2528	0.2129	▲ 0.2474	0.2203	▲ 0.3181	0.1224	▲ 0.1954	0.1101	▲ 0.2631	0.1972	▽ 0.1496
SWT	0.2469	▲ 0.2649	0.2697	▽ 0.2647	0.247	▲ 0.3222	0.1456	▲ 0.1645	0.1449	▲ 0.2753	0.1447	▲ 0.2388
Tomcat	0.1309	▲ 0.2155	0.1713	▲ 0.2259	0.1814	▲ 0.2153	0.1392	▲ 0.2101	0.0998	▲ 0.1726	0.1241	▲ 0.1686

Source: Research data.

fixer recommendation (Table 16). However, the KNN (HEN) had a lower performance than the baseline for most projects.

Figure 21 illustrates the average ranking of each algorithm using the critical difference diagram. The MLP (HEN), SVM (HEN), MNB (HEN), and KNN (HEN) algorithms are in the first, second, third, and fourth positions of the ranking, respectively. However, there are no statistically significant differences between the machine learning algorithms that use the two data representations.

Figure 21 – Critical difference diagram to compare statistical significance between multiple classifiers for fixer recommendation.



Source: Elaborated by the author.

Answer: HEN representation combined with MLP classifier had the best performance for fixer recommendation, but without statistical significance in our study.

5.3.4 RQ4 - Bug Localization

According to Table 17, bug localization (with K = 1) using HEN representation (HEN@1) achieves better performance (according to the measure *MAP*) than BoW representation for all projects. Also, bug localization (with K = 5) using HEN representation (HEN@5) achieves significantly better results for most projects.

When evaluating the contribution factor (α) for each method (see Figure 22), the method using HEN representation obtains a result equal to BoW when α=0. On the other hand, each method has a different performance when α is incremented. For example, the HEN@1 has better performance for UI project when $0 < \alpha < 0.06$.

Answer: The use of HEN representation (with K=1 and K=5) for bug localization had the best in our study.

5.4 Threats to Validity

In this section, we present the threats to the validity of our experiment. The threats identified are discussed concerning four validity types: construct validity, internal validity, external validity, and conclusion validity (WOHLIN *et al.*, 2012).

Table 16 – Results of the statistical tests for fixer recommendation.

	DT		*k*NN		MLP		RF		SVM-L		MNB	
Project	BoW	HEN	BoW	HEN	BoW	HEN	BoW	HEN	BoW	HEN	BoW	HEN
AspectJ	0.1268	△ 0.1323	0.1127	▲ 0.126	0.1255	▽ 0.1137	0.1646	▲ 0.1811	0.161	△ 0.1743	0.1663	△ 0.1713
Birt	0.0329	△ 0.0351	0.045	▽ 0.0439	0.0406	▲ 0.0517	0.0065	▲ 0.0174	0.0027	▲ 0.0429	0.0321	△ 0.0337
UI	0.0065	△ 0.0066	0.0130	▼ 0.0104	0.0075	▲ 0.0098	0.0012	▲ 0.0014	0.0012	▲ 0.0075	0.0037	▽ 0.0028
JDT	0.0248	▲ 0.0267	0.0343	▽ 0.0324	0.0344	△ 0.0381	0.0093	▲ 0.0128	0.0091	▲ 0.0282	0.0174	▽ 0.0144
SWT	0.04065	△ 0.0436	0.04074	△ 0.0419	0.056	△ 0.0584	0.0198	▲ 0.0288	0.0167	▲ 0.0499	0.1447	△ 0.2388
Tomcat	0.1007	▲ 0.106	0.0951	△ 0.1013	0.0977	△ 0.1044	0.0979	△ 0.0987	0.0979	△ 0.1027	0.1241	△ 0.1686

Source: Research data.

Table 17 – Results of the statistical tests for bug localization.

	K=1		K=5		K=10	
Project	BoW	HEN	BoW	HEN	BoW	HEN
AspectJ	0.05857	▲ 0.07015	0.06719	▲ 0.07675	0.07362	▲ 0.08207
Birt	0.04426	▲ 0.04626	0.04793	▲ 0.05001	0.05175	▲ 0.05358
UI	0.03373	△ 0.03453	0.04492	△ 0.04536	0.04925	△ 0.04961
JDT	0.07804	▲ 0.08301	0.08483	▲ 0.08788	0.09182	▲ 0.095
SWT	0.08662	△ 0.0872	0.1158	▽ 0.1157	0.126	▽ 0.1257
Tomcat	0.08813	△ 0.08894	0.096	▲ 0.09779	0.1015	△ 0.1024

Source: Research data.

5.4.1 Construct Validity

Construction Validity regards the relationship between theory and observation, i.e., if the treatment reflects the cause well, the result reflects the effect well (WOHLIN *et al.*, 2012). Our study aims to evaluate the performance of models that support bug report resolution. We chose the *Macro-F*[1] metric as an alternative to compare the bug report severity prediction and fixer recommendation models since the *Macro-F*[1] is recommended for problems that have unbalanced classes (CHOEIKIWONG; VATEEKUL, 2016). When the model wrongly classifies the most frequent class or rarest class, the metrics give the same weight to both.

5.4.2 Internal Validity

Inferences about the causal relationship between treatment and outcome are considered by internal validity (JONSSON *et al.*, 2016; WOHLIN *et al.*, 2012). In our study, all classification algorithms run with the same data pre-processing. Also, we use the default setting for each classifier. Then, no classifier benefited from special pre-processing steps or tuning settings.

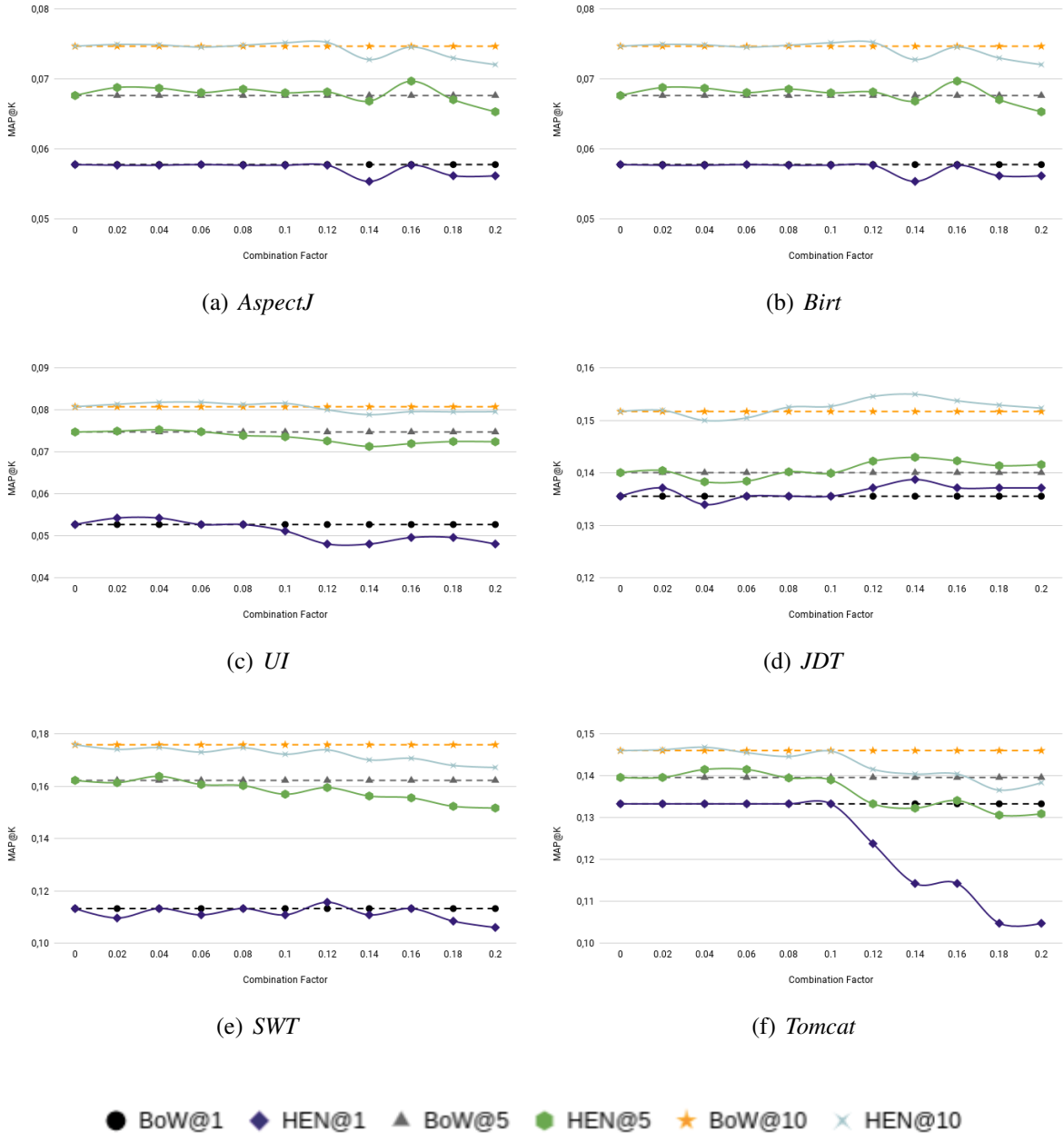
5.4.3 External Validity

External validity is the capacity to generalization our experimental results (WOHLIN *et al.*, 2012). We obtained data from six open source projects. These projects also use GIT as a version control system and Bugzilla as their bug tracking system. However, our study could not be generalized to large-scale industrial settings because we did not use data from proprietary software.

5.4.4 Conclusion Validity

Conclusion validity refers to correct inference about the relationship between treatment and experimental result (WOHLIN *et al.*, 2012). To increase the measure's reliability in our study, we chose only objective measures with the independent variable. Furthermore, we also increase the reliability of treatment implementation because we apply similar treatments (same HEN structure) to all subjects.

Figure 22 – Contributions of each method by the combination factor.



(a) *AspectJ* (b) *Birt*

(c) *UI* (d) *JDT*

(e) *SWT* (f) *Tomcat*

(g) *Legend*

Source: Elaborated by the author.

5.5 Final Remarks

This Chapter proposes a new approach based on a heterogeneous information network to support the bug report resolution, focusing on severity prediction, fixer recommendation, and bug localization. We combine software metrics with bug report metadata and textual data to build a heterogeneous information network.

The results of this quasi-experiment using data from six open source projects indicate

that: (i) HENBUR performs better than traditional monomodal representation (BoW); (ii) HEN representation combined with MLP classifier is the leader ranking for bug report severity prediction and fixer recommendation, but without significance in our study; and (iii) the use of HEN representation (with K=1 and K=5) for bug localization had the best in our study.

The overall conclusion is that HENBUR is a promising alternative for the BRR process's automation. Since when combining different software artifacts, HENBUR extracts valuable knowledge to support software test managers making decisions.

We also plan to extract data from proprietary software repositories to analyze HENBUR results in the industry in future work. Also, we intended to investigate the impact of HEN on other BRR activities, for example, identify the bug type (CATOLINO *et al.*, 2019).

CHAPTER

6

CONCLUSION

Due to the high cost of manual software fault correction, many methodologies, techniques, and tools have been defined and proposed to minimize fault correction costs.

The previous studies investigate the applicability of artificial intelligence techniques to BRR activities' automation strategies. For example, using machine learning algorithms to define predictive models to support bug report resolution activities. However, most studies use software information representations in a monomodal perspective (that is, they use only a software artifact). Also, they do not consider the interdependence between BRR activities.

The research problem addressed in this thesis involves the lack of studies that support the combination of different software artifacts to propose a unified representation that enables the automation of bug report resolution activities.

In the previous chapters are conducted quasi-experiments to investigate the contribution of HEN in this research problem context. Firstly, chapter 3 was validated the bug report representation using HEN (only the detailed descriptions of the bug reports) to support automatic severity bug report prediction. Then, chapter 4 was investigated the impact of HEN representation in a multimodal perspective (bug reports and software metrics) to support bug localization activity. Finally, chapter 5 was analyzed the unified heterogeneous information network (bug reports, software metrics, and bug fixers) to support the automatic bug report resolution activities from a multimodal perspective.

The quasi-experiments results suggest that a heterogeneous information network is a promising alternative to represent software data from different sources. Also, HEN is a promising alternative to support the automatic bug report resolution process from a holistic multimodal perspective.

The main contributions of this thesis are shown in Section 6.1, the limitations and difficulties are discussed in Section 6.2, the possible extensions and future work are presented in Section 6.3, and finally, Section 6.4 presents the data and codes availability statement.

6.1 Thesis Contributions

In this doctoral thesis, we have been the following contributions:

- Proposal for a new use of semi-supervised algorithms based on heterogeneous information network to support the bug report severity prediction (BARBOSA *et al.*, 2017) (more details in Chapter 3). Semi-supervised classifiers obtained competitive results relative to supervised classifiers regardless of the number of labeled examples. Samples with 20% of the label were sufficient to stabilize the performance of most classifiers. This work can help software engineers, for example, when one new software project has few labeled instances, and they use this approach to support decision making.
- Definition of a data representation based on a heterogeneous information network to support the bug localization activity (BARBOSA *et al.*, 2019) (details in Chapter 4). Chapter 4 examined an approach (BULNER) that evaluates the impact of model representation combination (Bag-of-Words with Word Embeddings) and networking regularization to support bug localization. In the quasi-experiment in an open source context, the BULNER with HEN representation and network regularization-based machine learning method obtain a more appropriate representation model. This work improves state of the art by supporting the bug localization from a model representation combination and networking regularization.
- Extending, standardizing, pre-processing, and making available a set of collections of open source bug reports with the corresponding software metrics from impacted source code files after the resolution of the bug reports (details in Chapter 5).
- Definition of a unified data representation based on a heterogeneous information network to support the bug report resolution process (bug report severity prediction, fixer recommendation, and bug localization) (details in Chapter 5). HENBUR represented different software elements and their respective relationship in the same structure. This work improves state of the art by supporting the BRR process from a unified multimodal representation.

Overall, this doctoral thesis demonstrates that heterogeneous information networking can be a promising alternative for the BRR process' automation. Bug reports are essential in the BRR process. As well as other software documents (e.g., source code, software metric) to specific BRR tasks (e.g., bug localization). Then, HEN representation is one alternative to define unified representation to support the BRR process' automation holistically.

6.2 Limitations

In this doctoral thesis, some limitations should be considered.

- **Quasi-experiment**
  Quasi-experiments were conducted restricted to open source projects. Therefore, quasi-experiments' results are not generalized to another context (e.g., closed source project). Then, case studies applied in the software industry might be different findings.

- **Data collection process**
  In quasi-experiments (present in chapters 4 and 5) were reused the connection between bug reports and their associated correction files after the application of Dallmeier and Zimmermann's heuristic by Ye, Bunescu and Liu (2014). Unfortunately, this dataset does not have program spectra (failing and successful execution traces) necessary to evaluate spectrum-based bug localization techniques.

- **Applications to bug report resolution process with heterogeneous information network**
  The findings from quasi-experiments are only proof of concepts that use real software information. There are some issues, for example, that should be investigated the impact of multi-grained (e.g., code token, code statement, code function) nature of programming language and noise in a bug report on the BRR process. Also, these quasi-experiments combine heterogeneous information network representation only with machine learning algorithms from state of the art.

6.3 Possible Extensions and Future Work

Based on the contributions and limitations of this doctoral thesis, we have the following suggestions for future work:

- **Using the dataset closed source project**
  We use only open source projects in all quasi-experiment from this doctoral thesis, then proprietary software product is a future opportunity to investigate the impact on model performance. Some work presents the best model performance when running with a closed source project dataset (CHATURVEDI; SINGH, 2012).

- **Apply other language models**
  The combination of HEN with Long Short Term Memory (LSTM), Convolutional Neural Network (CNN), or Bidirectional Encoder Representations from Transformers (BERT) algorithms should be investigated to support the BRR process. Also, HENBUR should be updated with source code representation (e.g., code2vec (ALON *et al.*, 2019), commit2vec (LOZOYA *et al.*, 2021)), and bug localization activity implemented by HENBUR could be compared with the proposed Deep Multimodal model for Bug Localization (DEMOB) (ZHU *et al.*, 2021).

- **Integrate other BRR tasks**
  Other tasks are related to the three investigated in the BRR process. For example, identifying the bug type (CATOLINO *et al.*, 2019) is a task that could be investigated by reusing the unified data representation of the data based on a heterogeneous information network.

6.4 Data and Codes Availability Statement

Data and source code files that support the findings and contributions of this doctoral thesis are available at:

- <http://sites.labic.icmc.usp.br/ipm/msr-sbrp> (about Chapter 3).
- <https://github.com/jacsonrbinf/bulner> (about Chapter 4).
- <https://github.com/jacsonrbinf/henbur> (about Chapter 5).

BIBLIOGRAPHY

ALON, U.; ZILBERSTEIN, M.; LEVY, O.; YAHAV, E. Code2vec: Learning distributed representations of code. **Proc. ACM Program. Lang.**, ACM, New York, NY, USA, v. 3, n. POPL, p. 40:1–40:29, Jan. 2019. ISSN 2475-1421. Available: <http://doi.acm.org/10.1145/3290353>. Citation on page 99.

ANTONIOL, G.; AYARI, K.; PENTA, M. D.; KHOMH, F.; GUéHéNEUC, Y.-G. Is it a bug or an enhancement? a text-based approach to classify change requests. In: **Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds**. New York, NY, USA: Association for Computing Machinery, 2008. (CASCON '08). ISBN 9781450378826. Available: <https://doi.org/10.1145/1463788.1463819>. Citation on page 44.

ANVIK, J.; HIEW, L.; MURPHY, G. C. Who should fix this bug? In: **Proceedings of the 28th International Conference on Software Engineering**. New York, NY, USA: ACM, 2006. (ICSE '06), p. 361–370. ISBN 1-59593-375-1. Available: <http://doi.acm.org.ez49.periodicos.capes.gov.br/10.1145/1134285.1134336>. Citation on page 45.

Apache Software Foundation. **Apache Jira Repository**. [S.l.]: Apache Jira Web Page, 2021. <https://issues.apache.org/jira/>. Citation on page 52.

Arcuri, A.; Briand, L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: **2011 33rd International Conference on Software Engineering (ICSE)**. [S.l.: s.n.], 2011. p. 1–10. ISSN 1558-1225. Citation on page 87.

Atlassian. **Jira Software**. [S.l.]: Project Web Page, 2021. <https://www.atlassian.com/software/jira>. Citation on page 49.

BARBOSA, J. R.; MARCACINI, R. M.; BRITTO, R.; SOARES, F.; REZENDE, S.; VINCENZI, A. M.; DELAMARO, M. E. Bulner: Bug localization with word embeddings and network regularization. In: **Proceedings of the VII Workshop on Software Visualization, Evolution and Maintenance (VEM '19)**. [S.l.: s.n.], 2019. Citations on pages 30, 67, 69, 73, 74, 75, and 98.

BARBOSA, J. R.; MATSUNO, I. P.; GUIMARÃES, E. R.; REZENDE, S. O.; VINCENZI, A. M.; DELAMARO, M. E. Mineração de textos para apoiar a predição de severidade de relatórios de incidentes: um estudo de viabilidade. In: **XVI Simpósio Brasileiro de Qualidade de Software (SBQS '17)**. [S.l.: s.n.], 2017. Citations on pages 30 and 98.

BREIMAN, L. Random forests. **Machine Learning**, v. 45, n. 1, p. 5–32, Oct. 2001. ISSN 1573-0565. Available: <https://doi.org/10.1023/A:1010933404324>. Citation on page 44.

CALEFATO, F.; LANUBILE, F.; MAIORANO, F.; NOVIELLI, N. Sentiment polarity detection for software development. **Empirical Software Engineering**, v. 23, n. 3, p. 1352–1382, Jun. 2018. ISSN 1573-7616. Available: <https://doi.org/10.1007/s10664-017-9546-9>. Citation on page 44.

CATOLINO, G.; PALOMBA, F.; ZAIDMAN, A.; FERRUCCI, F. Not all bugs are the same: Understanding, characterizing, and classifying bug types. **Journal of Systems and Software**, v. 152, p. 165 – 181, 2019. ISSN 0164-1212. Available: <http://www.sciencedirect.com/science/article/pii/S0164121219300536>. Citations on pages 95 and 100.

CHANG, S.; HAN, W.; TANG, J.; QI, G.-J.; AGGARWAL, C. C.; HUANG, T. S. Heterogeneous network embedding via deep architectures. In: **Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2015. (KDD '15), p. 119–128. ISBN 9781450336642. Available: <https://doi.org/10.1145/2783258.2783296>. Citation on page 40.

CHAPELLE, O.; SCHLKOPF, B.; ZIEN, A. **Semi-Supervised Learning**. 1st. ed. [S.l.]: The MIT Press, 2006. ISBN 0262514125, 9780262514125. Citations on pages 33, 34, and 35.

CHATURVEDI, K.; SINGH, V. An empirical comparison of machine learning techniques in predicting the bug severity of open and closed source projects. **International Journal of Open Source Software and Processes**, v. 4, n. 2, p. 32 – 59, 2012. ISSN 19423926. Available: <http://dx.doi.org/10.4018/jossp.2013040103>. Citation on page 99.

CHOEIKIWONG, T.; VATEEKUL, P. Improve accuracy of defect severity categorization using semi-supervised approach on imbalanced data sets. In: **Proceedings of the International MultiConference of Engineers and Computer Scientists**. [S.l.: s.n.], 2016. v. 1. Citation on page 93.

Cui, P.; Wang, X.; Pei, J.; Zhu, W. A survey on network embedding. **IEEE Transactions on Knowledge and Data Engineering**, v. 31, n. 5, p. 833–852, May 2019. ISSN 1041-4347. Citations on pages 39 and 40.

DALLMEIER, V.; ZIMMERMANN, T. Extraction of bug localization benchmarks from history. In: **Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2007. (ASE '07), p. 433–436. ISBN 9781595938824. Available: <https://doi.org/10.1145/1321631.1321702>. Citation on page 87.

de Oliveira Neto, F. G.; TORKAR, R.; FELDT, R.; GREN, L.; FURIA, C. A.; HUANG, Z. Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. **Journal of Systems and Software**, v. 156, p. 246 – 267, 2019. ISSN 0164-1212. Available: <http://www.sciencedirect.com/science/article/pii/S0164121219301451>. Citation on page 87.

DELALLEAU, O.; BENGIO, Y.; ROUX, N. L. Efficient non-parametric function induction in semi-supervised learning. In: **Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics**. Society for Artificial Intelligence and Statistics, 2005. p. 96–103. Available: <https://www.microsoft.com/en-us/research/publication/efficient-non-parametric-function-induction-in-semi-supervised-learning/>. Citation on page 36.

DEMsAR, J. Statistical comparisons of classifiers over multiple data sets. **J. Mach. Learn. Res.**, JMLR.org, v. 7, p. 1–30, Dec. 2006. ISSN 1532-4435. Available: <http://dl-acm-org.ez49.periodicos.capes.gov.br/citation.cfm?id=1248547.1248548>. Citations on pages 58, 62, 63, and 87.

Eclipse Foundation. **Bugzilla – Bug 4023**. [S.l.]: Eclipse Web Page, 2021. <https://bugs.eclipse.org/bugs/show_bug.cgi?id=4023>. Citation on page 41.

EFSTATHIOU, V.; CHATZILENAS, C.; SPINELLIS, D. Word embeddings for the software engineering domain. In: **Proceedings of the 15th International Conference on Mining Software Repositories**. New York, NY, USA: ACM, 2018. (MSR '18), p. 38–41. ISBN 978-1-4503-5716-6. Available: <http://doi-acm-org.ez67.periodicos.capes.gov.br/10.1145/3196398.3196448>. Citations on pages 70 and 81.

FRANK, E.; HALL, M. A.; WITTEN, I. H. **The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"**. 4. ed. [S.l.]: Morgan Kaufmann, 2016. Citation on page 57.

GARCIA, H. V.; SHIHAB, E. Characterizing and predicting blocking bugs in open source projects. In: **Proceedings of the 11th Working Conference on Mining Software Repositories**. New York, NY, USA: Association for Computing Machinery, 2014. (MSR 2014), p. 72–81. ISBN 9781450328630. Available: <https://doi.org/10.1145/2597073.2597099>. Citations on pages 43 and 47.

GARCIA, S.; HERRERA, F. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. **Journal of Machine Learning Research**, v. 9, n. 12, p. 2677–2694, 2008. Citations on pages 58 and 63.

GERS, F.; SCHMIDHUBER, J.; CUMMINS, F. Learning to forget: continual prediction with lstm. In: **1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)**. [S.l.: s.n.], 1999. v. 2, p. 850–855 vol.2. ISSN 0537-9989. Citation on page 44.

GOMES, L. A. F.; TORRES, R. da S.; CôRTES, M. L. Bug report severity level prediction in open source software: A survey and research opportunities. **Information and Software Technology**, 2019. ISSN 0950-5849. Available: <http://www.sciencedirect.com/science/article/pii/S0950584919301648>. Citations on pages 50 and 89.

GÓMEZ, O.; OKTABA, H.; PIATTINI, M.; GARCÍA, F. A systematic review measurement in software engineering: State-of-the-art in measures. In: FILIPE, J.; SHISHKOV, B.; HELFERT, M. (Ed.). **Software and Data Technologies**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 165–176. ISBN 978-3-540-70621-2. Citation on page 80.

HARRIS, Z. S. Distributional structure. **Word**, Routledge, v. 10, n. 2-3, p. 146–162, 1954. Available: <https://doi.org/10.1080/00437956.1954.11659520>. Citations on pages 81 and 82.

HESS, M.; KROMREY, J. D. Robust confidence intervals for effect sizes: A comparative study of cohen's d and cliff's delta under non-normality and heterogeneous variances. In: . [S.l.: s.n.], 2004. Citation on page 87.

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. **Science**, American Association for the Advancement of Science, v. 313, n. 5786, p. 504–507, 2006. ISSN 0036-8075. Available: <https://science.sciencemag.org/content/313/5786/504>. Citation on page 46.

Hoang, T.; Oentaryo, R. J.; Le, T. B.; Lo, D. Network-clustered multi-modal bug localization. **IEEE Transactions on Software Engineering**, v. 45, n. 10, p. 1002–1023, Oct 2019. ISSN 1939-3520. Citations on pages 77 and 78.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, Nov. 1997. ISSN 0899-7667. Available: <https://doi.org/10.1162/neco.1997.9.8.1735>. Citation on page 44.

HUO, X.; LI, M. Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code. In: **Proceedings of the 26th International Joint Conference on Artificial Intelligence**. [S.l.]: AAAI Press, 2017. (IJCAI'17), p. 1909–1915. ISBN 9780999241103. Citation on page 47.

HUO, X.; LI, M.; ZHOU, Z.-H. Learning unified features from natural and programming languages for locating buggy source code. In: **Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence**. [S.l.]: AAAI Press, 2016. (IJCAI'16), p. 1606–1612. ISBN 9781577357704. Citation on page 47.

ISO/IEC/IEEE. **ISO/IEC/IEEE 29119-1:2013 – Software and systems engineering – Software testing – Part 1: Concepts and definitions**. 2013. Citation on page 40.

JI, M.; SUN, Y.; DANILEVSKY, M.; HAN, J.; GAO, J. Graph regularized transductive classification on heterogeneous information networks. In: BALCÁZAR, J. L.; BONCHI, F.; GIONIS, A.; SEBAG, M. (Ed.). **Machine Learning and Knowledge Discovery in Databases**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 570–586. ISBN 978-3-642-15880-3. Citations on pages 36, 70, 73, and 82.

JONSSON, L.; BORG, M.; BROMAN, D.; SANDAHL, K.; ELDH, S.; RUNESON, P. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. **Empirical Software Engineering**, v. 21, n. 4, p. 1533–1578, Aug. 2016. ISSN 1573-7616. Available: <https://doi.org/10.1007/s10664-015-9401-9>. Citations on pages 45 and 93.

JUNG, W.; LEE, E.; WU, C. A survey on mining software repositories. **IEICE Transactions on Information and Systems**, E95.D, n. 5, p. 1384–1406, 2012. Citation on page 50.

KIBRIYA, A. M.; FRANK, E.; PFAHRINGER, B.; HOLMES, G. Multinomial naive bayes for text categorization revisited. In: WEBB, G. I.; YU, X. (Ed.). **AI 2004: Advances in Artificial Intelligence**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 488–499. ISBN 978-3-540-30549-1. Citations on pages 34 and 44.

KIM, D.; TAO, Y.; KIM, S.; ZELLER, A. Where should we fix this bug? a two-phase recommendation model. **IEEE Transactions on Software Engineering**, v. 39, n. 11, p. 1597–1610, Nov 2013. ISSN 0098-5589. Citation on page 46.

KOCH, K.-R. Bayes' theorem. In: **Bayesian Inference with Geodetic Applications**. [S.l.]: Springer, 1990. p. 4–8. Citation on page 34.

LAM, A. N.; NGUYEN, A. T.; NGUYEN, H. A.; NGUYEN, T. N. Combining deep learning with information retrieval to localize buggy files for bug reports. In: **2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.: s.n.], 2015. p. 476–481. Citation on page 84.

____. Bug localization with combination of deep learning and information retrieval. In: **2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)**. [S.l.: s.n.], 2017. p. 218–229. Citations on pages 27, 46, 47, 77, and 84.

LAMKANFI, A.; DEMEYER, S.; GIGER, E.; GOETHALS, B. Predicting the severity of a reported bug. In: **2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)**. [S.l.: s.n.], 2010. p. 1–10. ISSN 2160-1852. Citations on pages 27, 43, 47, 49, and 77.

LAMKANFI, A.; DEMEYER, S.; SOETENS, Q. D.; VERDONCK, T. Comparing mining algorithms for predicting the severity of a reported bug. In: **2011 15th European Conference on Software Maintenance and Reengineering**. [S.l.: s.n.], 2011. p. 249–258. ISSN 1534-5351. Citations on pages 44 and 51.

LAMKANFI, A.; PÉREZ, J.; DEMEYER, S. The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information. In: IEEE PRESS. **Proceedings of the 10th Working Conference on Mining Software Repositories**. [S.l.], 2013. p. 203–206. Citations on pages 52 and 64.

LANTZ, B. **Machine Learning with R**. [S.l.]: Packt Publishing, 2013. ISBN 978-1782162148. Citation on page 33.

LE, T.-D. B.; THUNG, F.; LO, D. Will this localization tool be effective for this bug? mitigating the impact of unreliability of information retrieval based bug localization tools. **Empirical Software Engineering**, v. 22, n. 4, p. 2237–2279, Aug 2017. ISSN 1573-7616. Available: <https://doi.org/10.1007/s10664-016-9484-y>. Citation on page 67.

LEE, S.-R.; HEO, M.-J.; LEE, C.-G.; KIM, M.; JEONG, G. Applying deep learning based automatic bug triager to industrial projects. In: **Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2017. (ESEC/FSE 2017), p. 926–931. ISBN 9781450351058. Available: <https://doi.org/10.1145/3106237.3117776>. Citations on pages 45, 46, and 47.

LOZOYA, R. C.; BAUMANN, A.; SABETTA, A.; BEZZI, M. Commit2vec: Learning distributed representations of code changes. **SN Computer Science**, v. 2, n. 3, p. 150, Mar. 2021. ISSN 2661-8907. Available: <https://doi.org/10.1007/s42979-021-00566-z>. Citation on page 99.

MAATEN, L. Van der; HINTON, G. Visualizing data using t-sne. **Journal of machine learning research**, v. 9, n. 11, 2008. Citation on page 40.

MANI, S.; SANKARAN, A.; ARALIKATTE, R. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In: **Proceedings of the ACM India Joint International Conference on Data Science and Management of Data**. New York, NY, USA: Association for Computing Machinery, 2019. (CoDS-COMAD '19), p. 171–179. ISBN 9781450362078. Available: <https://doi.org/10.1145/3297001.3297023>. Citations on pages 45, 46, 47, and 77.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. **Introduction to Information Retrieval**. [S.l.]: Cambridge University Press, 2008. Citations on pages 39 and 46.

MantisBT Development Team. **MantisBT 2.0 – Admin Guide**. [S.l.]: Project Web page, 2021. <https://www.mantisbt.org/docs/master/en-US/Admin_Guide/Admin_Guide.pdf>. Citation on page 49.

Menzies, T.; Marcus, A. Automated severity assessment of software defect reports. In: **2008 IEEE International Conference on Software Maintenance**. [S.l.: s.n.], 2008. p. 346–355. Citation on page 43.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013. Citations on pages 81 and 82.

MIKOLOV, T.; SUTSKEVER, I.; CHEN, K.; CORRADO, G. S.; DEAN, J. Distributed representations of words and phrases and their compositionality. In: **Advances in Neural Information Processing Systems 26**. Curran Associates, Inc., 2013. p. 3111–3119. Available: <http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>. Citation on page 68.

MOREIRA, J.; CARVALHO, A.; HORVATH, T. **A General Introduction to Data Analytics**. John Wiley Sons, Ltd, 2018. ISBN 978-1-119-29626-3. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119296294.fmatter>. Citation on page 38.

Mozilla Foundation. **Bugzilla – Bug tracking system**. [S.l.]: Project Web Page, 2021. <https://www.bugzilla.org/>. Citation on page 49.

NEYSIANI, B. S.; BABAMIR, S. M.; ARITSUGI, M. Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems. **Information and Software Technology**, p. 106344, 2020. ISSN 0950-5849. Available: <http://www.sciencedirect.com/science/article/pii/S0950584920301117>. Citation on page 49.

NGUYEN, A. T.; NGUYEN, T. T.; AL-KOFAHI, J.; NGUYEN, H. V.; NGUYEN, T. N. A topic-based approach for narrowing the search space of buggy files from a bug report. In: **2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)**. [S.l.: s.n.], 2011. p. 263–272. ISSN 1938-4300. Citation on page 46.

NIGAM, K.; MCCALLUM, A. K.; THRUN, S.; MITCHELL, T. Text classification from labeled and unlabeled documents using EM. **Machine Learning**, Kluwer Academic Publishers, v. 39, n. 2-3, p. 103–134, 2000. ISSN 0885-6125. Citation on page 38.

PHAM, V.; BLUCHE, T.; KERMORVANT, C.; LOURADOUR, J. Dropout improves recurrent neural networks for handwriting recognition. In: **2014 14th International Conference on Frontiers in Handwriting Recognition**. [S.l.: s.n.], 2014. p. 285–290. ISSN 2167-6445. Citation on page 45.

POLISETTY, S.; MIRANSKYY, A.; BAsAR, A. On usefulness of the deep-learning-based bug localization models to practitioners. In: **Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2019. (PROMISE'19), p. 16–25. ISBN 9781450372336. Available: <https://doi.org/10.1145/3345629.3345632>. Citation on page 84.

QUINLAN, J. R. **C4.5:Programs for Machine Learning**. [S.l.]: M.Kaufmann, 1993. Citation on page 34.

RAHMAN, M. M.; ROY, C. K. Improving ir-based bug localization with context-aware query reformulation. In: **Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering**. New York, NY, USA: ACM, 2018. (ESEC/FSE 2018), p. 621–632. ISBN 978-1-4503-5573-5. Available: <http://doi-acm-org.ez49.periodicos.capes.gov.br/10.1145/3236024.3236065>. Citation on page 67.

Ramay, W. Y.; Umer, Q.; Yin, X. C.; Zhu, C.; Illahi, I. Deep neural network-based severity prediction of bug reports. **IEEE Access**, v. 7, p. 46846–46857, 2019. ISSN 2169-3536. Citations on pages 44, 47, 49, and 77.

RISH, I. An empirical study of the naive bayes classifier. In: IBM NEW YORK. **IJCAI-Workshop Empirical Methods in Artificial Intelligence**. [S.l.], 2001. v. 3, n. 22, p. 41–46. Citation on page 34.

ROBERTSON, S.; ZARAGOZA, H.; TAYLOR, M. Simple bm25 extension to multiple weighted fields. In: **Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management**. New York, NY, USA: Association for Computing Machinery, 2004. (CIKM '04), p. 42–49. ISBN 1581138741. Available: <https://doi.org/10.1145/1031171.1031181>. Citation on page 43.

ROSSI, R. G. **Classificação automática de textos por meio de aprendizado de máquina baseado em redes**. Phd Thesis (PhD Thesis) — Universidade de São Paulo, 2015. Citations on pages 27, 35, 36, and 54.

ROSSI, R. G.; LOPES, A. A.; FALEIROS, T. de P.; REZENDE, S. O. Inductive model generation for text classification using a bipartite heterogeneous network. **Journal of Computer Science and Technology**, Springer, v. 3, n. 29, p. 361–375, 2014. Citation on page 57.

ROSSI, R. G.; LOPES, A. A.; REZENDE, S. O. A parameter-free label propagation algorithm using bipartite heterogeneous networks for text classification. In: **Proceedings of the 29th Annual ACM Symposium on Applied Computing**. New York, NY, USA: Association for Computing Machinery, 2014. (SAC '14), p. 79–84. ISBN 9781450324694. Available: <https://doi.org/10.1145/2554850.2554901>. Citations on pages 37, 54, 70, and 78.

ROSSI, R. G.; LOPES, A. de A.; REZENDE, S. O. Optimization and label propagation in bipartite heterogeneous networks to improve transductive classification of texts. **Information Processing & Management**, v. 52, n. 2, p. 217 – 257, 2016. ISSN 0306-4573. Available: <http://www.sciencedirect.com/science/article/pii/S0306457315000990>. Citations on pages 37, 39, 53, 54, 57, 78, and 86.

SAHA, R. K.; LAWALL, J.; KHURSHID, S.; PERRY, D. E. Are these bugs really normal? In: **2015 IEEE/ACM 12th Working Conference on Mining Software Repositories**. [S.l.: s.n.], 2015. p. 258–268. ISSN 2160-1852. Citation on page 42.

Saha, R. K.; Lease, M.; Khurshid, S.; Perry, D. E. Improving bug localization using structured information retrieval. In: **2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.: s.n.], 2013. p. 345–355. Citations on pages 46 and 47.

Shi, C.; Li, Y.; Zhang, J.; Sun, Y.; Yu, P. S. A survey of heterogeneous information network analysis. **IEEE Transactions on Knowledge and Data Engineering**, v. 29, n. 1, p. 17–37, Jan 2017. Citation on page 78.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. **Information Processing Management**, v. 45, n. 4, p. 427 – 437, 2009. ISSN 0306-4573. Available: <http://www.sciencedirect.com/science/article/pii/S0306457309000259>. Citations on pages 55 and 85.

SUN, C.; LO, D.; KHOO, S.-C.; JIANG, J. Towards more accurate retrieval of duplicate bug reports. In: **2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)**. [S.l.: s.n.], 2011. p. 253–262. ISSN 1938-4300. Citation on page 43.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introduction to Data Mining**. [S.l.]: Addison-Wesley, 2005. ISBN 0-321-32136-7. Citations on pages 51 and 53.

TAN, Y.; XU, S.; WANG, Z.; ZHANG, T.; XU, Z.; LUO, X. Bug severity prediction using question-and-answer pairs from stack overflow. **Journal of Systems and Software**, v. 165, p. 110567, 2020. ISSN 0164-1212. Available: <http://www.sciencedirect.com/science/article/pii/S0164121220300480>. Citations on pages 44, 45, and 67.

THUNG, F.; LE, T.-D. B.; KOCHHAR, P. S.; LO, D. Buglocalizer: Integrated tool support for bug localization. In: **Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering**. New York, NY, USA: ACM, 2014. (FSE 2014), p. 767–770. ISBN 978-1-4503-3056-5. Available: <http://doi-acm-org.ez49.periodicos.capes.gov.br/10.1145/2635868.2661678>. Citation on page 67.

TIAN, Y.; LO, D.; SUN, C. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In: **19th Working Conference on Reverse Engineering**. [S.l.: s.n.], 2012. p. 215–224. ISSN 1095-1350. Citation on page 43.

VAPNIK, V. N. **The Nature of Statistical Learning Theory**. [S.l.]: Springer-Verlag New York, Inc., 1995. Citation on page 34.

VARGHA, A.; DELANEY, H. D. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. **Journal of Educational and Behavioral Statistics**, v. 25, n. 2, p. 101–132, 2000. Available: <https://doi.org/10.3102/10769986025002101>. Citation on page 87.

WANG, B.; XU, L.; YAN, M.; LIU, C.; LIU, L. Multi-dimension convolutional neural network for bug localization. **IEEE Transactions on Services Computing**, p. 1–1, 2020. ISSN 1939-1374. Citations on pages 27 and 46.

WANG, S.; LO, D. Version history, similar report, and structure: Putting them together for improved bug localization. In: **Proceedings of the 22Nd International Conference on Program Comprehension**. New York, NY, USA: ACM, 2014. (ICPC 2014), p. 53–63. ISBN 978-1-4503-2879-1. Available: <http://doi.acm.org/10.1145/2597008.2597148>. Citation on page 46.

Wen, M.; Wu, R.; Cheung, S. Locus: Locating bugs from software changes. In: **2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.: s.n.], 2016. p. 262–273. Citation on page 46.

WOHLIN, C.; RUNESON, P.; HöST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering**. Norwell, MA, USA: Springer Berlin Heidelberg, 2012. ISBN 978-3-642-29043-5. Citations on pages 52, 72, 83, 84, 86, 91, and 93.

Wong, C.; Xiong, Y.; Zhang, H.; Hao, D.; Zhang, L.; Mei, H. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In: **2014 IEEE International Conference on Software Maintenance and Evolution**. [S.l.: s.n.], 2014. p. 181–190. ISSN 1063-6773. Citation on page 46.

XI, S.; YAO, Y.; XIAO, X.; XU, F.; LU, J. An effective approach for routing the bug reports to the right fixers. In: **Proceedings of the Tenth Asia-Pacific Symposium on Internetware**. New York, NY, USA: Association for Computing Machinery, 2018. (Internetware '18). ISBN 9781450365901. Available: <https://doi.org/10.1145/3275219.3275228>. Citation on page 45.

Xia, X.; Lo, D.; Ding, Y.; Al-Kofahi, J. M.; Nguyen, T. N.; Wang, X. Improving automated bug triaging with specialized topic model. **IEEE Transactions on Software Engineering**, v. 43, n. 3, p. 272–297, March 2017. ISSN 1939-3520. Citation on page 45.

XIA, X.; LO, D.; SHIHAB, E.; WANG, X.; YANG, X. Elblocker: Predicting blocking bugs with ensemble imbalance learning. **Information and Software Technology**, v. 61, p. 93 – 106, 2015. ISSN 0950-5849. Available: <http://www.sciencedirect.com/science/article/pii/S0950584914002602>. Citations on pages 27, 42, and 49.

XIAO, G.; DU, X.; SUI, Y.; YUE, T. Hindbr: Heterogeneous information network based duplicate bug report prediction. In: **2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)**. [S.l.: s.n.], 2020. p. 195–206. ISSN 2332-6549. Citation on page 28.

YAO, L.; MAO, C.; LUO, Y. Graph convolutional networks for text classification. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 33, n. 01, p. 7370–7377, Jul. 2019. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/4725>. Citation on page 46.

YE, X.; BUNESCU, R.; LIU, C. Learning to rank relevant files for bug reports using domain knowledge. In: **Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering**. New York, NY, USA: ACM, 2014. (FSE 2014), p. 689–699. ISBN 978-1-4503-3056-5. Available: <http://doi.acm.org/10.1145/2635868.2635874>. Citations on pages 19, 46, 47, 72, 74, 84, 86, 87, and 99.

____. Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. **IEEE Transactions on Software Engineering**, v. 42, n. 4, p. 379–402, April 2016. ISSN 0098-5589. Citation on page 46.

Ye, X.; Shen, H.; Ma, X.; Bunescu, R.; Liu, C. From word embeddings to document similarities for improved information retrieval in software engineering. In: **2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)**. [S.l.: s.n.], 2016. p. 404–415. ISSN 1558-1225. Citations on pages 68, 69, 73, and 84.

YIN, Z.; LI, R.; MEI, Q.; HAN, J. Exploring social tagging graph for web object classification. In: **Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: ACM, 2009. (KDD '09), p. 957–966. ISBN 978-1-60558-495-9. Citation on page 37.

YU, L.; LIU, H. Efficient feature selection via analysis of relevance and redundancy. **The Journal of Machine Learning Research**, JMLR. org, v. 5, p. 1205–1224, 2004. Citation on page 27.

YUGOSHI, I. P. M. **Mineração de opiniões baseada em aspectos para revisões de produtos e serviços**. Phd Thesis (PhD Thesis) — Universidade de São Paulo, 2018. Citations on pages 33, 36, 38, 50, and 54.

ZAIDI, S. F. A.; LEE, C.-G. Learning graph representation of bug reports to triage bugs using graph convolution network. In: **2021 International Conference on Information Networking (ICOIN)**. [S.l.: s.n.], 2021. p. 504–507. ISSN 1976-7684. Citations on pages 45 and 46.

ZARAGOZA, H.; CRASWELL, N.; TAYLOR, M.; SARIA, S.; ROBERTSON, S. Microsoft cambridge at trec-13: Web and hard tracks. In: **IN PROCEEDINGS OF TREC 2004**. [S.l.: s.n.], 2004. Citation on page 43.

ZHANG, T.; CHEN, J.; YANG, G.; LEE, B.; LUO, X. Towards more accurate severity prediction and fixer recommendation of software bugs. **Journal of Systems and Software**, v. 117, p. 166 – 184, 2016. ISSN 0164-1212. Available: <http://www.sciencedirect.com/science/article/pii/S0164121216000765>. Citations on pages 28, 42, 43, and 47.

ZHANG, T.; JIANG, H.; LUO, X.; CHAN, A. T. A literature review of research in bug resolution: Tasks, challenges and future directions. **The Computer Journal**, Oxford University Press, v. 59, n. 5, p. 741–773, 2016. Citations on pages 42 and 44.

ZHOU, D.; BOUSQUET, O.; LAL, T. N.; WESTON, J.; SCHöLKOPF, B. Learning with local and global consistency. In: **Proceedings of the 16th International Conference on Neural Information Processing Systems**. Cambridge, MA, USA: MIT Press, 2003. (NIPS'03), p. 321–328. Citations on pages 36 and 54.

ZHOU, J.; ZHANG, H.; LO, D. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: **2012 34th International Conference on Software Engineering (ICSE)**. [S.l.: s.n.], 2012. p. 14–24. ISSN 0270-5257. Citations on pages 46, 47, 68, and 73.

ZHOU, Y.; TONG, Y.; GU, R.; GALL, H. Combining text mining and data mining for bug report classification. **Journal of Software: Evolution and Process**, v. 28, n. 3, p. 150–176, 2016. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1770>. Citations on pages 28, 44, 47, and 77.

ZHU, X.; GHAHRAMANI, Z.; LAFFERTY, J. Semi-supervised learning using gaussian fields and harmonic functions. In: **Proc. of the Int. Conf. on Machine Learning**. [S.l.]: AAAI, 2003. p. 912–919. Citations on pages 37 and 70.

ZHU, X.; GOLDBERG, A. B. **Introduction to semi-supervised learning**. [S.l.]: Morgan and Claypool Publishers, 2009. ISBN 1598295470, 9781598295474. Citations on pages 33, 34, and 50.

ZHU, X. J. **Semi-supervised learning literature survey**. [S.l.], 2005. Citation on page 36.

ZHU, Z.; LI, Y.; WANG, Y.; WANG, Y.; TONG, H. A deep multimodal model for bug localization. **Data Mining and Knowledge Discovery**, Apr. 2021. ISSN 1573-756X. Available: <https://doi.org/10.1007/s10618-021-00755-7>. Citations on pages 27, 46, and 99.

ČUBRANIć, D. Automatic bug triage using text categorization. In: **In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering**. [S.l.]: KSI Press, 2004. p. 92–97. Citations on pages 27 and 77.

APPENDIX

A

SOFTWARE METRICS

Table 18 presents software metrics used in two quasi-experiment (chapter 4 and chapter 5). We extracted these metrics from Understand™ tools.

Table 18 – Software metrics used.

Name	**Software Metric**	**Metric Type**
AvgCyclomatic	Average Cyclomatic Complexity	Complexity
AvgCyclomaticModified	Average Modified Cyclomatic Complexity	Complexity
AvgCyclomaticStrict	Average Strict Cyclomatic Complexity	Complexity
AvgEssential	Average Essential Cyclomatic Complexity	Complexity
AvgLine	Average Number of Lines	Volume
AvgLineBlank	Average Number of Blank Lines	Volume
AvgLineCode	Average Number of Lines of Code	Volume
AvgLineComment	Average Number of Lines with Comments	Volume
CountDeclClass	Number of Classes	Object Oriented
CountDeclClassMethod	Number of Class Methods	Object Oriented
CountDeclClassVariable	Number of class variables	Object Oriented
CountDeclExecutableUnit	Number of Executable Unit	Volume
CountDeclFunction	Number of functions	Volume
CountDeclInstanceMethod	Number of instance methods	Object Oriented
CountDeclInstanceVariable	Number of instance methods	Object Oriented
CountDeclMethod	Number of local methods	Object Oriented
CountDeclMethodDefault	Number of local default methods	Object Oriented
CountDeclMethodPrivate	Number of local private methods	Object Oriented
CountDeclMethodProtected	Number of protected methods	Object Oriented
CountDeclMethodPublic	Number of public methods	Object Oriented
CountLine	Physical Lines	Volume

CountLineBlank	Blank Lines of Code	Volume
CountLineCode	Source Lines of Code	Volume
CountLineCodeDecl	Declarative Lines of Code	Volume
CountLineCodeExe	Executable Lines of Code	Volume
CountLineComment	Lines with Comments	Volume
CountSemicolon	Semicolons	Volume
CountStmt	Statements	Volume
CountStmtDecl	Declarative Statements	Volume
CountStmtExe	Executable Statements	Volume
MaxCyclomatic	Max Cyclomatic Complexity	Complexity
MaxCyclomaticModified	Max Modified Cyclomatic Complexity	Complexity
MaxCyclomaticStrict	Max Strict Cyclomatic Complexity	Complexity
MaxEssential	Max Essential Complexity	Complexity
MaxNesting	Nesting	Complexity
RatioCommentToCode	Comment to Code Ratio	Volume
SumCyclomatic	Sum Cyclomatic Complexity	Complexity
SumCyclomaticModified	Sum Modified Cyclomatic Complexity	Complexity
SumCyclomaticStrict	Sum Strict Cyclomatic Complexity	Complexity
SumEssential	Sum Essential Complexity	Complexity


ICMC
USP
SÃO CARLOS