USING MACHINE LEARNING ALGORITHMS FOR CLASSIFYING NON-FUNCTIONAL REQUIREMENTS - RESEARCH AND EVALUATION

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2021

Manal Khalid M Binkhonain

Department of Computer Science

Contents

Word Count: 68,361

List of Tables

List of Figures

Abstract

Requirements classification, the process of assigning requirements to classes, is essential to requirements engineering, as it serves to define and organize the requirements for application systems, to determine the boundaries of the systems, to establish the relationships among the requirements, and to ensure the correct kinds of functionality are implemented in the systems. As most requirements are written in natural language, the manual classification of textual requirements can be time consuming and error prone. Aiming to reduce the burden on the human analyst, the machine-learning (ML) approach has been used since the early 2000s for automatic requirements classification. The ML approach faces three problems in non-functional requirements (NFRs) classification: imbalanced classes, short text, and the high dimensionality of feature space. Although these problems are widely addressed in various classification tasks, they are less frequently considered in requirements classification.

In this thesis, we present two ML methods for automatically classifying NFRs. The main novelty of these methods lies in applying techniques that address the classification problems mentioned earlier. The first method integrates three techniques—dataset decomposition, semantic role-based feature selection, and feature extension—to address the three problems. The second method addresses short-text classification by adding the most similar requirements (i.e., the requirement extension technique). Both methods were evaluated on a publicly available NFRs dataset. The results of each method are compared with related methods, baseline methods, and state-of-the-art solutions to the problems. The results demonstrate the usefulness of addressing problems with NFR classifications and the effectiveness of the proposed methods, suggesting that these solutions could improve different requirements classification tasks.

To assess the generalization of the results of the proposed methods, we present a case study on the use of ML methods in sub-class NFRs classification. In particular, we reapply the proposed methods for classifying usability requirements according to usability aspects. This study includes the identification of the most common aspects of usability by systematically reviewing existing usability models. It also includes building usability requirements datasets. The results of applying ML methods in classifying usability requirements are similar to those provided by NFRs, confirming the usefulness of addressing problems with requirements classification.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

List of Publications

- Binkhonain, Manal & Zhao, L.. (2019). A Review of Machine Learning Algorithms for Identification and Classification of Non-Functional Requirements. Expert Systems with Applications.
- Binkhonain, Manal & Zhao, L.. Dealing with Imbalanced Class, Short Text and High Dimensionality Problems in Machine Learning-Based Requirements Classification: Method Development and Evaluation. Under Review.

List Of Abbreviations

RE	Requirements Engineering
NFR	Non-functional Requirements
FR	Functional Requirements
UR	Usability Requirements
ML	Machine learning
NL	Natural Language
NLP	Natural Language processing
SVM	Support Vector Machine
DT	Decision Tree
NB	Naïve Bayes
KNN	k-Nearest Neighbors
POS	Part-Of-Speech tagging
NER	Named Entity Recognition
TF-IDF	Term Frequency–Inverse Document Frequency

Acknowledgements

Doing a PhD is hard, especially for a mom of two who started her PhD with a newborn baby and ended it with the Coronavirus pandemic, and much in between. Without the encouragement and support of others, this would be impossible. I would like to thank everyone who helped and encouraged me to make it possible.

The first and deepest acknowledgement goes to my supervisor, Dr Liping Zhao, one of the kindest people I have ever met. Liping is always available, helpful, and encouraging. I often feel like she is a part of my family as a supportive and an advisor. Thank you Liping for all the roles you have played. Beyond the thesis subject, I have learned a lot from working with her; most importantly is that when I work on a task, I should perform it in a perfect manner, no matter how important the task is. Her comments, ideas, and actions were lessons for me to help me become a better person and to improve my research. I will always appreciate what I have learned from you. Thank you, Liping.

I am very grateful to know Dr Sarah Clinch (my co-supervisor) and to have been her student. In addition to her kind personality, Sarah is a supportive person who is always around when I need her and is available to help when I ask. Thank you, Sarah.

My thanks and appreciation go to my examiners Prof. Peter Sawyer and Dr Lucas Cordeiro for their valuable comments and suggestions, which were helpful to improve this thesis and my knowledge in the RE field.

Special gratitude goes to my sponsor, King Saud University, and the Saudi Cultural Bureau in London for their financial support and continuous help during my research. A special thanks goes to Arwa AlAmoudi at King Saud University for providing me with the requirements I needed for building my dataset.

Around me there are a lot of people who are willing to support me without ever being paid for it. A big thank you to those who took part in validating the datasets used during my research, even those that are not reported in this thesis. Thank you Amal Alsuwaidan for annotating a big dataset professionally and honestly. I really appreciated your midnight calls to understand the tasks and improve the quality of the dataset. Thank you Dr Areeb Alowisheq for your help; I still remember that when I asked you, you came to my office to take the papers and annotate them quickly and professionally. Thank you Hadeel Alshabanat, Asma Alotaibi, Dr Waad Alhoshan, Dr Seetah Alsalamah, Ruba Alassaf, Dr Alaa Alahmadi, Dr Deemah Alqahtani, Dr Mariam Alqasab, and Haifa Alrdahi for your support when I needed and for all the good memories I have with you; you are very kind people that anyone would be grateful to meet.

My small family is my biggest source of love, happiness, and support. My deepest thanks go to my husband, Mohamed Alsmari, for joining me on this journey, giving me everything I needed without asking, and understanding my stress and busyness. Thank you Mohammed for your unconditional and genuine love that made the completion of this thesis possible. My princess (Deemah) and my superhero (Abdulaziz) made this journey full of happiness and activity. Thank you for your pure love, interesting discussions, amazing pictures, beautiful cards, and much more. Your laughs were life-giving; playing with you was the most enjoyable time in my day and seeing your sleepy, small faces was the best ending to each day. Even during lockdown, you were full of energy and love, always saying "do not worry mommy" and cuddling me with your small arms when you saw me in a bad mood. You did a lot on this journey; I love you most Deemah and Abdulaziz.

I would not be who or where I am today without my parents. A big thanks to you Khalid and Munnerah Binkhonain for your continued support and encouragement. I really appreciate each call, unexpected shipment, prayer, and message. My sisters (Maha and Hyay) and my brothers (Mohammed, Faisal, Rashed, and Abdullah) were also sources of strong encouragement who were always there for care and support. Thank you for everything, including visiting me during your holidays and taking care of my children when I was extremely busy. I am lucky to have you, always.

I have friends who have become a part of my family. I would like to thank my lifelong friends Shatha Bin Dawood and Amani Aldous for everything, including caring, calling, always being a shoulder to cry on, and making a hard time easier. I am very lucky to be your friend.

The final acknowledgment goes to everyone I met during this journey, who made big or small differences, giving me everything from nice smiles in the corridors to useful discussions anywhere. Thank you to my officemates, all Saudi girls, and all of the staff in the Kilburn building.

Chapter 1

Introduction

"The first step in solving a problem is to recognize that it does exist."

Zig Ziglar

1.1 Research Context and Motivation

Developing any software system starts with the requirements engineering (RE) phase, the phase that deals with requirements, including specifying and validating user needs and constraints. It is well known that early phase RE commonly deals with informal requirements specifications [Yu97, NL03] and such documents are usually written in natural language (NL) [Rya93, AM81, LMP04, KNL14]. In spite of the flexibility and ease of understanding NL documents [DFFP18], such documents pose real problems inherent in NL (e.g., ambiguity, inaccuracy, inconsistency, and lack of structure [WRH97, TB13]). To overcome such problems, informal requirements specifications need to be processed/analyzed before they can be used in design and development. A common approach to processing requirements is via requirements classification [ZAF$^+$21].

Requirements classification can be defined as the process of assigning predefined categories to requirements based on their content [FML05]. The primary purpose of requirements classification, like classifications in other fields [MB02], is to reduce the diversity of the requirements into understandable, manageable groups of related requirements, before further analysis and processing are possible. Typically, requirements are classified into functional versus non-functional classes (categories), with the latter further divided into various subcategories, such as usability, reliability, security, etc. [Bro15]. Although there is no consensus in the RE community on what non-functional requirements (NFRs) are and how they should be classified [Gli05, CdPL09, Bro15, EVF16], requirements classification remains to be a critical analytic and organizational task in RE, as a recent survey shows [ZAF+21].

In recent years, the machine learning (ML) approach to text classification [Seb02] has gained widespread recognition, due to its effectiveness, efficiency, and portability [Seb02]. In particular, the ML approach has achieved visible successes in a large number of real-world applications, from fake news detection [AMPG20], to opinion mining [JHS09], spam email filtering [DBC+19], document categorization and retrieval [KBLK10], and medical diagnosis [SGSG19], just to name a few. At the same time, developing the ML approach for requirements classification is also gathering pace, with several research efforts being reported in the RE literature (e.g., [CHSZS07, ZYWS11, JMJ16, DDAÇ19, AKG+17, HKO08, CGC10, KM17]). Typically, since most requirements are written in NL documents, the ML approach to requirements classification is similar to that for text classification [Seb02]: To classify a set of requirements, we need to build a *text classifier* first. This involves training a ML algorithm by feeding to it a set of requirements called a *training dataset*, where each requirement in the set has already been pre-assigned a category by human experts. The trained ML algorithm, known as the *text classifier* or *classifier*, is then subject to validation and testing, to improve its classification accuracy and to ensure it will perform and generalize well to new requirements. Afterwards, the classifier can be used to classify new requirements, by automatically assigning each requirement to a category.

However, due to the nature of requirements text, ML-based requirements classification suffers from three major learning problems. The first one is *imbalanced classes* (also known as class imbalanced learning) [HG09, GFB+11], referring to uneven distributions of instances among classes. This problem is prevalent in requirements classification, as requirements categories are naturally imbalanced, with an uneven distribution of the requirements among different categories. For example, the functional requirements (FRs) are often the largest category; the distribution of requirements in different non-functional requirements (NFR) categories is also uneven [CHSZS07, EVF16]. The fundamental issue with this problem is that imbalanced categories can significantly compromise the performance of most standard learning algorithms because such algorithms assume or expect balanced class distributions [HG09], and when given the imbalanced classes, these algorithms will not be able to properly recognize the distributive characteristics of these classes. Furthermore, some algorithms, such as support vector machines (SVMs), are inherently biased toward the majority classes in order to minimize the high error rates of mis-classification, as such classes are more prevalent than minor classes [HG09]. Such biased classification can be dangerous in real-world applications. For example, in medical diagnosis, rare diseases are often present in minor classes, and biased classification can lead to misdiagnosis of such diseases [ASR$^+$15, Wei04, LLS09].

The second problem is *short-text* classification. This problem arises because guidelines and advice on writing requirements often advocate that requirements should be brief and concise [Mey93]. For example, IEEE guidance for developing system requirements specifications [Com98] states that a well-formed requirement is "a necessary, short, definitive statement of need". Wilson [Wil99] also suggests that an effective requirement writing style is the one that uses "short declarative sentences" and "simple sentence structures". It was calculated that the average length of a FR is about 20 words, whereas the average length of a NFR is between 14 and 28 words [KM17]. The fundamental issue with the short-text classification is *feature sparsity* [SHA12, AG19], i.e., lack of relevant features - distinct words or terms - in a dataset. Feature sparsity can significantly affect the performance of most standard learning algorithms because such algorithms perform classification by establishing a correlation between features and categories [AG19].

Third, requirements classification also suffers from a common text classification problem known as *high dimensional feature space*, [PP14, LJ98]. This problem is concerned with the presence of a large number of features in text data [IKT05], which can easily reach the tens of thousands in a single text document, and most of them are irrelevant to the classification task. Irrelevant features are the noise, which can mislead standard ML algorithms and reduce the accuracy of text classifiers [Seb02, LY05, AZ12, DLWZ19]. Taking a NFR statement provided by Glinz [Gli07] as an example, “The system shall prevent any unauthorized access to the customer data,” only three words (features) in this statement (i.e., prevent, unauthorized, and access) are relevant to classification, whereas the remaining words are irrelevant.

Clearly, these three problems are of high importance to ML-based requirements classification and have wide-ranging implications on the real-world applications of automatic requirements classification. However, due to the relatively young age of the application of ML in RE, to date, most proposed ML-based requirements classification methods have only focused on the high dimensionality problem [JMJ16, DDAÇ19, HKO08], with relatively few methods have considered addressing the imbalanced classes and short text problems [KM17, LL17].

This research focuses on investigating and addressing issues related to the autonomic classification of NFRs in particular for many reasons. First, considering that these requirements contribute to driving the architectural design, they need to be defined early [CHSZS07]. Addressing NFRs at a later stage may lead to an increase in the probability of project failure and an escalation in the cost of software development [HKO08, SW13, ROW13]. Second, the difficulty of identifying NFRs as they are spread throughout different parts of the requirements documents (since most requirements specifications are organized by functionality)[CHSZS07]. In addition, the confliction, dependency, and interaction among different NFRs further complicate the classification task [BDCD19]. Third, the identification and classification of NFRs are important for successful development by, for example, improving the quality of requirements by reviewing them with particular experts (e.g., reviewing security requirements with security experts) and helping in matching developers and tasks (e.g., performance and maintainability should be dealt with by developers with different expertise) [BDCD19, RR12].

1.2 Research Aim and Questions

The research presented in this thesis is motivated by a desire to investigate the impact of the three aforementioned problems (i.e., high dimensionality, imbalance class, short text classification) in NFRs classification. In doing so, we propose new, potential solutions to handle these problems and explore the impact of addressing these problems in different NFRs classification tasks.

In order to achieve this aim, the research seeks to answer the following research questions:

RQ1: What are the current methods for handling the three aforementioned problems in the context of NFRs classification? And where are the research gaps?

RQ2: What kinds of (new) solutions would be suitable to address these problems in NFRs classification?

RQ3: How effective and efficient are those solutions in comparison with the baseline and the state-of-the-art methods?

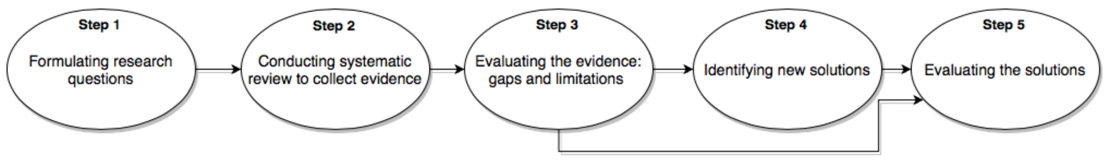

Step 1
Formulating research questions
Step 2
Conducting systematic review to collect evidence
Step 3
Evaluating the evidence: gaps and limitations
Step 4
Identifying new solutions
Step 5
Evaluating the solutions

Figure 1.1: Research methodology, which follows the evidence-based software engineering approach [DKJ05]. The arrows between steps mean that the findings of the first step are used by the other.

RQ4: Can the results of the proposed solutions be generalized to several NFRs classification tasks (e.g. detecting sub-classes of NFRs)?

Additional secondary questions are provided through the thesis, where these questions are introduced and handled separately within a single chapter. The secondary research questions help to construct and develop the potential solutions the main research questions aimed to explore and investigate.

The first part of the research questions (RQ1 & RQ2) is for understanding the existing literature on using ML for NFRs classification, which is addressed in the first part of this thesis. The remaining research questions (RQ3 &RQ4) are about developing and evaluating the solutions, and they are addressed in the second part of this thesis. This ensures the following research objectives are fulfilled:

1. To conduct a systematic literature review on the use of ML in NFRs classification.
2. To develop new potential solutions for addressing the NFRs classification problems.
3. To demonstrate the impact of these solutions against related works and similar solutions/methods proposed within the current literature.
4. To further evaluate the effectiveness of the proposed solutions by examining them across several NFRs classification tasks.

1.3 Research Methodology

Five research steps (see Figure 1.1) are followed in order to fulfill the aim of this thesis and address the research questions identified in the previous section. These steps are based on the evidence-based software engineering approach (EBSE), [DKJ05] which contains specifying research questions, collecting evidence, evaluating the evidence, identifying new solutions, and evaluating the solutions.

Step 1: specifying a set of answerable questions that describe the need for information or the problem. This step includes formulating the research questions mentioned in the previous section. These questions are determined after our reading in ML with text classification and NFRs characteristics.

Step 2: conducting a review of the literature to find the best available evidence. This including conducting a systematic review to understand how ML is used and evaluated in NFRs classification. Moreover, this step involves reviewing other classification tasks to understand how the problems are commonly addressed in different classification tasks.

Step 3: critically evaluating the collected evidence in terms of applicability, impact, and validity. This includes deeply analyzing ML methods used in NFRs classification, identifying gaps, and highlighting limitations in these methods. In particular, this step involves focusing on the three classification problems by assessing the attempts to address these problems in NFRs classification and comparing them with those used in other classification tasks.

Step 4: combining the evaluated evidence to identify a new solution or determine best practices. Based on the analysis in the previous step, we identified new solutions to address the three problems in NFRs classification. In particular, we developed two methods known as ML4RC (machine learning for requirement classification) and SE4RC (semantic extension for requirement classification). The first method addresses the three problems using three different techniques (feature selection for high dimensionality, data decomposition for imbalanced class, and feature extension for short text classification). The second method addresses the short text classification using a requirements expansion technique.

Step 5: evaluating the performance resulting from the proposed solution and looking for ways to improve it. The proposed solutions are evaluated in two different classification tasks. The first task is to detect and classify NFRs, while the second one is to classify NFRs into further sub-classes. In the second task, we classified usability requirements according to usability attributes (e.g., efficiency, effectiveness, and satisfaction).

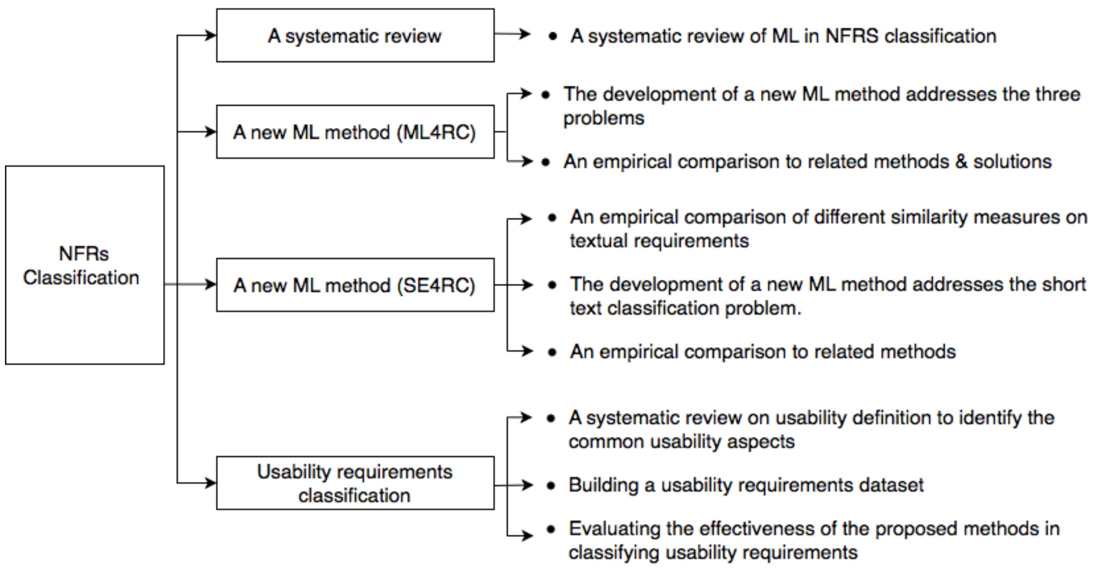

NFRs Classification
A systematic review
A new ML method (ML4RC)
A new ML method (SE4RC)
Usability requirements classification
A systematic review of ML in NFRS classification
The development of a new ML method addresses the three problems
An empirical comparison to related methods & solutions
An empirical comparison of different similarity measures on textual requirements
The development of a new ML method addresses the short text classification problem.
An empirical comparison to related methods
A systematic review on usability definition to identify the common usability aspects
Building a usability requirements dataset
Evaluating the effectiveness of the proposed methods in classifying usability requirements

Figure 1.2: Summary of the contributions

1.4 Research Contributions

The work described in this thesis makes four major contributions (see Figure 1.2), as follows.

Conducted a systematic review of the use of ML in NFRs classification. This review provides a better understanding of the field, including technical details of how ML classifiers are built and evaluated. It also includes a report on the general limitations of the reviewed literature and discusses the open challenges of using ML in NFR classification. The review's results helped us to analyze the most common classification problems (impacts and solutions) and evaluate the impacts of potential solutions. Part of the results of this review was published in [BZ19].

Developed and evaluated a new ML method for addressing the three classification problems. We have designed, implemented, and evaluated a ML method, Machine Learning for Requirements Classification (*ML4RC*). This method includes three key techniques, each of which addresses a single problem: *data decomposition*, for class imbalance problems; *semantic role-based* feature selection, for high dimensionality; and *feature extension*, for short text classification. To assess the effectiveness of ML4RC, we conducted two main studies:

- An empirical comparison of the entire ML4RC method with closely related methods.

- An evaluation study to compare the performance of ML4RC's three key techniques individually against a baseline method and related solutions (techniques).

These studies showed that ML4RC performs better than its peers, and the three techniques also achieve better classification results than a baseline method and common related solutions.

Developed and evaluated an ML method for addressing the short-text classification problem. Previous evaluation studies revealed that addressing short-text classification using the feature extension technique is a promising approach. For this reason, and due to the lack of application of such a technique in NFR classification, we proposed a new solution for handling this problem, Semantic Expansion for Requirements Classification (*SE4RC*). This technique expanded requirements with the most semantically similar requirements (not words, as with ML4RC). Two main evaluation studies have been conducted related to this contribution:

- An empirical performance evaluation of nine semantics-based similarity measures among NFR
- An empirical comparison of the SE4RC against baseline and two related methods

The first study was conducted in light of the scant attention paid to the semantic similarity measure among NFR, while the second study aimed to assess the performance of SE4RC. The findings of the first study (e.g., the most effective measure) were applied in SE4RC. The second study showed that SE4RC outperforms other methods, confirming the effectiveness of feature expansions in NFR classification. In addition, based on an in-depth analysis of the two studies' results, we highlighted a significant difficulty in measuring the similarity among NFR, since, for example, such requirements could have both implicit and explicit information. Such information not only hinders the process of automatically measuring similarities among requirements, but also manual measurements conducted by non-experts (e.g., requirements analysts).

Evaluated the proposed methods within the context of usability requirements (UR) classification. The systematic review (the first contribution) showed a lack of focus for analyzing UR, compared with other types of NFR (i.e., security). Thus, to further evaluate the proposed solutions, we applied them to the classification of UR according to usability aspects. In an initial step to achieve this, we conducted another systematic review to identify the common usability aspects and deeply analyzed the interrelationships among them. Then, we built and validated a UR dataset that was collected from different resources and application domains. The dataset was used to train and evaluate the proposed solutions (ML4RC and SE4RC). The evaluation study included a comparison of the performance of the proposed solution with baseline methods. The results were similar to those obtained by applying the methods in NFR classification, thus we concluded that the proposed solutions would be effective for other requirements/text classification tasks.

1.5 Thesis Structure

The overall structure of the thesis is illustrated in Figure 1.3 and outlined below.

Chapter 1 introduces an overview of the research problem, objectives, research questions, and contributions.

Chapter 2 establishes the necessary background to the work presented in this thesis. This chapter is divided into two parts: RE and text classification using ML. Each part describes relevant techniques and concepts, with emphasis on those adapted in the thesis.

Chapter 3 reports a systematic review of the related work (ML methods in NFRs classification), in which findings, limitations, and open challenges are discussed. Appendix A provides a comprehensive overview of the studies included in the review and data sources (tables) used to draw figures in this chapter.

Chapter 4 illustrates the three classification problems. For each problem, this chapter identifies the problem, briefly describes common solutions, and reviews the solution in NFRS classification using the studies identified in the previous chapter. Following this, it discusses limitations and gaps in the solutions and highlights the proposed methods' motivations. Appendix B tables and descriptions of preliminary experiments mentioned in this chapter.

Chapter 5 presents a proposed ML method (ML4RC) and the description of a series of experiments conducted to evaluate this method. This chapter presents these experiments' results and discusses the main findings, limitations, and strengths of that method.

Chapter 6 describes a proposed method (SE4RC). Since this method relies on the similarity between requirements, it provides an overview of text semantic similarity and requirements similarity. This chapter includes two main experimental studies. The first study is about measuring semantic similarity between requirements, including

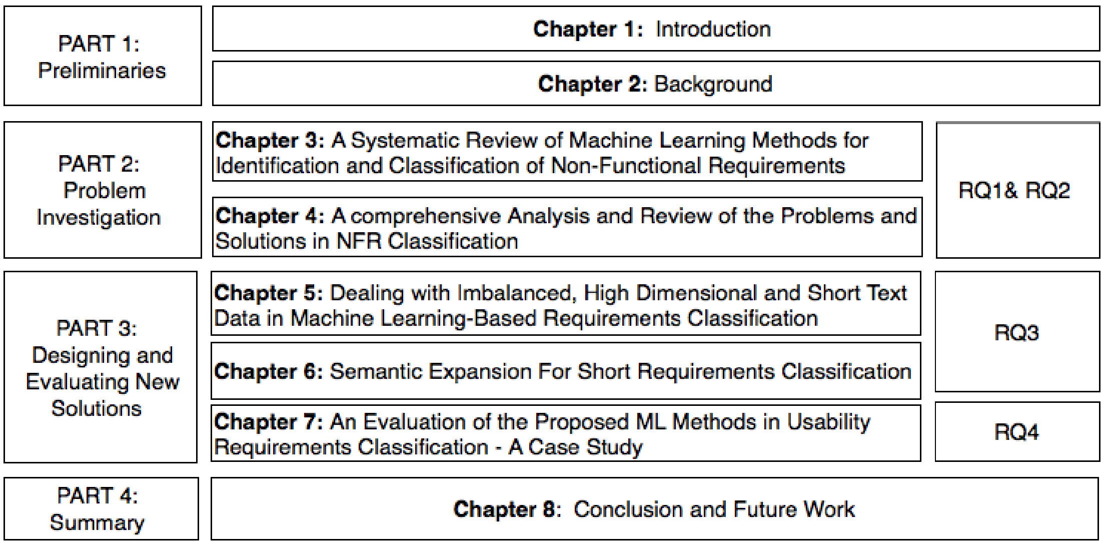

PART 1: Preliminaries
Chapter 1: Introduction
Chapter 2: Background
PART 2: Problem Investigation
Chapter 3: A Systematic Review of Machine Learning Methods for Identification and Classification of Non-Functional Requirements
Chapter 4: A comprehensive Analysis and Review of the Problems and Solutions in NFR Classification
RQ1& RQ2
PART 3: Designing and Evaluating New Solutions
Chapter 5: Dealing with Imbalanced, High Dimensional and Short Text Data in Machine Learning-Based Requirements Classification
Chapter 6: Semantic Expansion For Short Requirements Classification
RQ3
Chapter 7: An Evaluation of the Proposed ML Methods in Usability Requirements Classification - A Case Study
RQ4
PART 4: Summary
Chapter 8: Conclusion and Future Work

Figure 1.3: Thesis structure

an empirical study comparing different semantic similarity methods to measure the similarity between NFRs. The second study is about using requirements similarity in expanding requirements for NFRs classification (SE4RC), including assessing the performance of SE4RC in relation to baseline method and related work. Finally, the chapter discusses the findings, highlighting limitations and strengths.

Chapter 7 presents a case study of the use of ML methods (Chapter 5 and Chapter 6) in URs classification (i.e., classify URs according to usability aspects). Due to the lack of available URs models or datasets, this chapter uses the results of a systematic review on usability (reported in Appendix C) to identify common usability aspects and build the URs dataset. Moreover, the chapter provides some necessary usability background, case study methodology, results, and lessons learned.

Chapter 8 presents the conclusions from the research, including addressing the research questions, reviewing the findings of the thesis, and discussing directions for future work.

Chapter 2

Background

"Student: Dr. Einstein, aren't these the same questions as last year's final exam?
Dr. Einstein: Yes; But this year the answers are different."

Albert Einstein

This chapter provides a foundation for understanding the key areas of the thesis: RE and ML. Each of which is illustrated in a separate section—Section 2.1 describes system requirements, NFRs, and RE processes, and Section 2.2 provides an overview of the use of ML in text classification: building, evaluation, and analysis.

2.1 Requirement Engineering

The general process to develop a software system (i.e., system development life cycle) starts with requirements, moving into system design, implementation, testing, deployment, and maintenance. The first phase, which deals with requirements, is known as RE. This section provides an overview of the RE phase, starting with requirements definition, NFRs description, and RE concept and processes.

2.1.1 Definition of a Requirement

Requirements, which are the basis for every project, define what a potential new system should be or should do to satisfy stakeholders' needs (users, customers, suppliers, developers, businesses) [AS02, Lap17, HJD11]. Requirements are widely expressed in NL; however, they can have different representations (e.g., formal, mathematically rigorous, specification) based on stakeholder needs [Lap17]. Although NL is flexible and easy to understand, it raises challenges to capture the problem and user's needs completely and unambiguously [YDRG$^+$11].

The quality of requirements plays a critical role in a project's success. Since such requirements are addressed in the initial step for defining problem scope and control project activities, each subsequent development information is connected with requirements [AS02, HJD11]. Low-quality requirements lead to late delivery, go over budget, and consequently, project failure [AS02]. Nasir and Sahibuddin in 2011 [NS11] conducted a comprehensive study to analyze the critical success factors of different project sizes, domains in multiple countries. By analyzing 43 case studies, they found that within a list of 26 factors, "clear requirements and specifications" is the first critical factor identified in 26 studies (60.5%). Another survey conducted by the Standish Group in 2014 [Cla14] showed that a "Clear Statement of Requirements" is the third reason for project success, whereas "Incomplete Requirements & Specifications" is the second factor of a project to be challenged (over-budget, over the time estimate) and first factor for a project to be impaired and canceled.

Several criteria are used to characterize "a good requirement". Examples of these criteria are completeness—fully described requirements with no missing elements; unambiguous—having a clear and single understanding; consistency —no mismatch or contradict among requirements; traceability —linking requirements to system artifacts (models, tests, and code) in both forwards and backward direction, and testability–being quantified by providing a basis for a pass/fail test for an end product [Boe00, HJD11].

2.1.2 Non-Functional Requirements

Requirements can be either functional—specifying what a system must do, or non-functional—describing how well a system will do it [CdPL09]. Although the definitions may seem straightforward, there are many ways of understanding them, often leading to overlapping—when NFRs add functionality to a system. For example, security requirements, which ensure only specific users can alert data, may represent functionality. However, it is needed for a non-functional reason (i.e., security) [RR12]. According to Glinz [Gli07], the notion of NFR is representation-dependent; our classification of a requirement depends on the way we represent it. For example, Table 2.1 shows different representations provided by Glinz for a requirement that means "only

Requirement	**Type**
The system shall prevent any unauthorized access to the customer data	NFR
The probability for successful access to the customer data by an unauthorized person shall be smaller than 10-5	NFR
The database shall grant access to the customer data only to those users that have been authorized by their user name and password	FR

Table 2.1: Examples of related requirements having different requirements types, based on their representation. These requirements were provided by Glinz to shows the notion of NFRs is representation-dependent [Gli07].

authorized users can access a specific function or data". Although these requirements are related, the first and second requirements are NFRs, while the third is functional. Robertson and Robertson [RR12] distinguished NFR by their relation to a system's essential functions—not altering these functions, but adding properties to them.

NFRs are referred to by different terminology, including soft goals, quality requirements, constraints, extra FRs, and non-behavioral requirements [CNYM12]. Moreover, they have been characterized and classified differently in the literature. For example, ISO/IEC 9126 [II04], one of the most widespread quality standards [BBC$^+$04], describes and measures a software system's quality. This standard distinguishes external and internal quality; external quality measures the quality of executable software in testing, while internal for measuring non-executable software during designing and coding. There are six external and internal quality characteristics, which are further divided into sub-characteristics and related metrics (see Figure 2.2). The ISO/IEC 9126 standard also introduces the so-called quality in use: user's view of quality, including effectiveness, productivity, safety, and satisfaction.

Another example of NFRs classification is provided by ISO/IEC 25010 [fSEC$^+$11] (Figure 2.2), which consists of two parts. The first part is the software product quality model composed of eight characteristics (functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability), which are further subdivided into sub-characteristics. The second part is the system quality in use model composed of five characteristics (effectiveness, efficiency, satisfaction, freedom of risk, and context coverage).

Mairiza et al. [MZN10] conducted a systematic analysis of the extant literature of NFRs, including the analysis of NFRs types. They found that there are 252 types of NFRs, including quality attributes such (e.g., performance and reliability), external

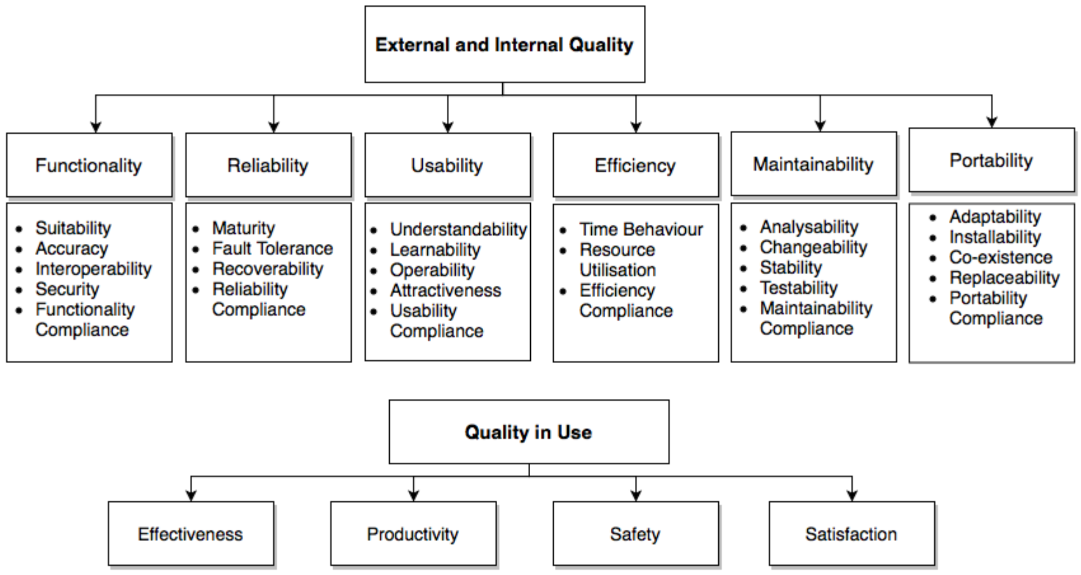


Figure 2.1: Quality characteristics of software product in ISO/IEC 9126 [II04]

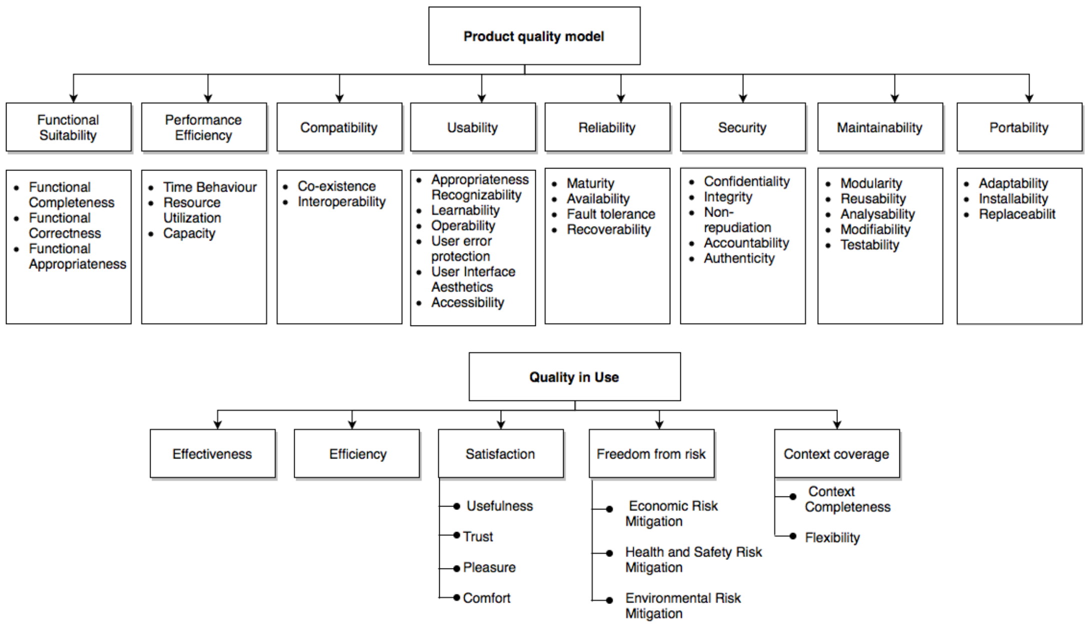


Figure 2.2: Systems and software Quality Requirements by ISO/IEC 25010 [fSEC$^+$11]

interfaces requirements (look & feel and user interface), development constraints (timing and cost), business rules (e.g., production life span), and others (e.g., cultural and environmental). In addition, their analysis includes identifying the five most frequent NFRs types: performance, reliability, usability, security, and maintainability. Table 2.2 provides detailed definitions of these types.

The literature indicates that errors related to NFRs are among the most expensive and difficult to correct errors [CY04]. Ineffective handling of NFRs does not only affect the system under development but also the development process itself [KS96, SGR09]. NFRs are contributed to driving architectural design, detecting risks, and scheduling needs [KO14, CHSZS07]. Therefore, inappropriate handling of these requirements can lead to several potential problems, including increase maintenance costs, considerable delays in delivering a system, users' dissatisfaction, or, in the worst case, project failure [MCN92, SGR09, MZN10]. An example of projects that fail due to a lack of NFRs is the U.S. Army intelligence sharing application. This application has cost $2.7 billions, but due to performance and usability issues, it failed [Hos11]. Electronic health record (EHR) systems have also failed due to a lack of usability [BSA10]. London Ambulance System was deactivated after its deployment for different reasons, including neglecting NFRs during the system developments (e.g., reliability, usability, and performance) [FD96].

Early addressing of requirements in RE help to avoid the aforementioned problems [CY04]. The classification of NFRs by their type is useful in the early analysis for different reasons [RR12]. First, grouping requirements by type helps in discovering conflict, missed, and duplicated requirements. Second, understanding a type of requirement helps to write an appropriate fit criterion, which improves the quality of the requirement. Third, identifying requirements type helps in specifying stakeholder involvement, by, for example, reviewing security requirements with a security expert.

2.1.3 Requirement Engineering Concept and Processes

The early dealing with requirements (i.e., RE) is crucial for software success [NE00]. It is argued that the term "engineering" was added to "requirements" to emphasize that dealing with requirements is an important part of each engineering process, and RE is a subset of system engineering in general, not just software engineering [NE00, HJD11]. RE has been considered among the most important phases of a system development life cycle[CBAB12]. Because errors identified in this phase are less costly to fix—may only require a quick discussion and a small amount of editing [AS02]. However, errors

NFRs	**Definition**	**Attributes**
Performance	Requirements that determine software's ability to provide appropriate performance using required resources and under specific conditions.	response time, space, capacity, latency, throughput, computation, execution speed, transit delay, workload, resource utilization, memory usage, accuracy, efficiency compliance, modes, delay, miss rates, data loss, concurrent transaction processing
Reliability	Requirements that specify software capability to operate without failure during a given time and with specified normal conditions.	completeness, accuracy, consistency, availability, integrity, correctness, maturity, fault tolerance, recoverability, reliability, compliance, failure rate/critical failure
Usability	Requirements that specify the interaction between system and end-users, including the efforts required to learn, operate, and interpret the system's outputs.	learnability, understandability, operability, attractiveness, usability compliance, ease of use, human engineering, user friendliness, memorability, efficiency, user productivity, usefulness, likeability, user reaction time
Security	Requirements that prevent and protect a system from, for example, unauthorized access to the system.	confidentiality, integrity, availability, access control, authentication
Maintainability	Requirements that describe a software's ability to be modified by, for example, correcting defects, upgrading, or changing the software.	testability, understandability, modifiability, analyzability, changeability, stability, maintainability compliance

Table 2.2: The most frequent NFRs identified by Mairiza et al. [MZN10]

discovered later are more costly to be fixed as many other subsequent decisions have been made depending on RE phase [CBAB12]. Thus, the successful development of a system is critically related to RE [NE00].

RE is a series of activities that help understanding system capabilities and attributes, starting from recognizing a problem to be solved to a detailed specification of the problem for both the stakeholders and developers involved in the system development [WB13, Som05]. Figure 2.3 shows RE processes (also known as tasks or activities): elicitation/discovery, analysis and reconciliation, documentation/representation, verification and validation, and requirements management [Lap17]. The first four activities are classified under requirement development, aiming to identify, capture, document, and confirm product requirements [WB13]. The last one is requirement management which deals with changes in requirements during a system development [WB13].

Requirement elicitation. This activity is conducted to discover requirements of the system to be developed [Som05]. The information on these requirements comes from

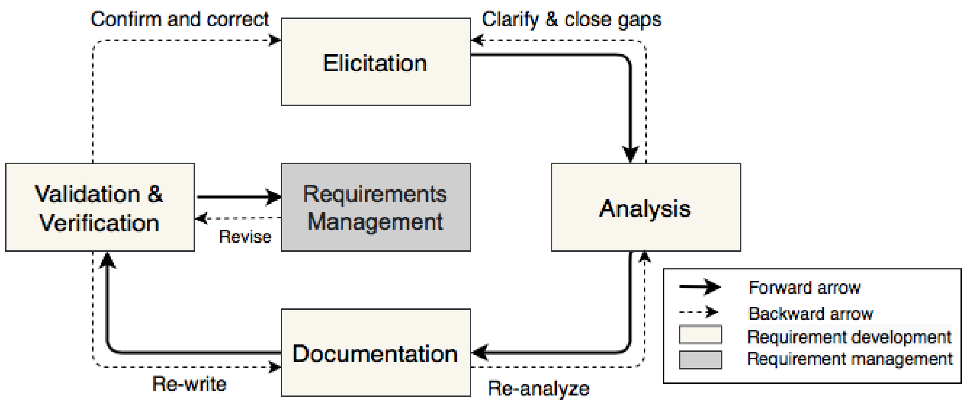


Figure 2.3: RE processes

various sources, examples of which include external data, applicable standards, and stakeholders (from whom requirements are gathered using well-defined approaches, such as brainstorming, card sorting, and prototyping [Lap17]). Elicitation includes determining NFRs, which might be a challenge, as some NFRs are not as clear in stakeholders' minds as FRs [CY04].

Requirements analysis and reconciliation. This activity is undertaken to understand the requirements collected from stakeholders. It includes addressing problems with the requirements (e.g., conflicts, vagueness, incorrectness, inconsistencies, and incompleteness), negotiating priorities, building a prototype, and evaluating feasibility [Lap17, WB13].

Requirements documentation/representation. The aim of this activity is to document the analyzed requirements consistently and understandably. To facilitate communicating these requirements and converting them into system architecture, different representations may be used [Lap17, WB13]. Examples of these representations are sketches, NL, and formal mathematical representation [Lap17].

Requirements verification and validation. This is the process of determining whether the requirements/specifications satisfy the users' needs [Lap17, WB13]. It involves going back to stakeholders with the specification and asking them to review it [CY04]. If there are any problems, they should be fixed until the stakeholders' complete agreement is obtained [WB13].

Requirements management. This activity is conducted to control changes to the requirements that occur during system development [CY04]. It includes evaluating the effects of the changes (cost and time), making appropriate business decisions regarding them, tracking the changes, and maintaining traceability [Lap17, WB13].

2.2 Machine Learning in Automated Text Classification: An Overview

Text classification, also known as *text categorization*, is the task of assigning one or more predefined categories to text documents based on their content [FML05]. ML algorithms, particularly supervised ML algorithms, have become increasingly used for performing this task [AZ12, IKT05, KJMH$^+$19]. Requirements classification can be seen as a particular instance of *text classification*, in that text documents contain requirements statements. Thus, we can use the concepts and methods of ML-based text classification, which are well-established, to inform our understanding of requirements classification, which is still a relatively young area. In this section, we attempt to build such understanding and use it as the background for our work.

2.2.1 Concepts and Processes

According to Deng et al., [DLWZ19], text classification can be formally defined as follows: Given a collection of N documents $D = \{d_1, d_2, ..., d_N\}$ and a set of k predefined categories $C = \{c_1, c_2, ..., c_k\}$, the problem of text classification can be modeled as finding a mapping F from the Cartesian product $D \times C$ to a set $\{True, False\}$, i.e., $F : D \times C \rightarrow \{True, False\}$. Based on this mapping, for a document $d_i \in D$ and a category $c_j \in C$, if $F(d_i, c_j) = True$, then d_i belongs to category c_j, otherwise d_i does not belong to c_j.

However, since F has no knowledge of the relationship between the content of a document in D and its category in C, the first step in text classification is to train F by feeding to it a training set containing m documents $D' = \{d'_1, d'_2, ..., d'_m\}$ where every document $d'_i \in D'$ is associated with a category $c_j \in C$. The objective of F' is to learn $F'(d'_i, c_j) = True$. After learning, F' can be applied to classify D, by automatically assigning every requirement $d_i \in D$ with a category $c_j \in C$, such that $F'(d_i, c_j) = True$. F' is called the *text classifier* or *classifier*.

The process of text classification as described above is called *supervised learning* [HTF09], as it involves training a classifier to learn the mapping between the documents in a training set and their intended categories. Because of this, text classification is also called *supervised classification* [BKL09]. Supervised learning is sometimes referred to as *predictive modeling* [Bro16] and the classifier is called the *predictive model*, since classification in ML is essentially a predictive task that makes predictions

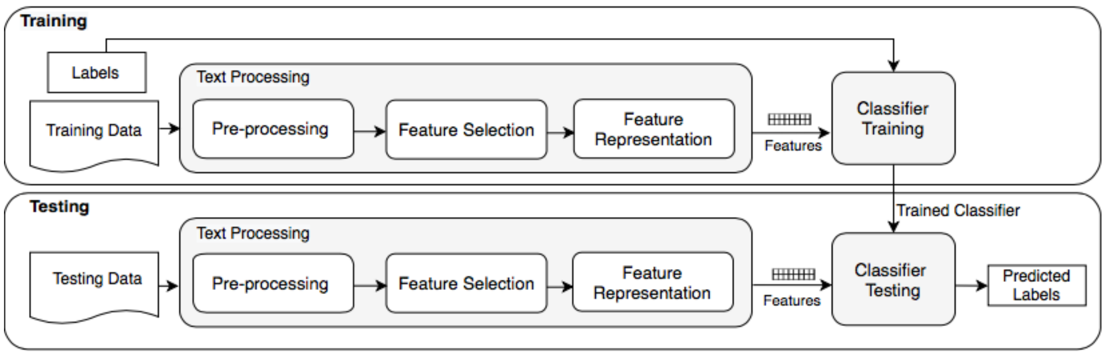


Figure 2.4: Process of text classification by supervised learning. During the training phase, a text classifier is trained on a training set, whose data have been classified in advance. During the testing phase, the trained classifier is applied to the new, unseen data. Both training and testing data will undergo a series of text preparation operations.

of correct categories for new documents [SGSG19].

Figure 2.4 provides a general process for text classification, which consists of two distinct phases: training and testing. The training phase is concerned with training and validating a text classifier whereas the testing phase is for an unbiased evaluation of the trained classifier, to ensure it can generalize well to new documents [Bro14]. As shown in Figure 2.4, before we train or test the classifier, the text documents in the input data need to undergo a series of transformations called *text processing*, to remove irrelevant words and to represent the text documents in a form that can be directly interpreted by the learning algorithm. Generally, text processing entails three distinct tasks, namely, *text pre-processing, feature selection*, and *feature representation.* These tasks are briefly described as follows.

Text pre-processing: This task performs a series of text cleaning operations on the text data, such as tokenization (splitting a sentence into a series of words), stop-words removal (removing auxiliary verbs, conjunctions, and articles in sentences), lowercase conversion (converting all letters into lowercase), and lemmatization (determining the infinitive form of verbs, and the singular form of nouns and adjectives of each word). These operations help remove the noise on the text, such as unnecessary words (i.e., stop words) and unnecessary characters (e.g., punctuation and special characters). Text pre-processing thus reduces the number of words in the text and, in doing so, paves the way for effective feature selection [KJMH$^+$19] and helps improve classification accuracy [UG14].

Features selection: This step is concerned with eliminating irrelevant, less informative and redundant features, so as to further reduce the number of text features in the learning space, thus improving classification efficiency and accuracy [DLWZ19]. Feature selection has been extensively studied by researchers from diverse communities, resulting in an abundance of methods (see, for example, [DL97, DLWZ19, TAL14]). Feature selection methods can be broadly divided into three categories, namely, *filter*, *wrapper*, and *embedded* [DLWZ19, CHTQ09]. The methods commonly used for text classification are filter methods, due to their simplicity and efficiency [DLWZ19]. Filter methods select features independently of ML algorithms by looking only at the intrinsic properties of the data, such as information, distance, consistency, and correlation [Seb02]. Traditional filter methods are based on statistical information retrieval techniques, such as term frequency (TF), term frequency inverse document frequency (TF-IDF), information gain (IG), and Chi-squared [DLWZ19, For03]. In recent years, new filter methods based on language features have been developed. Such *linguistic based filter methods* employ natural language processing (NLP) techniques such as part-of-speech (POS) tagging, syntactic parsing [KWM11, MMS13], and semantic role labelling [FDSSVA19] to extract important syntactic features from the text [MKIZ14]. A more detailed overview of feature selection methods is presented in Section 4.1.2.

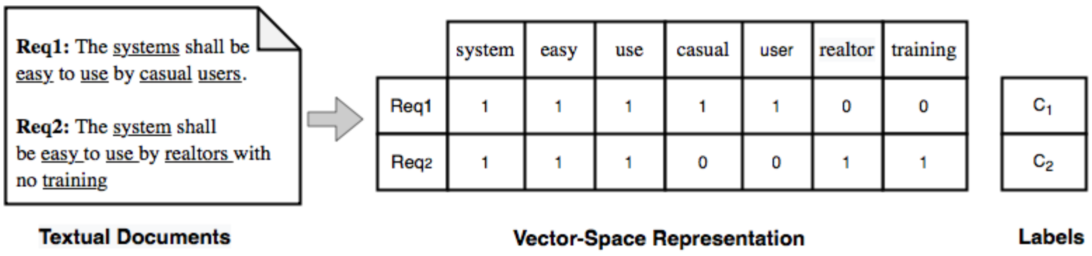
Feature representation: This step converts the selected features from each document into a vector of feature weights that can be processed by the learning algorithm. The feature weights are calculated using a suitable weighting method, such as TF, that determines the frequency of each feature in the document.

To illustrate the text processing tasks, consider a simple training set with two requirements:

Req1: The systems shall be easy to use by casual users.

Req2: The system shall be easy to use by realtors with no training.

The first step of text processing is to pre-process these requirements, by removing stop words in them (e.g., the, shall, be, to, by, with, no). Then, the relevant features of these requirements are selected using part of speech (POS) tags, resulting in a set of seven features system, easy, use, casual, users, relators, and training. Finally, this feature set is represented as a feature space model as shown in (Figure 2.5), with each requirement represented as a seven-dimensional feature vector. The size of the feature space is *14 = 7 features x 2 requirements*. This is an overly simplified example to illustrate how text preparation works; in practice, however, ML algorithms need to be trained on a large number of preclassified documents in order to achieve high classification accuracy.



Req1: The systems shall be easy to use by casual users.

Req2: The system shall be easy to use by realtors with no training

	system	easy	use	casual	user	realtor	training
Req1	1	1	1	1	1	0	0
Req2	1	1	1	0	0	1	1

Labels
C_1
C_2

Figure 2.5: Feature representation of two requirements in a vector (or feature) space, in which "1" indicates the presence of the corresponding feature in a requirement and "0" means the absence.

2.2.2 Classification Algorithms

Under supervised learning, ML algorithms are broadly divided into two categories: those which produce discrete categories are referred to as a *classification algorithms*, whereas those which return continuous values are called *regression algorithms* [SGSG19]. Classification is viewed as the analogue of regression when the variable being predicted is discrete, rather than continuous [PMB09]. Both classification and regression algorithms can be applied to text classification [AZ12]. In RE, however, as requirements are typically classified into discrete categories (e.g., functional, non-functional, usability, security, etc.), current ML approaches to requirements classification have been mainly based on classification algorithms [AAS+19].

Classification algorithms can be further grouped into Support Vector Machine (SVM) Based, Tree Based, Neural Network Based, Bayesian, and Proximity Based [AZ12]. SVM-based algorithms gained popularity among the ML community for their high performance and flexibility [For03, SGSG19], as they can be used for both classification and regression. Tree-based algorithms, such as Decision Tree (DT) and Random Forests (RF), are fast and accurate for document categorization [KJMH+19]. Most recently, neural network-based algorithms such as Deep Neural Networks (DNN), CNN, RNN, deep belief network (DBN), hierarchical attention networks (HAN), and combination techniques, have been applied in a wide range of domains for classification purposes [AZ12]. Proximity-based algorithms, such as Naïve Bayes (NB) and K-Nearest Neighbour (KNN), are more traditional but still commonly used in the scientific community [KJMH+19]. Comprehensive surveys of the state of the art text classification algorithms are provided by Aggarwal and Zhai [AZ12], and Kowsari et al. [KJMH+19].

In RE, classification algorithms that have been widely used and produced good

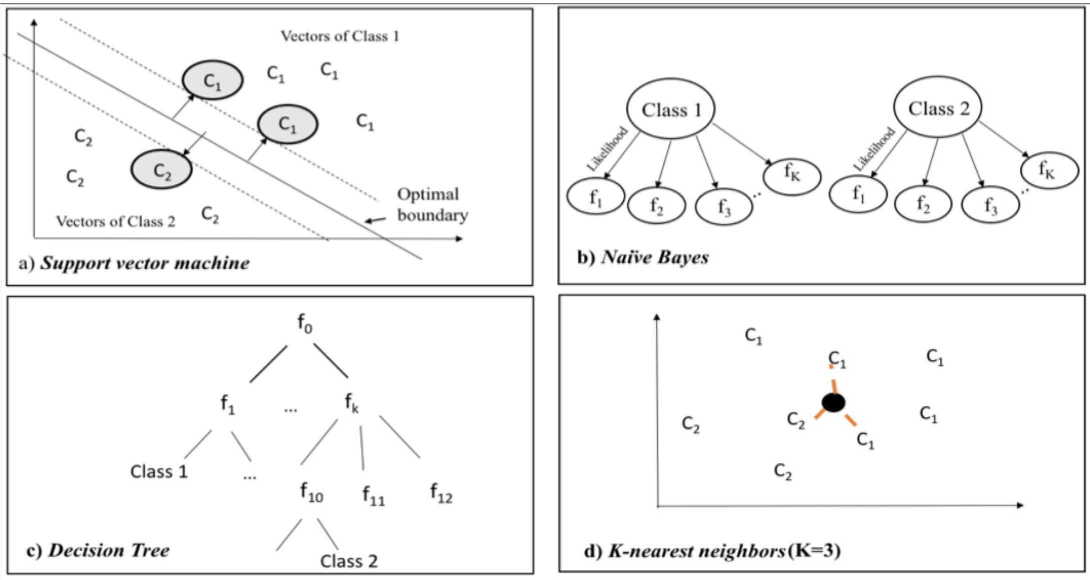


Figure 2.6: Illustration of Four Machine Learning Algorithms that are Commonly Used to Classify Non-functional Requirements. These Algorithms are Support Vector Machine, Naïve Byes, Decision Tree and K-Nearest Neighbors.

results are SVM, NB, DT, and KNN [BZ19]. These four algorithms are illustrated in Figure 2.6 and briefly described as follows.

Support Vector Machine (SVM)

SVM is a statistical-based algorithm that operates by defining the best decision boundary which separates training data points belonging to different categories. The best decision boundary is assumed to be obtained when the distance between data points belonging to different classes is maximal. Only those data points that are close to the decision boundary (known as support vectors) are used in classification tasks, while the rest of the training data points are ignored [CV95].

SVMs were originally designed for linear two-class classifications (i.e., binary classifications). However, they could be implemented in multi-class classifications by applying well-known techniques one-versus-one, or by using the one-versus-rest technique [KJMH+19]. The one-versus-rest technique constructs K SVM models, where K is the number of classes; the training for each model assumes that one class is positive, while all others are negative (e.g., class 1 against non-class1) [LZ05]. The new data point is classified based on the binary SVM that shows the best performance (i.e. high accuracy). The one-versus-one technique trains a classifier for each pair of classes, thus, K (K-1)/2 binary SVMs are trained. A voting strategy is then used to combine the outputs and predict the final decision of a new data point.

The main advantage of SVM is the ability to handle high dimensional input space (e.g., text) and pick out irrelevant features [KBLK10]. Therefore, it is less prone to overfitting. However, the major drawback of SVM is that it is computationally expensive, especially when the size of a training dataset grows [KBLK10].

Naïve Baye (NB)

NB is a simple probabilistic method that uses Bayes' theorem with a strong assumption of independence between features [SY19]. Although this assumption is unrealistic, the NB classifier is remarkably successful in practice, especially in text classification [PS03]. NB computes the probability that an input document belongs to different predefined classes; the class with the highest probability will be predicted. Assuming that R is an NFR represented by a vector of K dimensions, R= (w_1, w_2, .., w_K), the probability of R belonging to label C would be computed as follows [PS03]:

$$P(C|R) = \frac{P(C) \times P(\sum_{i=1}^{k} w_i|C)}{P(\sum_{i=1}^{k} w_i} \tag{2.1}$$

The strengths of this method are that it is easy to implement and compute and requires a small amount of training data to learn. However, this method performs poorly with highly correlated features due to the independence assumption [KBLK10]. Figure 2.6 (b) shows an example of Naive Bayes; each feature node has no parent except the class node. A lack of arrows between features indicates the independence of features. The arrows in this figure refer to the probability of having a specific feature with a specific class (i.e. parent).

Decision Tree (DT)

DT [Qui86] operates by constructing trees based on features in a training dataset. Each node is a decision node representing a feature in an instance to be classified, the edges represent the possible values for a particular feature, while leaves are predicted class labels[KZP07]. Classifying a new sample is straightforward—following a path starting from the root node through the tree to a leaf [SWK09]. The popular decision tree algorithms are ID3 (Iterative Dichotomiser) [Qui86], C4.5 [Qui93], CART (Classification And Regression Trees)[BFOS84]. Each algorithm uses different splitting techniques for splitting the data node at each level. For example, ID3 uses Information

Gain, C4.5 uses Gain Ratio, while CART uses the Gini Index as the splitting criterion [PARS13, SG14]. The main advantage of decision tree algorithms is that they are easy to understand and interpret, even for non-experts [Phy09]. However, the disadvantages are overfitting and poor generalization as decision tree classifiers are built to classify training datasets effectively, neglecting the performance of classifying other documents [KBLK10].

K-Nearest Neighbors (KNN)

KNN is based on the principle that similar instances in the dataset generally exist close to each other. This algorithm classifies a new instance by observing the K-nearest neighbours and identifying the most frequent class label [KZP07, KBLK10]. During the training phase, KNN does not need the use of a training dataset. Therefore, KNN is known as lazy ML algorithms, as it defers the decision of generalizing the training data until a new query is encountered [Seb02]. However, the training examples will be used during the testing (evaluation). This algorithm is easy to understand and easy to implement. However, it takes a long time to execute because it uses all features in distance computation [KBLK10].

2.2.3 Evaluation Techniques and Metrics

The performance of a supervised ML classifier is measured through three primary steps: (1) choosing an evaluation metrics (i.e., scores); (2) deciding the method for score estimation; and (3), performing statistical significance tests to compare the results of different classifiers [SIL15, Stą17].

Evaluation Metrics

A wide variety of metrics (also known as *score* and *measures*) have been used to evaluate supervised classifiers [SIL15, Stą17]. This section provides an overview of the metrics used (or frequently mentioned) in this thesis.

Precision, recall, and f-score are among the most frequent measures used in evaluating ML classifiers' performance, in general [SIL15, Stą17], and requirements classifiers, in particular [BZ19]. These measures are more appropriate for imbalanced datasets, compared to accuracy and classification error [SIL15, Stą17]. Recall is the percentage of instances that are correctly retrieved and classified. It is defined as follows:

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{2.2}$$

Precision is the percentage of correctly classified instances in relation to the total number of instances retrieved. It is defined as:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2.3}$$

F-score is a harmonic mean between recall and precision. It is defined as

$$F - score = \frac{(1+\beta^2) \times\ Recall.Precision}{\beta^2 \times Recall + Precision} \tag{2.4}$$

In Formula 2.2 and 2.3, true positive is the number correctly classified requirements, while false positive is the number of incorrectly classified requirements, and false negative is the number of requirements that incorrectly not classified. In Formula 2.4, β is to weight harmonic mean of precision and recall for emphasizing either precision or recall. For example, if both recall and precision have the same importance in a classification task, $\beta = 1$ and $F - score$ is referred to as $F1 - score$. $\beta = 2$ emphasize recall over precision, while $\beta = 0.5$ emphasize precision over recall.

The equations mentioned above measure the performance of a single class. To extend these measures to multiple classes, two types of average are commonly used: micro and macro [PZZ12]. For macro-averaging, each score is computed locally for each class. Then, the average of all classes is calculated. For example, macro precision for k classes is computed as the following.

$$P(Macro) = \frac{1}{k}.\sum_{i=1}^{k} Precision \tag{2.5}$$

For micro-averaging, the score is computed globally over all the classes (i.e., the average of true positives, false positives, and false negatives for all classes). For example, the equations to compute micro precision for k classes is as the following:

$$P(micro) = \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k}(TP_i + FP_i)} \tag{2.6}$$

Micro average does not handle rare classes as major ones; it gives equal importance to each instance; therefore, a class with more instances contributes more to the final score. In contrast, Macro average treats each class equally; thus, it is widely used in classification and retrieval problems [NPK$^+$16]. Besides, in multi-class classification, the micro-averages of precision, recall, and f-score have the same value as they are computed over all classes; the incorrectly predicated samples represent both false positive and false negative. For example, in the case of "A is misclassified as B", it is false positive for B and false negative for A.

The values of precision, recall, and f-score are in the range of {0-1}. The perfect precision value is 1.0 means that each requirement is labelled in a class is indeed belong to this class. The perfect recall value is 1.0 means that every requirement that belongs to a class is classified into this class.

Recall is favoured over precision for requirements engineering tools in general and for requirements classification problems in particular [BGST12, CHSZS07, RDPM19]. According to Cleland-Huang et al. [CHSZS07], recall is more important than precision in NFR classification, since searching for a false positive in the output is significantly easier than manually browsing the entire document looking for entirely missed NFRs. However, the precision is also important to lower the false positives and the levels of user frustration. Thus, in this thesis, we consider both recall and precision.

Score Estimation Methods

Several methods are regularly used to estimate ML classifier scores, including holdout and K-fold cross-validation. Holdout divides a dataset randomly into training and testing subsets according to a given proportion (e.g., 70%-30%). The training dataset is used for building a classifier, while the testing dataset is for evaluating the classifier. This technique provides a biased error estimate as it depends on a particular division of the dataset; thus, much of the data is not used in training a prediction model (i.e., statistically inefficient) [TMHM16].

K-fold cross-validation (Figure 2.7) is widely used in the ML community and for requirements classification in the RE domain [BZ19]. This method creates a K-fold partition of the dataset and runs the classifier K times using (k-1) folds for training and one-fold for testing. The classification performance is estimated as the average of the performance scores obtained from the K experiments (K iterations). K-fold cross-validation is less biased than holdout [TMHM16], as it does not rely on a particular partitioning of a dataset. In addition, each instance in a dataset is used for both training and testing [TMHM16]. Nonetheless, it is computationally expensive since the training process must be executed K times.

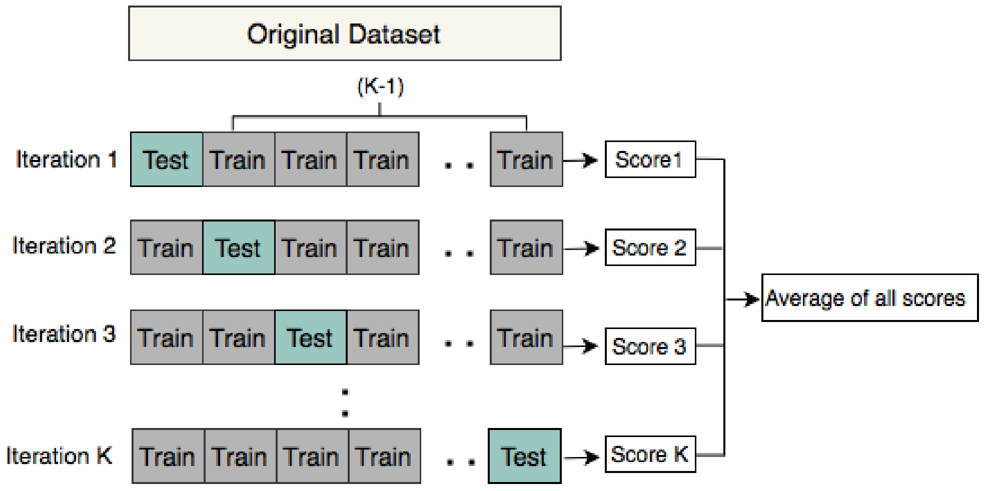


Figure 2.7: K-fold cross-validation

The choice of K is a trade-off; a larger K leads to less bias (fewer errors in training data), as the number of training instances is closer to the total instances and high variance (more errors in a testing dataset). Consequently, the risk of overfitting and execution costs (time) of a model are increased. The popular value of K is K = 10 [TMHM16], which is recommended by Han et al.[HKP12] due to its relatively low bias and variance. In this thesis, we used K-fold cross-validation with k = 10 to estimate precision, recall, and the f1-score.

Statistical Comparisons of Classifiers

To determine if the observed differences between classifiers are real or random, especially when the differences are unclear, statistical assessments are used [Stą17, Dem06, SIL15]. Examples of these tests that are applicable with k-fold cross-validation are Friedman's Aligned Ranks [SIL15] and Wilcoxon signed-rank test [Wil92]. Stąpor [Stą17] recommended the use of Friedman's Aligned Ranks test to compare the performances of multiple classifiers (less than five classifiers) with non-normal distribution data. However, this test does not provide pairwise comparisons of the classifiers. Therefore, Wilcoxon signed-rank test [Wil92] can be used instead, as recommended by Demšar [Dem06], to compare the performances between two classifiers with non-normal distribution data. The normality test can be conducted using a QQ plot [WG68] or KolmogorovSmirnov test [MJ51].

2.2.4 Model Diagnosis and Tuning

Hyperparameter tuning and learning curve analysis are two steps applied during building a ML classifier or analyzing the classification results. These steps are not mandatory to build or evaluate a prediction model; however, it helps to maximize classification accuracy (hyperparameter tuning) or examine ML models' generalizability (learning curve analysis). As these steps are used in this research, they are briefly described here.

Hyperparameter Tuning

ML algorithms have a set of hyper-parameters that need to be estimated from a dataset to maximize the performance score. The optimal values of hyper-parameters (e.g., K in KNN) cannot be estimated manually. Thus, the best way to choose these optimal values is to empirically try different parameter values and choose the best combination of values [Swa19].

There are two standard methods for hyperparameter tuning: grid search and random search [Swa19]. A list of predefined values for each parameter is required by the grid search. Then, all possible combinations of given values are used to train an ML model, where the best combination is chosen to build the final prediction model. In contrast, a given range of values of each hyperparameter is required by random search. Different combinations of hyperparameter values are empirically tested, the values that lead to high results are used to build the final model. The parameter values are selected randomly from a given range during the tuning process, where not all possible combinations are tested. Therefore, in comparison with grid search, random search is less computationally expensive. However, it may miss the optional combination since not all combinations are tested.

The hyperparameters are tuned during a training phase (before building an ML model) using a part of the training dataset in which an ML algorithm is evaluated against different values using the other part of the training dataset. The values that lead to the best performance are then used to build the model.

The evaluation methods and metrics used for tuning hyperparameters are the same as those used for testing an ML model. Varma and Simon[VS06] and Cawley and Talbot [CT10] recommended nested K-fold cross-validation for tuning the hyperparameters of small training datasets. Nested cross-validation re-applies K-fold cross-validation within the outer cross-validation training set (see Figure 2.8), where the K

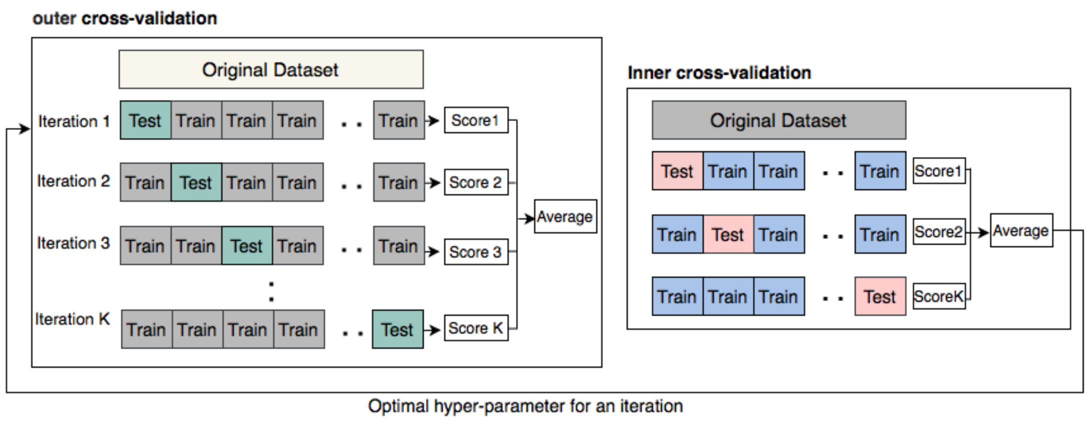

outer cross-validation
Original Dataset
Iteration 1
Test Train Train Train . . Train
Score1
Iteration 2
Train Test Train Train . . Train
Score 2
Iteration 3
Train Train Test Train . . Train
Score 3
Iteration K
Train Train Train Train . . Test
Score K
Average
Inner cross-validation
Original Dataset
Test Train Train . . Train
Score1
Train Test Train . . Train
Score2
Train Train Train . . Test
ScoreK
Average
Optimal hyper-parameter for an iteration

Figure 2.8: Nested cross-validation for tuning hyper-parameters

of inner cross-validation is different from outer cross-validation. The performance of an ML model in inner cross-validation is measured first, and hyperparameters are chosen for use in building a model for outer cross-validation. This process is repeated for each iteration in outer cross-validation, as shown in Figure 2.8.

Learning Curve

Learning curve plots are commonly used to analyze trained classifiers' performance with different sizes of training data, and diagnose classifiers problems (e.g., overfitting or underfitting [Gér19]). The horizontal axis indicates the number of instances sampled that selected randomly from a training dataset. The vertical axis represents the performance metrics (e.g., errors rate and F-score) of the trained model. Figure 2.9 shows examples of learning curves.

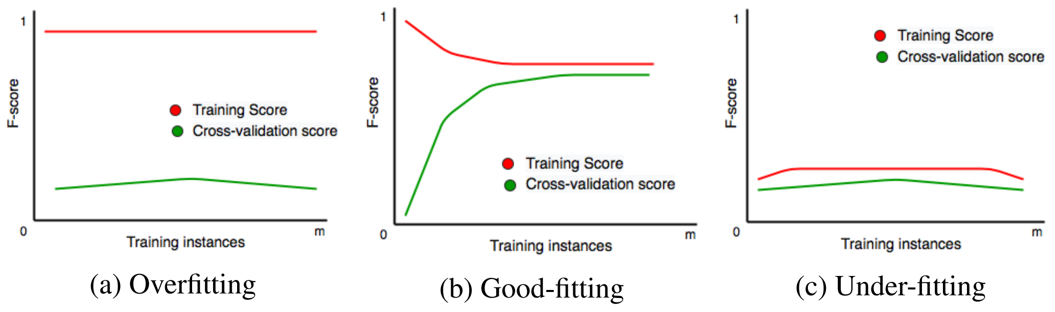


Figure 2.9: Learning curves of three models, , where m is training data size.. The first model suffers from overfitting due to low basie and high variance. The second model shows a good-fitting (low basie and low variance). The third model is underfiring (high bias and high variance).

As Figure 2.9 shows, there are two curves in each plot: the training curve and the cross-validation curve. The training curve shows the performance of a trained model in classifying the training dataset used to build that model, illustrating how well the model fits the training dataset. A large number of errors in the training curve means that the model suffers from high bias, while small errors indicate low bias. The cross-validation curve shows the cross-validation results of the trained model. Large numbers of errors indicate that a model suffers from high variance, while a small number means low variance. If the model performs well on training data but generalizes poorly according to cross-validation metrics, the model suffers from overfitting [Gér19]. Overfitting can be addressed by adding more examples until the validation errors are equal to training errors. If both training and cross-validation curves perform poorly, that model is underfitting. Underfitting a model means: it is simple and not appropriate for a classification task. Adding more examples cannot help to improve the model, but using a complex model or different features could do [Gér19].

2.3 Summary

This chapter presents an overview of two main areas of this research: RE and text classification using ML algorithms. The review of RE illustrates the importance of requirements to the success of the system to be developed and the development process itself. It also shows the disagreement in defining and naming NFRs, and describes the process of handling requirements (including NFRs) in RE processes. The review of supervised learning aims to improve the understandability of the use of ML in text classification, with a focus on techniques applied in this thesis.

The main findings of this chapter can be summarized as follows:

- Addressing NFRs early is essential to reduce maintenance costs.
- There is no consensus regarding NFRs, for example, in their definition and classification.
- Textual inputs need to be processed before including them in the ML algorithm.
- Text processing includes preprocessing, feature selection, and representation.
- Evaluating the performance of a supervised ML classifier involves choosing measures evaluation metrics, score estimation methods, and statistical significance tests.

- Hyperparameter tuning methods are used to choose the optimal value for the hyperparameter of the ML algorithms.
- Learning curves are applied to check whether the ML models work correctly or need improvement and diagnose biases and variance.

The next chapter (Chapter 3) provides a more detailed overview of how these models were used and evaluated in NFRs classification.

Chapter 3

A Systematic Review of Machine Learning Methods for or Identification and Classification of Non-Functional Requirements

"If I have seen further than others, it is by standing upon the shoulders of giants."

Isaac Newton

Since the middle of 2000, many ML methods have been proposed to automatically identify and classify NLRs from NL documents. To understand how ML models have been used and evaluated in NFRs classification, we conducted a systematic literature review. The first version of this review was published in the Expert system with an Application Journal in 2019 [BZ19]. That version reviewed 26 studies published between 2007-2017. This chapter, however, presents an updated version of the review, including 51 studies published between 2017- early 2021.

The sections of this chapter are organized as follows: Section 3.1 shows the motivation and review questions, Section 3.2 provides an overview of related reviews, Section 3.3 illustrates the review method and process, Section 3.4 shows results, Section 3.5 discusses our key research findings and open challenges, threats to the review validity is in Section 3.6, followed by Section 3.7 for the summary.

3.1 Motivation and Research Questions

As indicated in Chapter 1, due to the difficulty of manually analyzing NFRS (time-consuming and error-prone), RE researchers have been proposing automatic or semi-automatic approaches for identifying NFRs in requirements documents [SRS14, VR11, SLRR05]. ML algorithms have been integrated into these approaches, with promising results being reported [CHSZS07]. However, a systematic understanding of these emerging methods is currently inadequate in the literature. This review aims to fill this gap.

Specifically, this chapter reports on a systematic review of **51** current ML-based methods. The review intends to determine what ML algorithms have been used to classify NFRs, how these algorithms work, and how they are evaluated. These findings will enable us to identify what challenges need to be addressed to improve the state of ML-based methods. Our review is driven by the following research questions (RQ):

- RQ1: What machine learning algorithms have been applied in the selected studies? Which ones are the most popular?
- RQ2: What are the processes that the reported ML-based approaches follow to identify and classify NFRs in requirements documents?
- RQ3: What measures have been used to evaluate the performance of the ML algorithms applied in these methods? What are the performance results of these algorithms?

3.2 Related Reviews

A few related reviews are available in the literature. To the best of our knowledge, only one review was published before the first version of our review in 2019. This review was conducted by Meth et al. [MBM13] to explore the tools used for eliciting automatic requirements from textual documents. That review included 36 studies that were published between January 1992 and March 2012. Those studies have been analyzed by adopting two perspectives: design, which focuses on the technical concepts adopted in each tool, and evaluation, which describes methods for evaluating the effectiveness of those tools. From the design perspective, the studies involved in that review were classified into four categories: identifying both FRs and NFRs, generating models, analyzing quality requirements by defining defects, and finding key abstractions. Besides, that review examined the degree of automation for each tool (full or semi-automation) and the approaches used to generate knowledge for the reuse of either requirements or knowledge related to requirements. Conversely, the evaluation perspective contains evaluation approaches, concepts, and measures used to evaluate tool performance. Identifying requirements automatically is a small part of that review, and these requirements might be either NFRs or FRs, regardless of the techniques used. By contrast, our review focuses only on identifying NFRs via ML algorithms.

Recently, more similar reviews were published [PVSGOH20, AFK$^+$20, IEL18]. Besides the fact that these reviews were published after the first version of our review, they also were conducted for different purposes than this review. For example, Pérez-Verdejo et al. in 2020 [PVSGOH20] conducted a review on using ML for automated requirement classification. Their review aims at identifying the applications of ML techniques in software requirements classification to provide insight into state-of-the-art in this field. Their review is restricted to studies published between 2010 to 2019, containing only 13 studies. In contrast, our review includes 51 studies published from 2007 to early 2021. Whereas their review addresses ML in requirement classification in general, our review particularly focuses on NFRs.

Other examples of recently published related reviews are those conducted by Ahmad et al. [AFK$^+$20] and Iqbal et al. [IEL18]. Ahmad et al.'s review aims to identify the ML algorithms applied to identify software requirements on the Stack Overflow platform. Iqbal et al. [IEL18] conducted a (traditional) related review to provide an overview of ML algorithms used to support RE tasks, including requirement classification. Both reviews have different aims than ours. Ahmed et al.'s review is limited to StackOverflow [1] as a source of requirements, whereas our reviews include any textual sources. Iqbal et al. include all ML learning applications in RE, while our review focuses on a specific task, which is NFR classification. Moreover, Iqbal et al. concluded a traditional review while we do a systematic review.

3.3 Review Method and Process

To answer our research questions, we have adopted the snowballing approach proposed by Wohlin [Woh14] to identify relevant studies. Snowballing is an alternative to the traditional systematic literature review (SLR) approach [KC07]. The idea is to use

[1]https://stackoverflow.com/

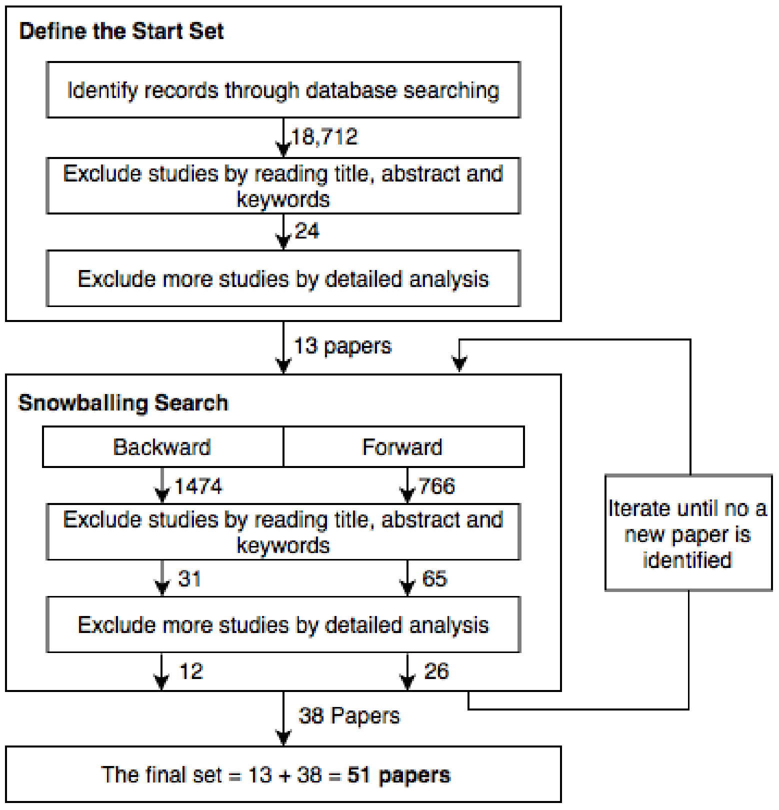

Define the Start Set
Identify records through database searching
18,712
Exclude studies by reading title, abstract and keywords
24
Exclude more studies by detailed analysis
13 papers
Snowballing Search
Backward
Forward
1474
766
Exclude studies by reading title, abstract and keywords
31
65
Exclude more studies by detailed analysis
12
26
Iterate until no a new paper is identified
38 Papers
The final set = 13 + 38 = 51 papers

Figure 3.1: The process of studies selection based on Wohlin's snowballing method [Woh14]

the reference list of a paper and the paper's citations to identify additional papers systematically. Snowballing consists of two steps: backward snowballing (looking at the reference lists of a paper) and forward snowballing (looking at the citations in which the paper is actually cited). Figure 3.1 depicts the process of snowballing [Woh14].

The starting point of the snowballing approach is the identification of a start set of relevant papers. Based on each paper in the start set, one can then conduct backward and forward snowballing. In the following subsections, we detail how we used the snowballing approach to identify the relevant papers for our review.

3.3.1 Identification of the Start Set

The start set was identified through the traditional SLR search method, in which search strings are formulated to query relevant online databases. To avoid bias towards a specific publisher, Wohlin [Woh14] advises to use Google Scholar to search for the start set. However, we found that the search engine of Google Scholar is too general to be really useful. For example, the search string "('non-functional requirements' OR

Criterion	Description	Main keywords	Alternatives
Population	Requirements Engineering (RE)	Non-Functional Requirement	Non-functional requirements OR NFRs OR Quality Requirements OR Quality Attributes
		Requirements Elicitation, Requirements Analysis, Requirements Specification	Requirements Extraction OR Requirements Analysis OR Requirements Specification OR Requirements Categorization OR Requirements Classification
Intervention	Machine Learning (ML) approaches for RE	Machine Learning	Machine Learning OR Supervised learning OR Unsupervised learning OR Semi-supervised learning
Comparison	Not applicable	Not applicable	
Outcome	An understanding of the ML approaches for identifying and classifying non-functional requirements (NFRs) from natural language documents		
Contex	Requirements Engineering (RE)		

Table 3.1: PICOC criteria to define the search string for the start set

NFRs) AND ‘classification’ AND ‘machine learning’ ” returned 26,700 results. We have therefore decided to conduct searches on online databases that are known to us. Our search string was defined by breaking down the research questions according to the PICOC criteria (population, intervention, comparison, outcome, and context), as recommended by Kitchenham and Charters, 2007 [KC07]. These terms were connected using Boolean operators; the operator OR was used for synonyms (alternative terms) and AND was used for linking the search terms, as illustrated in Table 3.1.

By manually inspecting the search results (18,712 records), we selected 120 papers for further selection. This entailed (1) reading the title and abstract of each paper and (2) reading the full text of each remaining paper. The following inclusion and exclusion criteria were used for study selection:

Inclusion:

- All the papers that present the primary studies on using ML algorithms for identifying NFRs are included.

- If the same method is reported by more than one paper, the paper that provides the most detailed description of the ML method is included.

Exclusion:

- Grey literature and non-English written paper are excluded.
- Papers that present secondary studies, such as surveys and literature reviews, are excluded.

At the end of the selection process, we identified 13 relevant papers, which were then included in our start set (see Table 3.2).

3.3.2 Backward Snowballing

For each paper in the start set, we conducted a backward snowballing search on the paper's references. The process of backward snowballing, as shown in Figure 3.1, is detailed as follows.

First, the title and reference context of each referenced paper was reviewed, and in some cases, additional sections of the referenced paper, such as the abstract and keywords, were evaluated. Reference context was based on the text surrounding the reference in the paper.

Second, the inclusion and exclusion criteria were applied based on a full-text reading. Papers identified as relevant were added to the starting set, and this process was repeated until no new papers were identified. During four iterations of this process, 1474 referenced papers were examined, 31 of them were identified as relevant, but only 12 were added to the start set. The start set now consists of 25 papers.

3.3.3 Forward Snowballing

For each paper in the start set, we conducted a forward snowballing search on the paper's citations. The citations of each paper in the start set were identified on Google Scholar. Each citation paper was reviewed as follows: first, the title, abstract, keywords, and reference context (i.e., the context of text surrounding each citation) of each study were reviewed. Relevant papers were added to the starting set, and this process was repeated until no new papers were identified. During four iterations, 766 studies were examined, 65 studies were identified, and 26 relevant studies were added to the start set based on the inclusion and exclusion criteria.

No	Author	Year	Title	Publication venue	Database	Citation counts*
1	Casamayor et al.	2010	Identification of non-functional requirements in textual specifications: A semi-supervised learning approach	Information and Software Technology	Science Direct	149
2	Slankas and Williams	2013	Automated extraction of non-functional requirements in available documentation	NaturaLiSE	IEEExplore	96
3	Kurtanović and Maalej	2017	Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning	RE Conference	IEEExplore	86
4	Lu and Liang	2017	Automatic Classification of Non-Functional Requirements from Augmented App User Reviews	EASE	ACM	56
5	Abad et al.	2017	What Works Better? A Study of Classifying Requirements	RE Conference	IEEExplore	55
6	Hussain et al.	2008	Using linguistic knowledge to classify non-functional requirements in SRS documents	NLDB	Springer Link	51
7	Knauss et al.	2011	Supporting Requirements Engineers in Recognising Security Issues	REFSQ	Springer Link	39
8	Jindal et al.	2016	Automated classification of security requirements	ICACCI	IEEExplore	19
9	Deocadez et al.	2017	Automatically Classifying Requirements from App Stores: A Preliminary Study	RE Conference	IEEExplore	10
10	Gokyer et al.	2008	Non-functional requirements to architectural concerns: ML and NLP at crossroads	ICSEA	IEEExplore	14
11	Nguyen et al.	2015	Rule-based extraction of goal-use case models from text	ESEC/FSE	ACM	12
12	Malhotra et al.	2016	Analyzing and evaluating security features in software requirements	ICICCS-INBUSH	IEEExplore	7
13	Knauss and Ott	2014	(Semi-) automatic Categorization of Natural Language Requirements	REFSQ	Springer Link	5

* Collected in February 2021

Table 3.2: The initial start set of the relevant papers for snowballing

Data item	Description
Bibliographic information	Authors, title, publication venue, and publication year
Study goal/Contribution	The main purpose of the selected paper
ML algorithm	Type of ML algorithm used to identify or classify the textual requirements
Requirements document	The source and size of requirements documents
Types of NFRs	Types of NFRs that automatically identified by the studies
ML approach	The process and techniques used for identifying the NFRs
Evolution methods and results	How the effectiveness of the algorithm was measured in the selected paper and what were the results

Table 3.3: Data extraction form

At the end of forwarding snowballing, 51 relevant papers were selected as the final set for our review. This set of papers is listed in Appendix A (Table A.1), numbered as S1, S2, etc., in chronological order.

3.3.4 Extracting and Synthesizing the Data

The required data were extracted from each of the 51 selected studies. The data extraction form (see Table 3.3) was used to record the data for each study. Two types of data were extracted: the data required for answering the research questions and the data required for displaying the bibliographic information of the study. The extracted data were stored in an Excel file.

The data synthesis method used in this review was based on the constant comparison method (CCM), a core element of grounded theory [GS17] that has been widely used for qualitative analysis [DWAJ$^+$05, Har18]. CCM focuses on finding the similarities and differences between the data extracted from the studies. The CCM method can be used independently [Har18] or combined with summarization. In our case, we applied both CCM and summarization to synthesize the extracted data.

3.4 Results

This section presents an overview of studies and answers our review research questions.

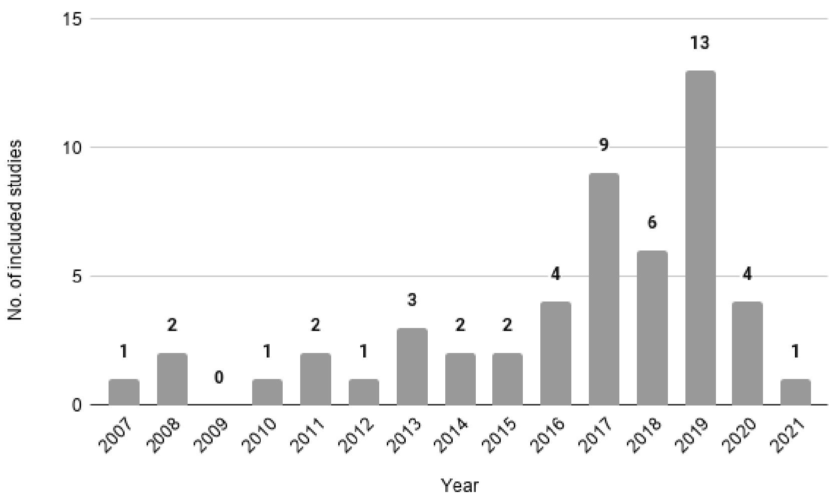


Figure 3.2: Distribution of the 51 selected studies from the beginning of 2007 when the first study was reported to early 2021 (i.e., 30 February 2021) when our search was completed.

3.4.1 General Overview

The 51 studies were categorized according to their main goal(s) (i.e., their contributions), as shown in Table 3.4. A comprehensive description of the studies is provided in Table A.2 in Appendix A. Figure 3.2 shows the distribution of the selected primary studies published between 2007 and 2021. The number of studies published each year fluctuated significantly, with the largest number of studies published in 2019 (13 studies) and 2017 (9 studies). Studies were published in different databases, including IEEEXplore (26 studies), Springer Link (10 studies), ACM Digital Library (7 studies), Science Direct (6 studies), Semantic Scholar (one study), and MDPI (one study). The ratio between conference papers and journal articles was 38:13. Most conference papers were published in the Requirements Engineering Conference (8 studies), Foundation for Software Quality Conference (3 studies), Evaluation and Assessment in Software Engineering Conference (2 studies), and International Conference on Advances in Computing, Communications and Informatics (2 studies). Most journal articles were published by Requirements Engineering Journals (2 studies), Empirical Software Engineering (2 studies), Journal of Systems and Software (2 studies), Information and Software Technology (2 studies), and Journal of Industrial Information Integration (2 studies).

Study goal	**Study ID**	**No. studies**
To automatically annotate requirements	S30	1
To extract and classifying NFRs from the uncommon form of requirements (i.e., textual requirements), such as user comments, commit messages and use case	S8, S10 ,S13, S14 ,S19, S24, S29, S33, S35 , S41, S42, S43, S44	13
To process textual requirement (e.g., feature selection, extraction, extension or representation)	S2, S18, S19, S22, S23, S31, S34, S45, S48	9
To build ML models using new, unused or modified algorithms	S1, S4, S7, S9, S15, S20, S21, S25, S26, S27, S33, S36, S37, S38, S40, S47, S51	17
To compare the performance of different text processing techniques or learning algorithms in NFR classification (i.e., comparison)	S5, S12, S20, S21, S32, S28, S39, S49, S50	9
To use the extracted requirements in another context (e.g., supporting other RE tasks)	S3 , S11, S24, S43	4
To extract and classify security requirements.	S6, S11, S16, S17, S27, S45, S46, S48, S49	9

Table 3.4: The main goals of selected studies; a study could have more than a goal

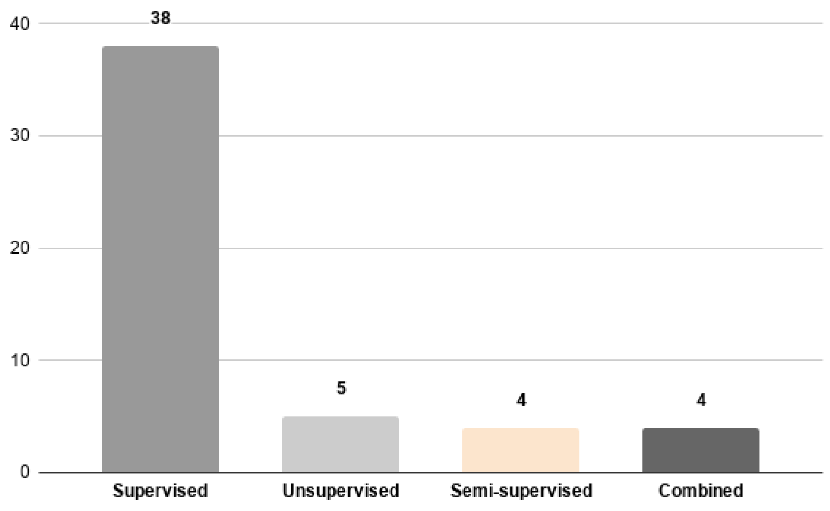


Figure 3.3: Distribution of the 51 selected studies by learning types

3.4.2 ML Algorithms (RQ1)

RQ1: What machine learning algorithms have been applied in the selected studies? Which ones are the most popular?

Out of 51 studies, 38 (74.5%) applied supervised learning, 5 (9.8%) unsupervised, and 4 (7.8%) semi-supervised learning. The rest are either combined supervised with unsupervised (3 studies, 5.8%) or semi-supervised and supervised methods (one study, 1.9%). Figure 3.3 shows the distribution of the studies by learning types.

In detail, a total of 36 various ML algorithms were found in the 51 selected studies: 23 out of 36 are supervised learning (SL) algorithms, 6 unsupervised learning (UL) algorithms and 7 semi-supervised learning (SSL) algorithms, as summarized in Table 3.5. Table 3.5 confirms that supervised learning is the most used type of ML and indicates that SVM is overall the most popular ML algorithm, found in 20 studies (39.2%).

In the following subsections, we introduce the 37 ML algorithms found in our review.

Supervised Learning Algorithms

The twenty-four supervised learning algorithms found in our review are briefly described in this section. These algorithms include seven ensemble learning algorithms, six neural-network-based algorithms and one optimization algorithm. We briefly describe all the SL algorithms in this section, except SVM, NB, KNN, and DT, which are

Type	ML algorithms	Study ID	No. Studies
SL	SVM	S3, S5, S8, S9, S10, S11, S12, S20, S21, S22, S28, S30, S31, S34, S35, S39, S44, S49, S50, S51	20
	NB	S4,S6, S7, S9, S11, S19, S20, S21, S31, S33, S41, S44, S48, S49, S50, S51	16
	DT	S2, S16, S17, S19, S20, S23, S28, S33,S39, S41, S48, S49	12
	KNN	S9, S14, S20, S21, S28, S30, S33, S36, S39, S49, S50	11
	Logistic Regression	S28, S31, S45, S48, S49, S50, S51	7
	Convolutional neural networks	S25*, S26, S38, S42, S46*, S49*, S51*	7
	Multinomial NB	S7, S10, S11, S26, S28, S39	6
	Bagging	S11, S19, S33, S49	4
	Bernoulli NB	S23, S28, S39	3
	Gaussian NB	S28, S39	2
	Recurrent Neural Networks	S40*, S49	2
	Random forest	S30, S31	2
	Stochastic Gradient Descent	S29, S39	2
	Discriminative Multinomial NB	S7	1
	Rule-Based	S18	1
	Artificial Neural Network	S38	1
	Binary Relevance	S44	1
	BERT	S47	1
	Bossting	S49	1
	Random Subspace	S49	1
	RUSBoosted Trees	S51	1
	Extra Tree	S28	1
	Multi-Layer Perceptron	S28	1
UL	LDA	S7, S23, S24, S32, S43	5
	k-means	S15, S23	2
	Hierarchical agglomerative clustering (HAC)	S15, S23	2
	Biterm Topic Modeling	S23	1
	One-Class SVM	S27	1
	Word2vec	S37	1
SSL	Expectation Maximization	S4	1
	Self-Training	S20	1
	Rel-RASCO	S20	1
	RASCO	S20	1
	Active learning	S21	1
	Label Propagation	S28	1
	Label Spreading	S28	1

* Clearly mentioned that deep learning was applied

Table 3.5: ML algorithms applied in the selected studies grouping by learning types: supervised learning (SL), semi-supervised learning (SSL) and unsupervised learning (USL).

already explained in Chapter 2.

Logistic Regression. This is a probabilistic classifier with a decision boundary that predicts probability values which are then mapped into two or more discrete classes [KJMH$^+$19]. It uses the logistic function (i.e., logistic sigmoid function) to estimate a probability value of a discrete class (i.e., dependent variable) based on a specific feature (i.e., independent variable) [MBW$^+$19].

Rule-Based (RB). This is similar to DT where the dataset is modeled by collections of rules. However, RB classifiers allow overlaps in decision space, while DT uses a strictly hierarchical approach [KZP06]. The left side of these rules consists of conditions, while the right side contains the classes. These rules are derived from the dataset [KZP06].

Multinomial NB (MNB), *Bernoulli NB*, *Gaussian NB* are variants of classical NB classifier (described in section 2.3). The main difference between these classifiers is based on the assumption regarding the distribution (i.e., representation) of features [Xu18]. Instead of representing a document as a set of features doc= $\{f_1, f_2, .. f_{doc\_length}\}$, such as classical NB classifier, MNB represents an input document as a frequency vector, where each element represents a word/phrase frequency in the document. Bernoulli NB represents each document as a vector of binary features; each element has either 0 or 1 (1 means the document contains a specific word, while 0 means does not). GNB is used when the features have continuous values (e.g., TF-IDF). In this case, a typical assumption is that continuous values associated with each class are represented in Gaussian distribution (i.e., normal distribution) [Xu18]. Discriminative Multinomial Naive Bayes (DMNV) is a discriminative version of MNB that adds a discriminative element to the frequency estimate in order to discriminatively compute the appropriate frequencies from the data [SZLM08].

Artificial Neural Network (ANN). This is a biologically driven model that is inspired by the operation of biological neural networks. This model builds a network with nodes (known "neurons") and lines connecting the nodes (see Figure 3.4a). The neurons are segmented into three layers: the first layer represents the inputs, and the last layer contains the outputs. The layer between them is called a "hidden layer," which performs transformations of inputs into the output layer through neurons [Zho12]. NNs with many hidden layers (usually more than one [KG18] or two [ZWL18]) are known as deep NNs (or deep learning), whereas those with fewer layers are known as shallow NNs [KG18]. Each node in the hidden and output layer is a functional unit with inputs and outputs; the output value is obtained by doing some mathematical operation (i.e., activation functions) using the inputs. Each node in the input layer corresponds to one element of a feature vector [Zho12]. All nodes, except those in the output layer, are connected to all the next layer nodes. A connection line represents the weight of the connection between two distinct neurons, reflecting the importance of the communication between neurons in the network.

Multi-Layer Perceptron (MLP). This is an example of ANN, which contains one or more hidden layers for enhancing classification accuracy. The neurons in MLP are interconnected unidirectionally, from the input layer to the output layer (i.e., feedforward neural network) [TM18].

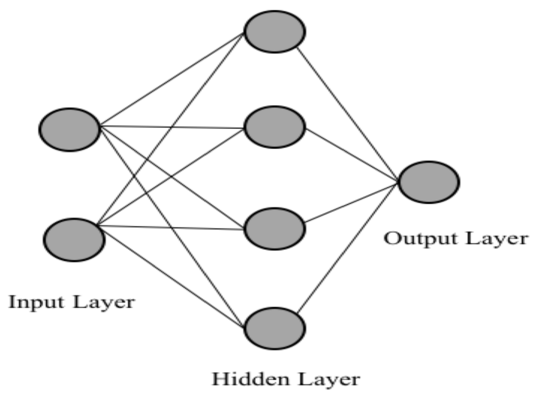
Convolutional neural networks (CNNs). CNN is a type of NN that is commonly applied to image processing [KJMH+19]. It is different from other NN by having hidden layers called "convolutional layers". These layers take a batch of the dimensions (spatial structure) and moves it through the network instead of taking a single dimension (Figure 3.4b). In NN, each neuron is fully connected to all neurons in the previous layer; however, CNN layers do not connect to all but a small region of neurons.

Recurrent Neural Networks (RNN). It is another type of NN architecture used for temporal (sequential) data (i.e., data that requires past experience to predict new outcomes). The main difference between RNN and other NN architectures is that it has a feedback loop, which means it feeds previous time steps into the current step (Figure 3.4c). Thus, each node in RNN acts as a memory cell during the computation to allow for the comparison (or use) of the previous value within computing the current one [Swa19].

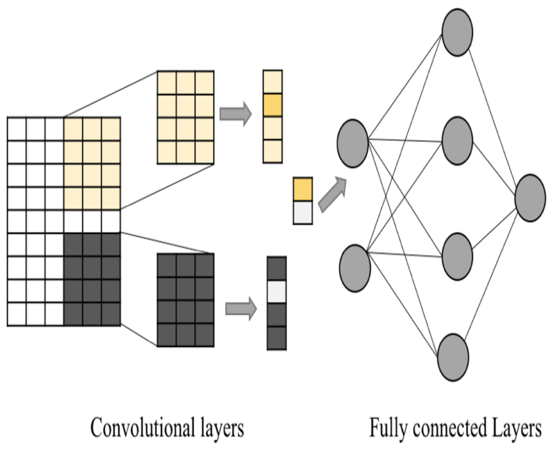
Bidirectional Encoder Representations from Transformers (BERT). BERT is a language transformation model that learns the deep representation of texts by considering both the left and right contexts (“bidirectional”). The fundamental idea behind BERT is to overcome NN models’ key limitations (i.e., requiring a large dataset and being time-consuming) by using a pre-trained model for two steps: pre-training and fine-tuning. In pre-training, a model is trained using a large unlabeled dataset and an NN algorithm. During fine-tuning, the pre-trained model is modified by changing the output layer using fine-tuning methods and a small labeled dataset [DCLT18].

Binary Relevance (BR). BR is the most intuitive solution for handling a multi-labeling classification task [ZLLG18]. This algorithm builds a binary classifier for each class; the outcome of each classifier is used to determine the final prediction(s).

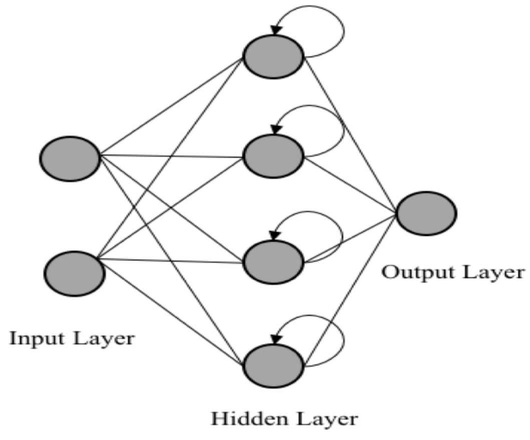
Bagging algorithm (or bagged tree). This is an ensemble-based model, a model



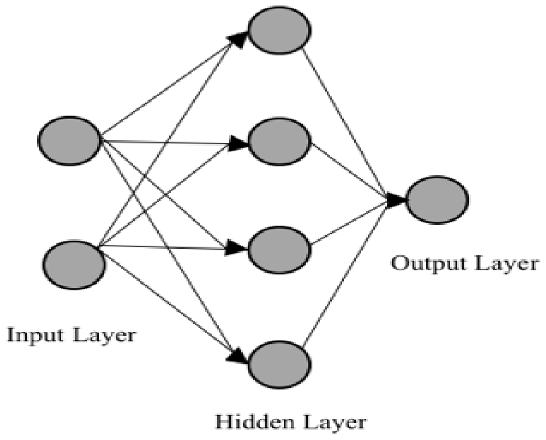
(a) ANN



(b) CNN



(c) RNN



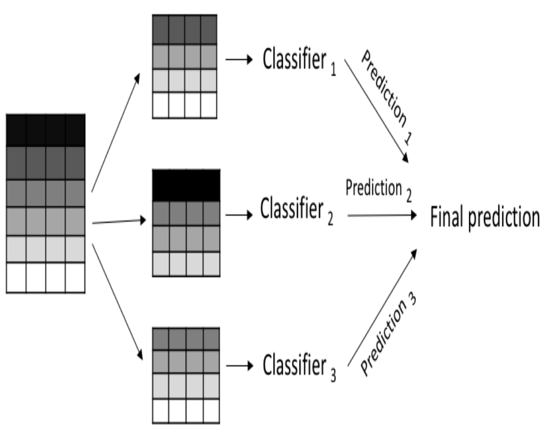
(d) MLP

Figure 3.4: Illustrations of different types of NN architecture

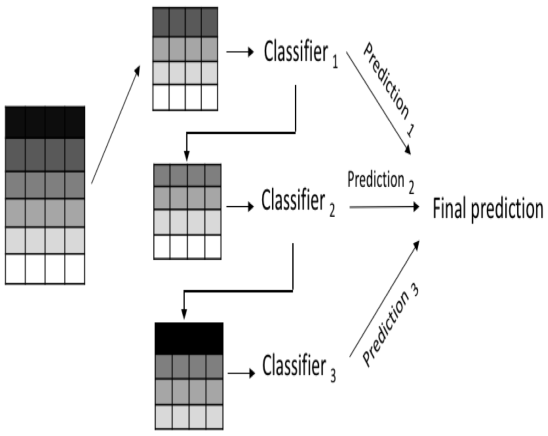
that builds by combining multiple ML models to obtain optimal predictions (see Figure 3.5a). Bagging algorithm samples subsets of training instances (rows) to be fed to different ML models. A new instance's classification decision is then made based on the majority voting of all models results [Zho12].

Subspace ensemble method (also known as attribute bagging [BGOQ03]). This method randomly selects a subset of features before applying an ML algorithm, where each classifier is trained in a different feature set [ZZYZ21]. The final prediction is determined by majority votes.

Random forest (RF). This is an extension of the Bagging algorithm by integrating features sampling (columns) to the instances sampling (rows) (i.e., a combination of bagging and subspace ensemble methods) [Zho12]. RF is implemented by randomly



(a) Bagging



(b) Bossting

Figure 3.5: Illustrations of ensemble classifiers

selecting a subset of features and then carrying the conventional split selection procedure of bagging within the selected features [Zho12].

Extremely Randomized tree (Extra Tree). This algorithm is similar to RF in terms of combining multi trees and uses majority voting to determine the final decision. However, the difference consists in the use of the whole training set in training each classifier, instead of samples [GEW06]. Besides, whereas RF uses a specific measure to determine an optimal split point for the decision tree (e.g., Gini index) [EBdS20], the Extra tree selects such a point randomly [GEW06].

Boosting (or Boosted Trees). Boosting is similar to the bagging ensemble method. However, it trains in sequence with a focus on miss classification instances (Figure 3.5b). For example, data that misclassified in the previous classifier is used by the next classifier. The majority votes of the models from each iteration are used to make the final classification decision [Zho12].

Random undersampling boosting tree (RUS Boosted Trees). This is an extension of the boosting method, incorporating random undersampling with boosting method [SKVHN09]. The under-sampling strategy is mainly proposed to handle the class imbalance problem by removing instances/samples from the majority (large) classes to obtain a balanced classes distribution. We will go back to this strategy in Chapter 4.

Stochastic Gradient Descent (SGD). This is an optimization algorithm that uses only a part of the data to optimize lost function (errors between the actual values and predicted value) on training data [GBC16]. This optimization algorithm can be used with different ML algorithms, including (linear) SVM, logistic regression, and deep learning algorithms [GBC16]. S39 clearly explained using SVM with SGD, but S29 did not.

Un-Supervised Learning Algorithms

The six unsupervised learning algorithms identified in the selected studies are briefly described as follows:

Latent Dirichlet Allocation (LDA). This is a generative probabilistic model used to automatically identify topics that documents contain based on certain probabilities [ZXY+17, BNJ03].

K-means. This is an iterative algorithm, which classifies documents randomly into certain numbers of clusters (K). K-means work iteratively to assign each data point in the space into the nearest single cluster of K-clusters [AKG+17, MW16].

Hierarchical Agglomerative Clustering (HAC). This is an iterative algorithm merging the similar elements of a dataset into a large cluster until the entire dataset forms a single cluster. In contrast to K-means, HAC does not need to determine the number of clusters up front. This algorithm includes three categories: complete linkage, single linkage, and average linkage. The difference between these categories is the method used to measure the distance between two clusters for each iteration. For example, single linkage uses the most similar pair of elements, while complete linkage chooses the most dissimilar pair of elements; average linkage defines the distance as the average between the pairs in the data element [AKG+17, MW16].

Biterm Topic Modelling (BTM). This approach identifies topics by modeling word-to-word co-concurrence patterns [YGLC13]. Such patterns are called biterms, which are unordered pairs of words that appear frequently together in a dataset [YGLC13].

One-Class SVM . This is similar to the SVM algorithm; however, it focuses on one class. Instead of using a hyperplane to separate two classes of instances, it determines a boundary to encompass all instances of only one class [MY01]. New observations are classified based on their position in relation to the boundary (inside or outside). This algorithm is considered an unsupervised learning process because only instances of one class are used to train a model. The other class can be predicted during the testing. For example, S27 used general (non-domain specific) descriptions of security requirements to classify requirements as security and non-security requirements.

Word2vec is an unsupervised learning model that does not require labeled data. It uses an large unlabeled dataset to learn a vector representation where similar words appear close to each other, and dissimilar words are located far from each other. The similarity distance measures (e.g., cosine) is then used to measure the similarity between two words (i.e., the distance between two words). S37 measured the similarity between indicator keywords of each NFR category and requirement statements to identify the type of NFRs.

Semi-Supervised Learning Algorithms

The seven SSL algorithms identified in the selected studies are briefly described as follows:

Expectation Maximisation (EM). This is an iterative approach using probabilistic functions for maximum likelihood estimation in problems with missing data [NMM06]. Unlabeled documents are handled as missing data due to a lack of labels.

Self-training. It is incremental semi-supervised training [RHS05], where the labeled datasets are used to train a supervised classifier, which then is used to classify unlabelled data. The most confidently labeled data are added to the training set, and the classifier is re-trained [Zhu05].

Active learning. It is aimed to achieve high accuracy by choosing the labeled dataset carefully, which will derive high confidence/accuracy. The instance which has the least confidence will be manually labeled and added to the training set [LHG+18].

RAndom Subspace Method for Co-training (RAS-CO). It is used both a random subspace method and a co-training method. Co-training consists of two classifiers that are trained with different parts of a labeled dataset [Zhu05]. RAS-CO basically selects a random subspace from the featured spaces and trains the classifier of each subspace. The idea behind this algorithm is that each classifier is sensitive to different features and can complement the other classifier [WLZ08].

Relevant Random Subspace Method for Co-training (Rel-RASCO). This is similar to RAS-CO; however, the aim of this algorithm is to select a random subspace that contains relevant features' subspaces [DHR17].

Label Propagation is a graph-based algorithm that builds a graph of connected nodes where nodes are data points (labeled and unlabeled), and edges represent the similarity between points. Labeled data points are used to iteratively label all nodes by propagating information through the graph using the edges [alm06]. *Label Spreading* is similar to label propagation; however, it used a normalized graph (i.e., a normalized weight of the edge) to be more robust to noise [alm06].

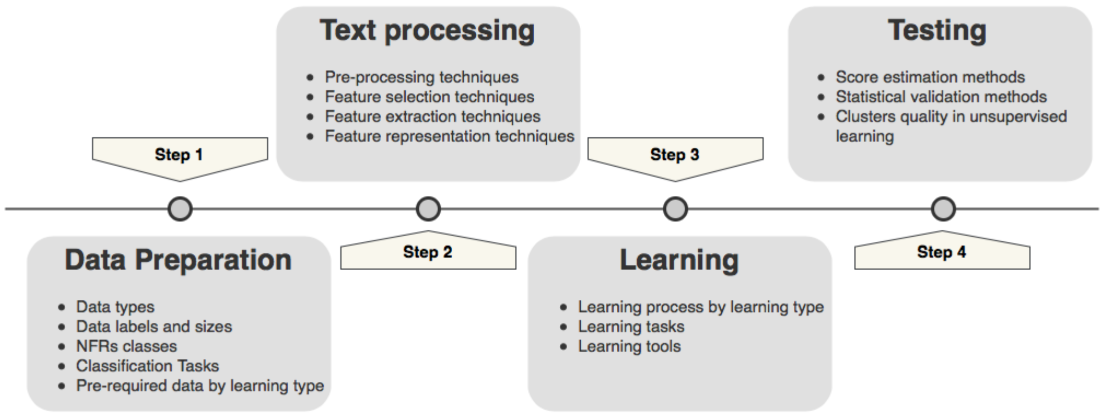

Step 1
Data Preparation
Data types
Data labels and sizes
NFRs classes
Classification Tasks
Pre-required data by learning type
Step 2
Text processing
Pre-processing techniques
Feature selection techniques
Feature extraction techniques
Feature representation techniques
Step 3
Learning
Learning process by learning type
Learning tasks
Learning tools
Step 4
Testing
Score estimation methods
Statistical validation methods
Clusters quality in unsupervised learning

Figure 3.6: A general process of applying ML algorithms to classify NFRs, including key topics to be discussed in RQ2

3.4.3 Process of Using ML Algorithms to Identify NFRs (RQ2)

RQ2: What are the processes that the reported ML-based approaches follow to identify and classify NFRs in requirements documents?

Our analysis of the 51 selected studies has revealed a general process pattern for applying ML-based methods to identify and classify NFRs from a textual document. This process consists of four major steps: data preparation, text processing, learning, and testing. Figure 3.6 shows these steps and indicates the key topics (techniques or strategies) that have been frequently discussed in the context of each step. The four steps are similar to those applied for supervised ML in text classification (Chapter 2). Therefore, in this section, we will not redescribe them in detail. Instead, we report how these steps have been implemented in NFRS classification, common techniques, and key observations.

Dataset Preparation

Dataset Types. Dataset collection is a very initial required step for building an ML model. Many datasets have been used by the selected studies. These datasets came from three main sources: 1) academic—where the requirements were written by academic students and researchers, 2) industrial—where they were written to describe or simulate industrial products and 3) untraditional—where they were written by end-users as App reviews, Q&A posts, user stories or user requests in open-source communities (e.g., sourceforge.net). The academic datasets were used the most in the selected

Dataset Type	Dataset Name	Study IDs	No of Studies	Total
Academic	PROMISE *	S1, S2, S4, S5, S8, S9, S13, S16, S18, S22, S23, S25, S26, S28, S29, S31, S34, S35, S36, S37, S38, S39, S40, S47, S48	25	25
	Relabeled PROMISE*	S34, S47	2	
	Expanded Promise*	S50	1	
	Concordia	S8, S18	2	
Industrial	Multiple domains	S15, S34*, S36, S46, S51	5	16
	Healthcare Domain	S9*, S11*, S36, S37*	4	
	SecReq*	S6, S26,S27, S48, S49	4	
	Automotive domain	S10, S12	2	
	Archive file formats	S32	1	
	Requirements definitions/standards	S27	1	
Untraditional	Agile & open-source software development	S7, S21,S41,S42, S43, S45	6	11
	App reviews	S19*, S20, S22, S33*, S44*	5	
	Q&A website	S24	1	

* Available Datasets, a more detailed description is provided in Table 3.7

Table 3.6: Overview of datasets used in the selected studies

studies (25 studies), followed by the industrial dataset (16 studies) and then the informal dataset (11 studies). Table 3.6 shows the studies' distribution according to the dataset type, where some studies use multiple dataset types (e.g., S8, S18, S22, S26, S36, S37, and S48). Four studies have not been included in the table (i.e., S3, S14, S17, S30), as there is not enough information about a dataset or dataset domain.

Although there are many different datasets used by the studies, few are publicly available (see Table 3.7). As the table shows, PROMISE and SecReq datasets were used most frequently, and both were provided for the RE'17 Data Challenge[2]. Besides the datasets mentioned in Table 3.7, there are datasets that are no longer available (e.g., Concordia used by S8 and S18), and datasets that are partially available—the original text is publicly available, but the labeled version is not. For example, user stories collection [3] used by S42, security requirements [4] by S45, and NFRs [5] in S51.

[2] `http://ctp.di.fct.unl.pt/RE2017/pages/submission/data_papers/`, Last accessed February 2021

[3] `https://data.mendeley.com/datasets/7zbk8zsd8y/1`, last accessed February 2021

[4] `https://goo.gl/qr8Y4W`, last accessed February 2021

[5] `http://fmt.isti.cnr.it/nlreqdataset/`, last accessed February 2021

Dataset Name	**Description**	**Introduced by**	**Reused by**	**URL**
PROMISE	NFR dataset that includes 625 requirements and 12 classes and; 255 FRs and 370 NFRs labeled in 11 different categories. These requirements are collected from 15 software development projects and manually annotated by MS students at DePaul University.	S1	S2, S4, S5, S8, S9, S13, S16, S18, S22, S23, S25, S26, S28, S29, S31, S34, S35, S36, S37, S38, S39, S40, S47, S48	`https://doi.org/10.5281/zenodo.268542`
Dalpiaz et al. data set	It consists of 877 requirements collected from multiple domains, including PROMISE, and labeled as functional or quality aspects.	S34	S47*	`https://github.com/explainable-re/RE-2019-Materials`
PROMISE-exp	It comprises 47 requirements documents, 15 of which belong to the original PROMISE dataset, whereas 34 are collected from requirements documents over the Internet. Altogether, there are 444 FRs and 525 NFRs assigned to the same 11 categories as in PROMISE.	[LVC$^+$19]	S50	`https://tinyurl.com/PROMISE-exp`
SecReq	A security requirements dataset consisting of 511 requirements collected from three industrial specifications: Common Electronic Purse (ePurse), Customer Premises Network (CPN), and Global Platform Specification (GPS). The requirements are classified into three classes: security, non-security, and unknown.	S6	S26, S27, S48, S49	`http://ctp.di.fct.unl.pt/RE2017//downloads/datasets/SecReq.zip`
Slankas and Williams Dataset	A multi-labels NFRs dataset consists of 11876 sentences,15 classes, collected manually from 11 documents related to the electronic healthcare domain and PROMISE.	S9	S37*	`https://github.com/RealsearchGroup/NFRLocator`

Dataset Name	**Description**	**Introduced by**	**Reused by**	**URL**
Riaz-Dataset	A security requirements data set consisting of 10963 sentences collected from 6 different Healthcare domains. These sentences are classified according to six security objectives.	S11	N/A	`http://go.ncsu.edu/securitydiscoverer/`
User reviews	1278 user review sentences collected from two Apps iBooks (iOS) and WhatsApp (Android). These reviews are classified into five NFRs classes.	[WLL18]	S19, S33	`https://tinyurl.com/y7tj9lkq`
Jha & Mahmoud Dataset	6000 user reviews collected from different apps in the Apple App Store. 2346 of these reviews labeled with at least one out of four NFR classes. The remaining reviews are labeled as Mis, which means, according to the authors, miscellaneous.	S44	N/A	`http://seel.cse.lsu.edu/data/emse19.zip`

* Used only a part of the available database

Table 3.7: An overview of the publicly-available datasets used by the selected studies

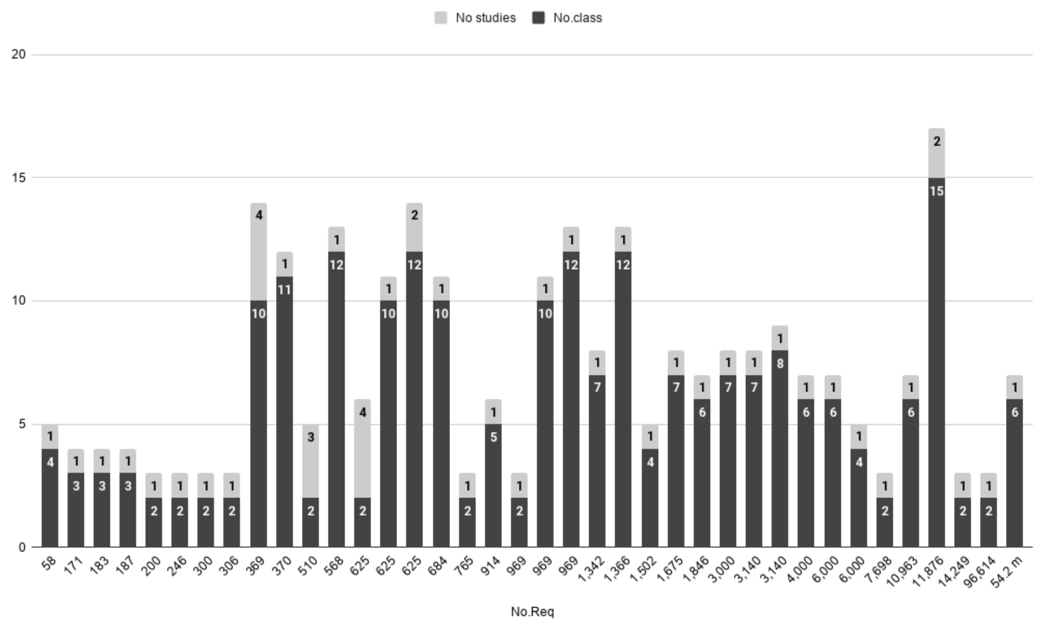


Figure 3.7: Distribution of studies by dataset size and a number of classes

Data Labels and Sizes. The number of requirements in datasets varies greatly in the selected studies. For example, S16 used 58 requirements (with four categories) to train a ML classifier, and S9 use 11876 (with fifteen categories). Figure 3.7 shows the distribution of classes according to the number of requirements and classes. More details about the studies IDs and class names are provided in Table A.3. A significant finding in S1 is that an ML classifier performs better with a large training set in classifying NFRs.

NFRs Classes. In total, there are 46 different NFRs classes used by the studies: Figure 3.8 shows the frequency of the 46 NFRs classes in the selected studies, indicating that Security is the most frequent class, followed by Usability and Performance. The classes in Figure 3.9 are grouping based on their name. For example, Security and Authorization- Authentication are separated classes. Figure 3.9, on the other hand, shows the frequency of classes by grouping them as security-related, performance-related, and usability-related requirements. The figure shows that Security is the most frequently used group, followed by Performance, and finally, Usability. Table A.4 in Appendix A provides more details by showing the NFRs classes for each study.

Classification Tasks. The datasets used for the selected studies were applied to five different classification tasks:

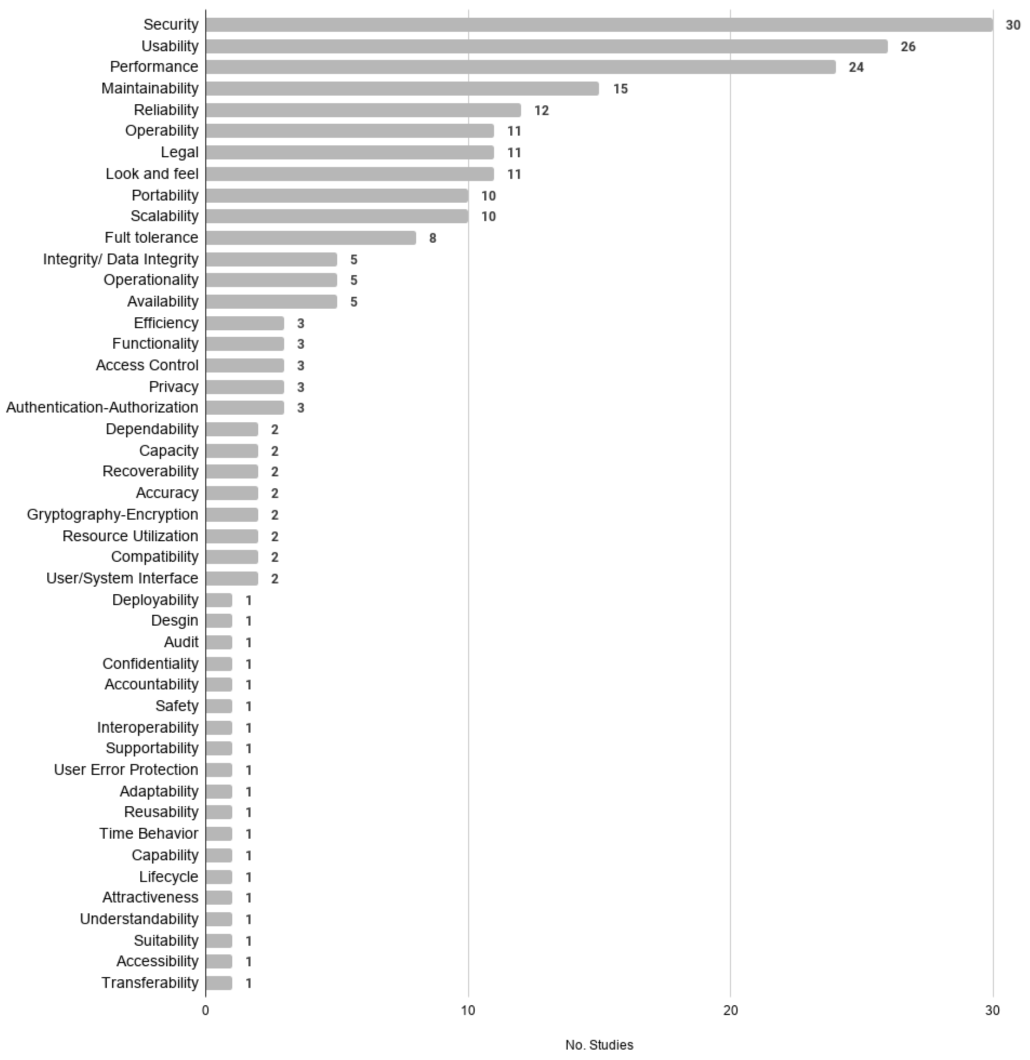


Figure 3.8: Distribution of the selected studies by each NFR type

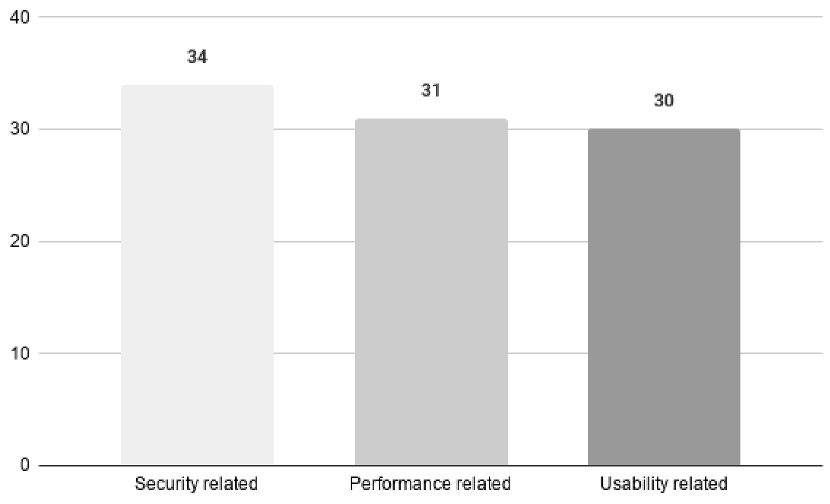


Figure 3.9: Distribution of the selected studies by three groups of NFRs

1. NFR identification—classifying requirements into two classes: NFRs and non-NFRs (e.g., FRs)
2. NFR classification—classifying NFRs into non-functional classes in which the dataset is annotated with only NFR classes (e.g., security, usability, and legal).
3. NFR identification and classification—a combination of the two aforementioned tasks in which the dataset is annotated according to NFRs and other classes (e.g., FRs).
4. Sub-class NFR classification—classifying NFRs into further categories (e.g., classifying security requirements based on security objectives).
5. Identifying a single class of NFRs by, for example, annotating requirements into security or non-security.

Table 3.8 shows the frequency of studies applied to each task. As the table shows, some studies performed more than one task (i.e., S4, S22, S26, S31, S47, and S50). The classification of NFRs into NFR classes is the most commonly addressed classification task.

Pre-required Data by Learning Type. As there are different types of ML algorithms applied in NFRs classification (i.e., supervised and unsupervised learning), dataset preparation is slightly different in each type. For example, supervised learning methods require a set of NFRs correctly classified according to their categories. To

Classification Task	Study ID	No Studies
Classifying NFRs into NFR classes	S3, S4, S5, S7, S13, S21, S23, S24, S28, S29, S30, S35, S36, S38, S40, S42, S44, S47, S50, S51	20
Identifying/detecting NFRs	S2, S4,S20, S22, S23, S26, S31, S32, S34, S41, S43, S47,S50	13
Identifying and classification NFRs	S1, S4, S8, S9, S14, S15, S19, S25, S33, S37, S39, S50	12
Identifying a single non-functional class	S6, S22, S26, S27, S31, S45, S46, S48, S49	9
Classifying NFRs into NFRs sub-classes	S11,S16,S17, S18	4

Table 3.8: Distribution of the selected studies by classification tasks addressed by the studies

do so, datasets were labeled manually in all of the selected studies (apart from S30). Thus, two possible threats are posed: the subjectivity of the labeling process (i.e., annotations) and incorrectly labeling. The key method applied by the studies to overcome such threats is to annotate requirements separately by different annotators and used one of the following validation methods:

- Percent Agreement (two studies): A simple method that computes the percentage of agreement between annotators (i.e., the number of agreements in observations divided by the total number of observations).

- Kappa measurements (8 studies): Statistic measures are widely used to assess the reliability of agreement between raters on nominally scaled data. Examples of kappa measures are Cohen Kappa [Coh60] which measures the agreement between two raters (S7, S10, S12, and S45), Fleiss' kappa [Fle71] assess the agreement between multiple raters (S24), and Randolph's kappa [Ran05] (S9) is for multiple annotators with no restrictions on the distribution of instances into different categories.

- Krippendorff's (alpha) [Kri11] (one study): A reliability coefficient used to measure the agreement among annotators.

- Spearman correlation [CF14] (one study): A correlation coefficient used to measure the strength of a non-linear relationship between two variables.

- Majority voting (four studies): It is to select a label used the most frequently in annotating a single requirement.

Data validation methods	No Studies	Studies IDs
Annotators reviewing/discussion	9	S8, S10, S12, S19, S33, S34,S42, S44, S45
Kappa measurements	8	S7, S9, S10, S11, S12, S24, S45
Majority voting	3	S21, S44, S51
An additional annotator for resolving the conflict	3	S11, S19, S34
Percent Agreement	2	S19, S33
Krippendorff's alpha	1	S34
Spearman correlation	1	S7
Annotators' experience and confidence	1	S51

Table 3.9: Distribution of the studies by data validation methods

- An additional annotator for resolving the conflict (three studies): It is to resolve the disagreements in observation by an additional annotator who makes the final decision.
- Annotators reviewing/discussion (9 studies): It is to discuss the disagreements between annotators until a consensus is achieved.
- Annotators' experience and confidence (one study). It is to use an annotator's experience and confidence in resolving conflicts. For example, S51 asked each annotator about their experience and confidence in annotating each requirement. In case of conflict, the labels provided by high experienced annotators or with high confidence levels were selected.

Table 3.9 shows the frequency of each data validation method. It can be seen from the table that some studies (e.g., S11, S19, S34, S51) use more than one method, in which one method determines the agreement level, the other resolves the conflict. For example, S11, S19, S34 reported the agreement score, and resolved the conflict (e.g., when kappa < a threshold) using other methods (e.g., discussion). S51 applied majority voting, but in cases of disagreement, another method is used (annotators' experience and confidence).

The unsupervised learning algorithms are simply about portioning text documents into different parts or categories. Thus, in general, the unsupervised learning algorithms do not require any prerequisites where a given document is classified based on its content without any pre-defined categories. However, some of the studies used some pre-defined data in NFRs classification. Using the data is not necessary to perform the

Pre-defined data type	**No Studies**	**Studies IDs**
Pre-defined wordlists	6	S7, S15, S24, S32, S37, S43
Requirement definition/description	1	S27

Table 3.10: The distribution of studies by predefined data type to build unsupervised NFRs classifier

unsupervised learning algorithms; however, it improves the classification performance of the NFRs. For example, S23 used unsupervised learning without pre-defined data and got the worst performance than other studies that used pre-defined data. Two types of pre-defined data were used by the studies applying unsupervised learning algorithms, which are as follows:

- Pre-defined wordlists. Several domain-independent word lists have been derived from different sources such as ontology for software quality measurement (S7), ISO/IEC 9126-1 Software Quality Characteristics (S7, S24, S43), keywords in Cleland-Huang et al. work [CHSZS07] (S37), simple words that containing NFR categories and their representative words (S15), or NFRs characteristics—e.g., the behavior of systems such as speed and size (S32).

- Requirement definition/description. A domain-independent definition of requirements has been used to train an unsupervised classifier. For example, S27 uses a description of weaknesses in the Common Weakness Enumeration database [6] in training an unsupervised classifier (one-class SVM).

Table 3.10 shows the distribution of studies based on pre-defined data used in building unsupervised NFRS classifiers. The table indicates that keywords are most frequently used with unsupervised classifiers.

Text Processing Step

This step consists of four different sub-steps: pre-processing, feature selection, feature extraction, and feature representation.

Text Pre-Processing. This step takes a textual requirements document as input and then applies different natural language processing (NLP) techniques to pre-process the input document. A total of 20 NLP techniques were found in the selected studies. Table 3.11 shows which study used which NLP technique. These 20 techniques are briefly summarized as follows.

[6]`https://cwe.mitre.org/data/definitions/416.html`, last accessed March 2021

- Tokenization: splitting a sentence into a series of words [KBLK10].
- Sentence splitting: splitting a text into a set of sentences.
- Stop-words removal: removing auxiliary verbs (be, do and have), conjunctions (and, or) and articles (the, a, and an) in sentences [KBLK10]
- Slang and abbreviation transformation: converting slang words or abbreviations into their basic form (e.g., change "don't" to "do not").
- Spelling correction: fixing typos error is a requirements specification. Many techniques are proposed in the literature (e.g., hashing-based and context-sensitive spelling correction techniques [KJMH+19]). However, the studies that applied this technique (S36 and S41) did not clearly show how this technique is performed.
- Lowercase conversion: converting all letters into lowercase.
- Noise removal: removing unnecessary characters (e.g., punctuation and special characters). It is argued that these characters can be detrimental to classification algorithms [KJMH+19]. Fourteen studies explicitly indicated that they applied noise removal, while one study (S26) implicitly showed that. This study discarded such noise by adding space around punctuation, which was then discarded using the word embedding technique.
- Digits removal: removing digits from requirements (5 studies) or converting numbers to words (e.g., converting 3 to "three") in S40.
- Restricting the size of words: removing words with a specific number of characters (e.g., less than three characters).
- Compound splitting: decomposing words into compound parts. This technique was only applied by the studies (S10 and S12) that classified requirements written in German, which may have more compound words than English.
- N-gram: separating each given string into subsequent N items, where the items can be words, letters, or statements [CT+94].
- Stemming: reducing inflected (or sometimes derived) words to their word stem, base or root form. For example, the words 'goes', 'gone' and 'going' will map to 'go', and the word 'mice' will map to 'mice'.

- Lemmatization: determining lemma (the infinitive form of verbs and the singular form of nouns and adjectives) of each word. For example, the words ‘goes’, ’gone’ and ‘going’ will map to ‘go’, and the words ‘mice’ will map to ‘mouse’.
- Part of speech (POS) Tagging: assigning tags to words based on their part of speech (e.g., noun, verb, and preposition) in a sentence [CGC10].
- Dependency parsing: identifying the grammatical relationships between words in a sentence to recognize the important parts and ignore unimportant parts in the sentence [DMM08].
- Noun chunker: using the structural properties of a sentence to extract noun phrases (NP).
- Named entity recognition (NER): detecting name entities in documents and classifying them to pre-defined categories, such as date, time, percent, money, and cardinal [AKG$^+$17].
- Temporal tagging: recognizing and normalizing temporal expression (e.g., time and duration in S23) [AKG$^+$17].
- Thematic role tagging: annotating words according to their semantic roles (e.g., Agent, Theme, and Instrument).
- Regular expressions: matching a defined expression that describes string patterns. This expression contains specialized notations, such as the character ($^\wedge$) means ‘not’ [AKG$^+$17].
- Words augmentation: extending words with more related words. This technique is used to enhance the similarity between requirements (S37) and handle the short text classification (S19).

Feature Selection. This step is to select features that have high importance or weight (i.e., most relevant) to predict output variables. It is applied by 27 studies that used 9 different techniques, as shown in Table 3.12. In the following, we summarize the 9 techniques.

- Term frequency: ranking words, phrases, or word characteristics (e.g., POS tags) according to their frequency in a given document.

NLP technique	**No. stud-ies**	**Studies IDs**
Stop-words removal	31	S1, S2, S3, S4, S5, S7, S9, S13, S14, S15, S16, S17, S19, S22, S24, S26, S28, S29, S30, S32, S33, S35, S36, S37, S38, S39, S40, S41, S44, S50, S51
Stemming	19	S1, S2, S4, S5, S8, S15, S16, S17, S18, S20, S28, S30, S32, S33, S35, S36, S38, S41, S44
Tokenization	17	S8, S16, S17, S18, S20, S21, S23, S24, S30, S31, S36, S37, S39, S40, S41, S44, S51
Noise removal	15	S4, S7, S13, S20, S22, SS26,28, S29, S31, S35, S36, S37, S39, S40, S51
Lemmitization	14	S9, S13, S14, S15, S19, S21, S22, S29, S31, S33, S37, S40, S50, S51
Lowercase conversion	13	S4, S7, S20, S34, S26, S29, S31, S36, S38, S39, S41, S44, S51
POS tagging	11	S2, S3, S9, S14, S18, S22, S23, S30, S34, S37, S51
Digits removal	6	S4, S13, S20, S31, S40, S51
N-gram	5	S10, S12, S29,S35, S41
Named entity recognition	4	S9, S14, S18, S23
Sentence splitting	3	S8, S18, S19
Dependency parsing	3	S9, S34, S48
Spelling correction	2	S36, S41
Words augmentation	2	S37, S19
Slang and abbreviation transformation	2	S19, S26
Restricting the size of words	2	S5, S20
Compound splitting	1	S10
Noun chunker	1	S18
Temporal tagging	1	S23
Thematic role	1	S18
Regular expressions	1	S23

Table 3.11: Distribution of the studies per NLP technique used for text pre-processing

- TF-IDF: weighting words based on the frequency that a word appears in a document inverse of the number of times the word appears in the corpus.
- Information Gain: measuring the features' importance based on the presence and absence of each feature in each category within training documents
- CHI: using the Chi-Square test to assess the independence between two variables (feature and classes).
- ML-based technique: computing the importance of a feature using machine learning algorithms (e.g., tree-based classifiers), where a high value of a feature means that the feature is relevant to a specific class.
- LDA (unsupervised-based technique): extracting main topics (themes) within texts. This technique is also used for the unsupervised classification of NFRs (section 3.4.2).
- Pre-defined Keywords: using a list of predefined keywords (either domain-specific or non-domain-specific) as features. This includes replacing existing features with a list of predefined keywords (e.g., keywords matching in S48).
- Cleland-Huang et al.' technique [CHSZS07]: a probability-based feature selection technique was proposed by Cleland-Huang et al. (S1) to select keywords (known as indicator terms). S45 reuses this technique.
- Linguistic heuristics: defining rules at the syntactic or semantic level to select features. For example, selecting specific POS groups (e.g. adjectives and adverbs) or particular order of dependency parsing tags (e.g., adverb followed by a noun).

Feature Extraction. This step creates (i.e., synthesize) new features by analyzing a corpus. Three different methods have been applied to extract features in the studies. These methods are illustrated in Table 3.13 and defined below.

- Sentiment analysis: using the result of sentiment analyzing the requirements (i.e., sentiment score) as features by assuming that different NFRs are expressed using different sentiments.
- ML results: using the results of ML models as features. Each model either has a different type of features (S45) or a different representation (S51). The results

Feature selection technique	**No. studies**	**Studies ID**
Term frequency	8	S2,S3, S7,S23, S27,S34, S35, S50
TF-IDF	5	S9, S20, S33, S41, S50
Pre-defined keywords	4	S21, S29, S44, S48
Information Gain	3	S5, S11, S16
Cleland-Huang et al.' technique	2	S1, S45
Linguistic heuristics	2	S37, S48
CHI	2	S19,S50
ML-based techniques	1	S22

Table 3.12: Distribution of the studies per feature selection technique

Feature extraction	**No. studies**	**Studies ID**
Heuristic property	7	S2, S21, S22, S23, S34, S45, S48
ML-based results	2	S45, S51
Sentiment analysis	1	S44

Table 3.13: Distribution of the studies per feature extraction technique

of these methods are then fed to a ML model to provide the final classification decision.

- Heuristic property: manually or automatically analyzing the characteristics of requirements (e.g., average a specific POS tag, text length, and specific syntactic patterns). These characteristics are used as features, representing as columns of feature space.

Feature Representation. This step converts the textual features into a format that ML algorithms can understand (i.e., set of vectors). Seven different techniques have been used in the selected studies: Table 3.14 shows the frequency of using each technique. The first four techniques (i.e.,TF, TF-IDF, word2vec, BERT) have been also used for feature selection (see Section 3.7) or requirement classification (see section 3.4.2). The remaining 3 techniques are described in the following:

- Boolean (binary representation) represents a feature based on its appearance. The value of an element vector is 1 if a document contains the feature and 0 if it does not.

- Doc2vec is similar to word2vec in terms that it is a NN-learning-based method. However, it learns numerical representations of documents, instead of words.

Feature representation	**No. studies**	**Studies ID**
TF-IDF	13	S4, S16, S27, S28, S29, S30, S31, S33, S36, S39, S41, S50, S51
TF	13	S15, S19, S20, S21, S23, S29, S31, S34, S38, S39, S44, S50, S51
word2vec	7	S25, S62, S31, S40, S46, S49, S51
Boolean	4	S5, S21, S23, S48
BERT	1	S51
Doc2vec	1	S31
Sparse Composite Document Vectors (SCDV)	1	S31

Table 3.14: Feature representation techniques used in the selected studies for text processing

- Sparse Composite Document Vectors (SCDV) is similar to word2vec, but it has the ability to distinguish the semantic meaning of words when the same words have a different meaning. Thus, it incorporates word2vec with a clustering algorithm called Gaussian Mixture Model (GMM).

Table 3.14 shows that TF-IDF is the most frequently used method for requirements representation. Despite that, S5 argued that Boolean is better for representing requirements, as the requirements are short and do not need complex representation. S26 argued that using word embedding (i.e., word2vec) helps to overcome the challenge of having a small dataset with NN models.

Learning

Learning Process by Learning Type. This step involves the application of ML algorithms to build the NFRs model using the processed text. Each type of ML learning has a slight difference in the learning step. For example, supervised learning algorithms use NFRs' labeled datasets, which have been manually classified by type, to learn certain parameters (features, patterns, or functions) of each type of NFR. In other words, these algorithms find the relationships between NFR statements (input) and their labels (output) for building a model that uses these relations for further NFR classification tasks. Unsupervised learning, in contrast, uses input requirement documents to drive structure (i.e., groups of requirements) by looking at the relationship between the inputs (i.e., textual requirements) themselves. Semi-supervised learning algorithms apply supervised learning iteratively with both labeled and unlabelled data. Labeled

Learning task	**No. studies**	**Studies ID**
Binary-class	26	S2, S5 , S6, S8, S10, S16, S18, S20, S21,S22, S23, S26, S27, S29, S31, S34, S39, S41,S42, S43, S45, S46, S47, S48, S49, S50
Multi-class	24	S1, S3, S4, S9, S12, S13, S14, S17, S19, S21, S22, S23, S24, S28, S29, S33, S35, S37, S38, S40,S42, S47, S50, S51
Multi-labels	9	S7, S8, S10, S11, S15, S30, S32, S36, S44

Table 3.15: The distribution of the studies by learning Tasks

data are used to train the model to classify further unlabelled requirements.

Learning Tasks. NFRs classifiers applied three different learning tasks: binary, Multi-class, and Multi-labels classification task. Binary classification refers to tasks in which an input data point into one of two classes is classified. Multiclass tasks have more than two class labels, and each data point is classified into one out of K classes (where $K > 2$). In the multilabel classification task, each data point is classified into one or more class labels. For example, a single requirement could be Functional and Security at the same time. Table 3.15 shows the distribution of the studies by learning tasks. As the table shows, binary learning is most widely applied, followed by Multi-class, and finally Multi-labels learning task.

Learning Tools. Seven tools have been used to build NFRs classifiers by the selected studies. These tools are briefly described as the following:

- Weka [7] is a publicly available ML workbench, which provides easy access to state-of-the-art ML algorithms for data mining [FHH+09]. It has graphical user interfaces, which facilitate the whole data mining process, including preparing the input data, building and evaluating the learning models, and visualizing the inputs and results [FHH+09]. Weka's main drawback is its lack of flexibility for customizing feature selection methods. Thus, some studies, such as S2, have built their own feature selection programs and use Weka for training and evaluating their ML model. On the other hand, one of Weka's main strengths is its user friendliness and its large number of implemented text classification algorithms [JBB14].

[7] `https://www.cs.waikato.ac.nz/ml/weka/`, Last accessed March 2021

- Scikit-learn [8] is a free Python package, which must be used by skilled Python programmers due to its command-line interface. Scikit-learn provides variously supervised and unsupervised learning algorithms and supports several ML process steps, including dimensionality reduction, feature extraction, feature selection, parameter tuning, and evaluation. It is more flexible than Weka; however, it required python programming language skills to run a command-line interface, as mentioned earlier.

- GATE (General Architecture for Text Engineering) [9] is a Jave-based tool developed at the University of Sheffield. GATE provides extensive multilingual extraction techniques [AC06]. However, according to Adeva and Calvo [AC06], it does not offer direct access to text classification functions. Nevertheless, some plug-ins have been developed and used to perform NFR classification, such as Machine_Learning used by S8.

- SVMlight framework [Joa99] is developed by researchers at Cornell University. SVMlight is an open-source implementation of SVM in C, with a fast optimization algorithm. However, it requires C programming language skills to run SVM through a command-line interface.

- LIBSVM [CL11] is an open-source library for SVM to perform both classification and regression tasks. It has been widely used by researchers to fulfill different tasks and integrate into WEKA as a default SVM module [AW15].

- Gensim [10] is an open-source Python-based library for unsupervised document indexing and similarity techniques with large corpora. Three studies used this library (i.e., S26, S37 andS51) for generating word2vec embedding, which is then used to predict NFR class with semantic similarity method (S37) or to be fed to NN classifiers (S26 and S51).

- TensorFlow [11] is an open-source Python-based library used for developing ML, with a focus on deep learning (i.e., NN models). It was developed by Google to be applied to different kinds of data, including text . Three studies used this library for building NN classifiers (i.e., S26, S38, and S51).

[8]`https://scikit-learn.org/stable/`, Last accessed March 2021
[9]`https://gate.ac.uk`, Last accessed March 2021
[10]`https://pypi.org/project/gensim/`, Last accessed March 2020
[11]`https://www.tensorflow.org/`, Last accessed March 2012

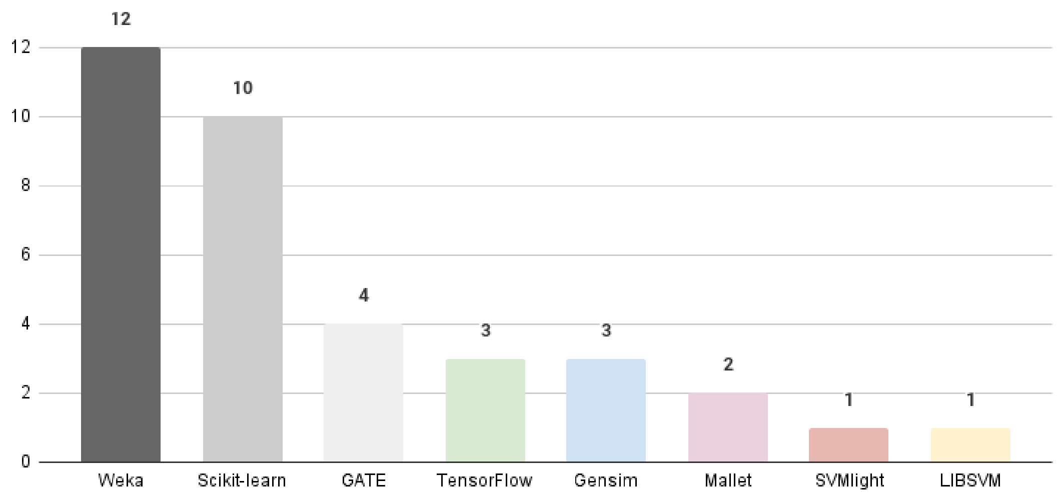


Figure 3.10: The distribution of studies by ML tools

- Mallet [12] is a Java-based package for NLP, document classification, clustering, information extraction, and other ML applications to text. Mallet is a Multi-Language support package that running via a command-line interface.

Figure 3.10 shows the distribution of the studies by tools. As the figure shows, Weka has been used more often than other NFR classification tools. The user-friendly interface of Weka and the diversity of ML algorithms could be a reason behind the widespread use of Weka, not only in NFRS classification but in different data mining tasks [JBB14]. Scikit-learn is the second most used tool, followed by GATE, Gensim, TensorFlow, Mallet, SVMlight, and LIBSVM.

Evaluation

In this step, an NFR classifier's evaluation is conducted using various methods to determine the effectiveness of trained ML classifiers. Generally speaking, ML classifiers' effectiveness is measured by comparing the human-based labels against those predicted by ML models. This, as mentioned in Chapter 2, is generally done through three steps: score estimation methods, evaluation measures, and statistical validation methods. Evaluation measures and results will be discussed in Section 3.4.4. In this section, we review the score estimation and statistical validation methods. Also, we highlight the differences between evaluating supervised and unsupervised methods.

[12]`http://mallet.cs.umass.edu/topics.php`, Last accessed March 2020

Scores Estimation Methods. Scores estimation methods determine how to deal with available data to estimate the performance scores (evaluation measures). The studies used six different scores estimation techniques; the distribution of the studies by the techniques is shown in Figure 3.11. This figure indicates that K-fold cross-validation and holdout (illustrated in Chapter 2) are the most frequently used techniques. The remaining four techniques are illustrated as follows.

- Project-level cross-validation (Pflod). It is similar to the K-fold cross-validation; however, it divides data according to the projects instead of instances (data size). Thus, data is separated into K folds, where each fold has a similar number of projects. The learning algorithm is run K times, and the average of K results is calculated to produce a single result.
- A new testing domain. It is like the holdout technique in that there are two separate datasets: one for training and the other for testing. However, the testing dataset comes from different domains than the training data.
- Out-of-sample bootstrap. It randomly selects samples with replacement from a dataset to train a ML model. The samples that are not included in the training are used to evaluate the model. This bootstrap process is repeated M times, and the average performance is calculated [TMHM16].
- Train fitness: This is the reuse of a training dataset to evaluate a trained classifier for measuring how well it fits its training data.

Statistically Validation Methods. These methods assess whether there are statistically significant differences between the results of different learners. Only five studies (9.8%) have adapted these methods and each study uses a different statistical test. These tests are :

1. Student's T-test used by S7. It is a test for continuous data that investigate if the values of two groups are the same. It assumes that data are normally distributed with equal variance [DPRHB10].
2. Welch's t-test used by S22. It is similar to Student's T-test; however, it does not assume equal variance (i.e., homoscedasticity) or sample sizes [Sak16].
3. Analysis of Variance (ANOVA) used by S21. It is similar to Student T-test; however, it is used to compare the means among three or more groups.

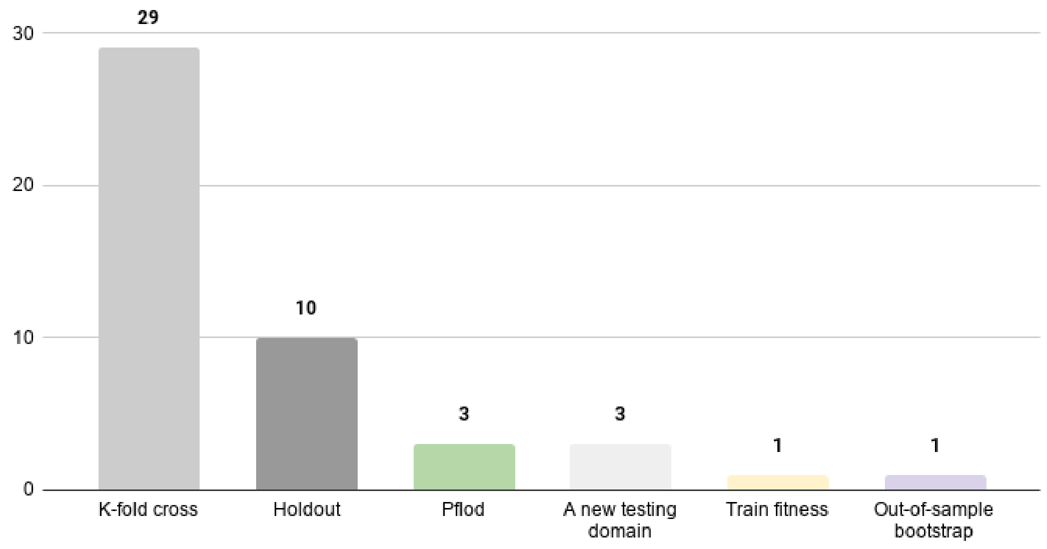


Figure 3.11: The distribution of studies by the score estimation methods

Statistically validation methods	**No. studies**	**Studies ID**
Student's T-test	1	S7
ANOVA	1	S21
Welch's t-test	1	S22
Scott-Knott test	1	S31
Wilcoxon statistical	1	S51

Table 3.16: The distribution of studies by statistical validation methods.

4. Scott-Knott test by S31. It is a hierarchical clustering algorithm designed for ANOVA to partition samples into distinct homogeneous groups. Similar to ANOVA, it assumes normality and homoscedasticity [SK74].

5. Wilcoxon statistical Test by S51 for ordinal or continuous data. In contrast to the Student's T-test, the Wilcoxon test does not assume the normality distribution of data.

Clusters Quality in Unsupervised Learning. Besides measuring the correctness of predicted labels against the true ones, some of the studies that applied unsupervised learning also measured the quality of clusters before predicting their classes (S15 and S23). The cluster's quality includes cohesion (members in each cluster are close to each other) and separation (members of a cluster are far away from other clusters) methods. S15 measures cluster cohesion and separation using semantic similarity between words in each cluster (i.e., pairwise semantic similarity). S23 computed silhouette coefficient for measuring cluster cohesion and separation. These techniques are related to the internal quality of each cluster, not their final output (i.e., the predicated categories).

3.4.4 Performance of the Reported ML Algorithms (RQ3)

Q3: What measures have been used to evaluate the performance of the ML algorithms applied in these methods? What are the performance results of these algorithms?

Of the 51 selected studies, 47 studies (92%) have reported classification results. In total, 13 different measures have been used to assess classifier performance. Table 3.17 shows the distributions of the studies by measures. The table shows that precision, recall, and f1-score, illustrated in Chapter 2, are the most frequent measures. Besides these measures, there are 10 measures which are briefly described below.

- *Accuracy.* This is the measure of the number of correct classifications divided by the total number of classifications. It is defined as accuracy = (true positives + true negatives)/(true positives + true negatives + false positives + false negatives). This measure was applied by 11 studies; however, none provided a reason for computing the accuracy of their work.

- *Confusion matrix.* This is used to analyze the classification results showing the number of correct classifications (true positive, true negative) and incorrect classifications (false positive, false negative). Only six studies provide the confusion matrix for their classifiers (S1, S2, S8, S18, S25, and S4). S1 and S8 mentioned that the reasons for computing this matrix were to provide useful classification results and identify room for more improvement.

- *ROC values.* Sometimes known as the "area under the curve" or AUC, this method visualizes the performance of classifiers as graphs; the x-axis represents ‘1-specificity’ (false positive rate), while the y-axis represents sensitivity (true positive rate). ROC represents the accuracy of the performance in relation to new instances. Only five studies adopted this method to compute their classifiers’ performance (S7, S16, S34, S46, and S49). The authors of S7 argued that ROC suffers from less bias than other measures, which skews toward the positive class, especially in the case of an imbalanced class.

- $F2-Score$. It is a weighted harmonic mean of recall and precision, with β = 2 in equation 2.4. The studies that applied this measure emphasized recall over precision. The main assumption is that false positives tend to be easier than false negatives. In other words, searching for a false positive in the output is easier than manually browsing the entire document looking for missed NFRs.

- *Loss*. It measures the absolute difference between the prediction and actual values. Practically, the lower value of the loss, the better an algorithm performs. Four studies used this measure: S1 (Mean Absolute Error), S2 (Cross-entropy Loss), S3 (test loss), and S44 (Hamming Loss).

- Kappa metric. This metric is often used to assess the agreement between raters (see Section 3.4.3). It is also used to assess a classifier's performance by measuring the agreement between predicted labels and truth labels. Two studies (S2 and S23) used Kappa to assess classifier performance, however, none of these studies specified which Kappa measure was used.

- *Transductive* and *inductive accuracy*. Transductive accuracy measures the accuracy of predicting unlabelled instances in training, while inductive accuracy measures the accuracy of predicting unseen test data (testing data) [DHR17]. Both of these values were used to measure the accuracy of the semi-supervised learning algorithms in one study S20 as the authors of this study argued that there were two types of learning in semi-supervised learning: one with testing data and the other with unlabelled data.

- *Subset Accuracy (SA)* and *Hamming Score (HS)* are both used by S44 to evaluate a multi-labels classifier's performance due to their commonly used for such learning tasks. Subset Accuracy, known as Exact Math, is rigorous metric computing the number of completely correct predictions divided by the total number of classified instances. Hamming Score is the number of correctly predicted labels divided by the total number of identified labels (predicted and actual) for a data instance.

Table 3.18 shows studies' evaluation measures and the corresponding results (the confusion matrix is not included in that table as it is hard to be reported in one cell). The results reported in the table are the highest results achieved by each study in the case of using multiple datasetes or classifiers. Therefore, the table shows the corresponding classification setting for each study. As some studies did not provide the

Evaluation measures	**No. studies**	**Studies ID**
Precision & Recall	38	S1, S2, S4, S6, S8, S9, S10, S11, S13, S15, S18, S19, S21, S22, S23, S24, S25, S26, S27, S28, S29, S31, S33, S34, S35, S36, S37, S38, S39, S40, S41, S42, S44, S45, S47, S48, S50, S51
$F1-Score$	30	S4, S5, S6, S7, S8, S9, S11,S12, S18, S19, S21, S22, S25, S26, S27, S28, S29, S31, S33, S34, S35, S37, S38, S39,S40, S42, S47, S48, S50, S51
Accuracy	11	S4, S21, S26, S30, S34, S36, S39, S40, S41, S46, S49, S51
ROC values	5	S7, S16, S34, S46,S49,
$F2-Score$	4	S15, S26, S44, S45
Confusion matrix	6	S1, S2, S8, S18, S25, S41
Loss	4	S2, S26, S46, S44
Kappa	2	S2, S23
Transductive accuracy	1	S20
Inductive accuracy	1	S20
Subset Accuracy (SA)	1	S44
Hamming Score (HS)	1	S44

Table 3.17: The distributions of the studies by evaluation measures

overall performance (average), we report the results as a range by showing minimum and maximum results (min, max).

Among supervised classifiers shown in Table 3.18, SVM is reported more than other algorithms. However, the performance of SVM classifiers, and other classifiers, vary from studies to another. For example, the SVM classifier in S8 performed higher than the SVM classifier applied in S9. Therefore, it is hard to determine which algorithms are more promising in NFRs. The classification type, number of classes, dataset size, quality of requirements, algorithm configuration setting, evaluation methods all affect the final results. For example, DT outperforms NB in S48, while NB outperforms DT in S48. Besides, the comparison of 13 supervised classifiers in S28 shows that MNB is better than SVM, while S10 found that SVM is better than MNB.

ID	**#Class**	**ML**	**Type**	*P*	*R*	*F*	**Acc.**	**TA**	**IA**	**AUC**	**Sen.**	**Kappa**	**Loss**	**SA**	**HS**
S1	10	Own	SL	0.14	0.76	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S2	2	DT	SL	1.0	0.98	0.99	0.98	N/A	N/A	N/A	N/A	0.97	0.09	N/A	N/A
S3	NOT Provided														
S4	10	NB +EM	SSL	0.80-1.0	(0.35-1.0)	(0.48,1.0)	(0.80, 0.97)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S5	3	SVM SL	N/A	N/A	(0.40, 0.65)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
S6	2	NB	SL	0.79	0.91	0.84	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S7	6	Bayesian learners	SL	N/A	N/A	N/A	N/A	N/A	N/A	(0.55, 0.89)	N/A	N/A	N/A	N/A	N/A
S8	7	SVM	SL	0.84	0.84	0.84	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S9	14	SVM	SL	0.54	0.73	0.62	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S10	unknown	SVM	SL	(0.80, 0.83)	(0.64, 0.66)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S11	6	Combined (KNN, NB,SVM)	SL	0.8	0.76	0.78	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S12	unknown	SVM	SL	N/A	N/A	(0.35, 0.55)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S13	11	not clear	SL	(0.23, 0.70)	(075, 0.95)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S14	NOT Provided														
S15	12	HAC+TSS + NGD_Wiki	USL	(0.50, 0.52)	(0.74, 0.88)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S16	4	DT	SL	N/A	N/A	N/A	(0.69, 0.83)	N/A	N/A	(0.69, 0.83)	(0.65, 0.80)	N/A	N/A	N/A	N/A
S17	NOT Provided														
S18	8	Rule-based	SL	0.98	0.96	0.97	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S19	6	Bagging	SL	0.71	0.72	0.72	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S20	2	Self-Training(SVM)	SSL	N/A	N/A	N/A	N/A	0.75	0.76	N/A	N/A	N/A	N/A	N/A	N/A
S21	7	SVM-B + active Learning	SSL	(0.58, 0.94)	(0.20, 0.83)	(0.11, 0.82)	(0.67, 0.71)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

ID	#Class	ML	Type	*P*	*R*	*F*	Acc.	TA	IA	AUC	Sen.	Kappa	Loss	SA	HS
S22	2	SVM	SL	0.87	0.87	0.87	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S23	2	NB	SL	0.95	0.94	0.94	N/A	N/A	N/A	N/A	N/A	0.89	N/A	N/A	N/A
S24	6	LDA	USL	0.68	0.76	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S25	12	NN	SL	0.80	0.78	0.77	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S26	2	CNN	SL	0.93	0.92	0.92	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S27	2	One-class SVM	USL	(0.61, 0.73)	(0.64, 0.79)	(0.61, 0.74)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S28	10	MNB	SL	0.84	0.68	0.72	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S29	3	SGD	SL	(0.57, 0.89)	(0.62, 0.72)	(0.62, 0.76)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S30	12	SVM	SL	N/A	N/A	N/A	0.91	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S31	2	SVM, LR or NB	SL	(0.44, 0.93)	(0.44, 0.70)	(0.52, 0.76)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S32	2	LDA	USL	N/A	N/A	N/A	0.90	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S33	6	NB	SL	N/A	N/A	(0.60, 0.63)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S34	4	SVM-CV	SL	(0.59, 0.88)	(0.17, 0.93)	(0.59, 0.81)	N/A	N/A	N/A	(0.57, 0.86)	N/A	N/A	N/A	N/A	N/A
S35	3	SVM	SL	(0.78, 0.98)	(0.62, 0.85)	(0.73, 0.91)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S36	7	KNN	SL	(0.45, 0.68)	(0.28, 0.56)	N/A	(0.21, 0.43)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S37	12	Word2vec	USL	0.75	0.56	0.64	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S38	5	CNN	SL	(0.82, 0.94)	(0.76, 0.97)	(0.82, 0.92)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S39	not clear	SGDSVM	SL	0.66	0.61	0.61	0.76	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S40	10	RNN (LSTM)	SL	0.97	0.97	0.96	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S41	2	NB	SL	0.65	0.71	N/A	0.97	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S42	7	CNN	SL	0.74	0.41	0.53	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/Av N/A	
S43	2	LDA	USL	N/A	N/A	N/A	0.09*	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S44	4	BR-SVM	SL	0.64	0.54	0.56	N/A	N/A	N/A	N/A	N/A	N/A	0.18	0.40	0.49

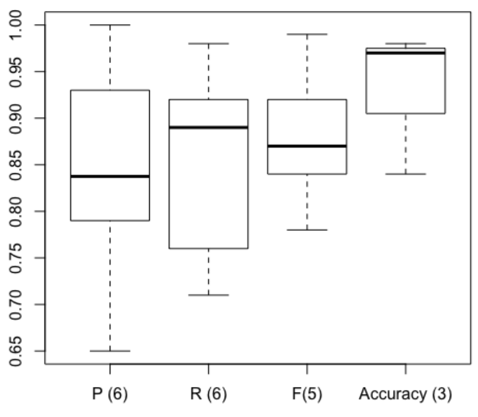
ID	#Class	ML	Type	*P*	*R*	*F*	Acc.	TA	IA	AUC	Sen.	Kappa	Loss	SA	HS
S45	2	LR	SL	(0.88, 0.95)	(0.57, 0.67)	(0.81, 0.88)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S46	2	CNN	SL	N/A	N/A	N/A	(0.65, 0.71)	N/A	N/A	(0.50, 0.75)	N/A	N/A	(0.8, 5.2)	N/A	N/A
S47	10	BERT	SL	N/A	N/A	0.82	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S48	2	J48	SL	0.80	0.76	0.78	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S49	2	RNN	SL	N/A	N/A	N/A	0.84	N/A	N/A	(0.33, 0.87)	N/A	N/A	N/A	N/A	N/A
S50	12	LR	SL	0.78	0.79	0.78	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S51	7	CNN + LR	SL	0.95	0.95	0.94	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

*The accuracy of S34 is computed manually using the number of correctly classified requirements / a total number of requirements, provided by the authors.

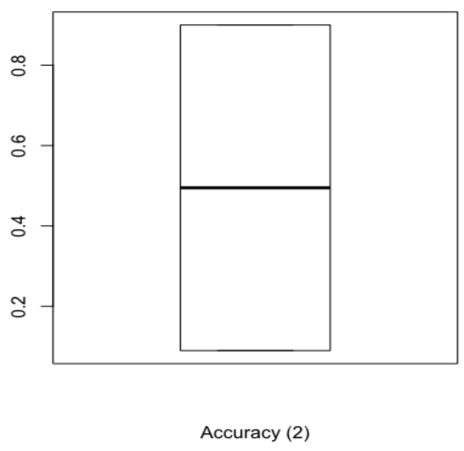
F in S44 and S45 is F2

Table 3.18: Performance measures and results in 51 selected studies

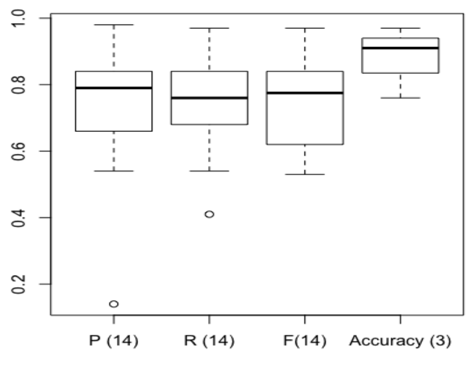
Figure 3.12 shows an overview performance of the studies, that reported the overall performance (nor range) of an ML classifier. These studies are grouped based on learning types (supervised or unsupervised) and learning tasks (binary or multi-class learning). The figure indicates that supervised classifiers generally show higher performance than unsupervised. This is similar to the finding of S23, which compares supervised learning performance with multiple unsupervised learning. Besides, the figure shows that binary supervised classification is better than multi-classes classification, more likely due to the simplicity of binary tasks.



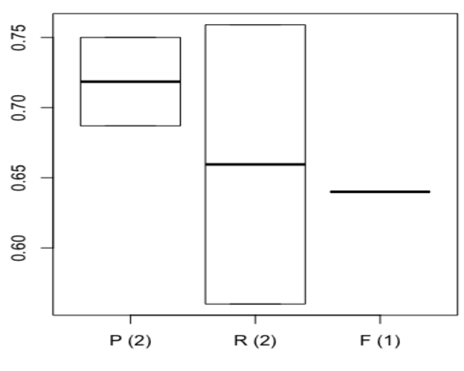
(a) Supervised-Binary



(b) UnSupervised-Binary



(c) Supervised-Multi-class



(d) UnSupervised-Multi-class

Figure 3.12: The performances of existing NFR classifiers according to learning type and task. The number between braces "( )" represents the number of studies involved.

3.5 Key Findings, Limitations, and Open Challenges

In this section, we derive the key research findings from our review results and discuss the limitations of the surveyed approaches and open challenges.

3.5.1 Key Findings

The key findings that we have identified from our review are:

- Classifying NFRs using ML has got much attention in the last 3 years, where about 22 out of the 51 studies were published between 2018-2020.
- ML-based methods have generally performed well, achieving an accuracy of more than 70% in detecting and classifying NFRs.
- Supervised learning applied most frequently in NFR classification, and SVM being the most frequently used algorithm.
- NN models are used less frequently due to the lack of availability of large labelled datasets. However, the use of pre-trained models (e.g., word embedding) or transfer learning (e.g., BERT) helped overcome the small size of datasets.
- Supervised learning methods performed better than unsupervised learning methods. Additionally, binary supervised classifiers showed higher performance than multi-class classifiers.
- A pre-defined data (e.g., keywords) is required for building an effective NFR classifier using unsupervised learning.
- Validating the manual annotation of datasets is frequently applied by the studies, and a discussion of the disagreements is the common method to address the conflict between annotators.
- Binary learning-based classification is commonly applied, while multi-label classification is less frequently used in building NFRs classifiers.
- NFRs classification into further classes (e.g., security and usability) is the most frequently addressed task, followed by NFRs identification (i.e., distinguish NFRs from other requirements).

- Security requirements are mostly discussed and analyzed requirement type among 46 classes identified in the studies.
- Preprocessing and features representation steps are applied more often than feature selection or feature extraction steps.
- Stop-word removal is the widely used pre-processing technique, and TF-IDF is the widely used feature representation technique.
- Most of the feature selection techniques are Frequently-based techniques.
- Statistically significant validation methods are used less often in NFRS classification.
- Weka used most frequently in building NFRs classifiers.
- The cross-validation technique is widely used to assess NFRs classifiers.
- Precision, recall, and F-score are the most common measures used in evaluating NFRs classifier, with a slight focus on the importance of recall over precision.

These findings show that, despite being still in the early stage of research, ML-based approaches have already produced promising results and key investigations in identifying and classifying NFRs. This is a positive prospect.

3.5.2 Limitations

During our review, we have observed a number of limitations specific to reporting and evaluating ML approaches. These limitations are described as follows.

Lack of Reporting Standards

Both supervised and unsupervised algorithms require pre-defined data: supervised learning requires labeled data, while unsupervised learning needs pre-defined categories and associated terms to achieve high performance. Before applying ML (supervised or unsupervised) algorithms, ML approaches must take some steps to process input requirements and identify relevant features for the ML algorithms. After that, these methods are evaluated and results are reported. However, this process has not been clearly described in the reviewed approaches. Many of the studies treat ML methods as "black boxes" and provide no details on how these methods work or are evaluated. This makes our review quite difficult. Additionally, the replicability of the methods will not be possible.

To ensure review consistency, we used a general process (Figure 3.6) as a common structure to assess individual studies. The main topics of each step shown in the figure are extracted by carefully examining the studies. Since we use the CCM method, we defined these topics iteratively by analyzing the studies' similarities and differences. The contents (topics or techniques) that frequently appeared are reviewed and reported.

Lack of Evaluation Standards

Although the majority of the studies provided evaluation results (47 out of 51), 76% of them (36 out 47) did not explain how and why they used a certain evaluation method. For example, regarding precision and recall, these studies did not mention which was more important: high precision and low recall, or high recall and low precision. Furthermore, they did not say why they provided F-Score and not accuracy, or vice versa. In addition, some studies reported that they used a weighted average (e.g., S8, S47), micro (e.g., S50), or macro averages (e.g., S1) without specifying the reasons behind such choice.

Based on the above, it can be concluded that the majority of the studies did not know why they used a specific method that was different from the others or how to explain their results. This raises threats to the validity of the reported results and, consequently, findings. An example of such a threat is bias. For example, it is well known that accuracy measure is biased to the majority class on the imbalanced dataset [Wei04, SWK09], and holdout method is a more biased and least stable validation technique compared with cross-validation [TMHM16].

3.5.3 Open Challenges

Through our reflection on the review, we have identified three open challenges faced by researchers, elaborated as follows.

Lack of Shared Training Datasets

One open challenge of the current research is the lack of training data. Developing an accurate ML required a sufficient amount of high-quality training dataset. The dataset should be correctly pre-classified for supervised learning. Developing such a dataset is time-consuming and requires much effort. It requires collecting real-life NFRs, manually classifying these requirements, and validating this classification. Unfortunately, few shared datasets are available (shown in Table 3.7), where much of them are of relatively small size, developed by academic communities, and suffering from validity issues. For example, PROMISE was critically reviewed by Li et al. [LHM$^+$14] who found some issues of requirements in the PROMISE dataset such as vagueness and unmeasurable NFRs. Building a shared requirement corpus with a sufficient amount of high-quality requirements can encourage the researchers to conduct more experiments, improve the quality of investigations, and provide good benchmarks for future performance.

Lack of Standard Definition, Classification, and Representation of NFRs

As stated in Chapter 2, although NFRs are critical for the success of a system and system development, there is still no consensus in the RE community on what non-functional requirements are and how they should be classified and represented [Gli07]. Consequently, there is a diverse range of terms to define NFRs (e.g., property, characteristic, attribute, quality, constraint, and performance), leading to not only terminological but also major conceptual discrepancies [Gli07]. Therefore, the diversity of NFR definitions inevitably leads to a divergent classification of NFRs. As our review results show (Figure 3.8), each of the 51 studies uses a different set of NFR categories (46 types), and there is little agreement between them.

These definition and classification problems make ML algorithms extremely challenging, due to the difficulty of distinguishing requirement categories with a lack of agreement on category names or requirement classes. Thus, ML classifier cannot be easily generalized to data or classes that were not used for training, which makes the automated classification of requirements more complex and error-prone. Furthermore, these problems also make it difficult to compare the performance of similar methods and to set performance benchmarks.

The Overlapping Nature of NFRs

All ML methods applied by the 51 studies, except NN methods, includes selecting/extracting useful features from requirements. The selection of such features helps in avoiding optional noise and reducing overfitting. However, it is challenging, as words that are often associated with a particular NFR tend to be scattered over the other NFRs (e.g., security and performance [CHSZS07], performance, and scalability [SRS14]).

Consequently, the feature selection or extraction techniques do not always provide meaningful features.

Although the proposed classifiers showed high performance, many of the features used by these classifiers were either insufficient or not appropriate. These features were either meaningless or shared between different NFR categories. For example, in S1, 'take' and 'user' under the performance category were meaningless features and the word 'product' was identified in operational, scalability, and security categories. Also, S10 included many trivial features, such as '10', '0' and 'increase' in performance, 'word', 'let' and 'voice' in usability. On the other hand, in unsupervised approaches, the features were either too abstract or meaningless. Examples of abstract features are adapted by S15, such as 'security and 'secure' for security. Furthermore, in S7, 'human' can be considered meaningless features for the usability category.

These meaningless features can significantly increase the number of false positives and affect the performance of the classifiers. In addition, this indicates that the proposed classifiers were limited in recognizing and retrieving NFRs from the domain for which they had been trained (i.e., overfitting). Therefore, we believe that overlapping nature poses a great challenge in identifying useful features and developing an accurate ML classifier. Further investigations are needed to address this issue.

3.6 Threats to the Review Validity

In this section, the threats to the results of this review are discussed. To organize this section the threats are classified according to the review process as described below:

3.6.1 Study Identification

In this review, we used the snowballing approach rather than the traditional approach to identify the primary studies (database search). The main reason behind avoiding the traditional approach is due to a large number of false-positive results returned, which required a lot of time to select and screen. The snowballing approach allows us to carefully select a seed set and then use snowballing to identify further relevant studies around this set. This may produce a more targeted set of papers. Nevertheless, there is still the possibility of misidentifying important studies. To overcome this threat, we formulated different search strings to identify the initial set in such a way as to return highly cited papers. We used different online databases for ensuring diversity in studies identified and avoiding publisher bias.

3.6.2 Inclusion and Exclusion

Some of the primary studies did not clearly describe their objectives, contribution, and research design. This made the inclusion/exclusion process difficult and increased the possibility of excluding relevant studies. In addition, some studies used difficult and complicated ways to describe their objectives, adopted approaches, and results. This could have led to misunderstandings and an inaccurate selection of studies. In order to mitigate these threats, the decision regarding which studies to include in this review was discussed with my supervisor.

3.6.3 Data Extraction

In some of the primary studies, important information was not clear for answering the research questions. Therefore, other resources were analyzed to ensure that the correct data were extracted. These resources included related research that was mentioned in these studies or written by the same authors about similar topics or publicly available material of the studies (e.g., code or dataset).

3.7 Conclusion

This chapter provides a systematic review of 51 carefully selected ML methods used for identifying and classifying NFRs from requirements documents. This review systematically answers a number of research questions related to which ML algorithms have been used in these methods and how these methods worked and were evaluated. The overall conclusion that stems from our review results is that using ML algorithms to identify NFRs is not an easy task, as it requires appropriate methods to process and classify the text. This review also found that ML algorithms, especially supervised learning, hold great promise in NFR classification, as they can achieve a high level of performance.

This review's key findings will be continuously applied to our research. By analyzing the 51 studies, we noticed that much of the studies' contribution was about applying new techniques or using new datasets, with a lack of focus on potential problems posed by NFRs to build or evaluate an ML classifier. Thus, in the following chapter, we review the key issues commonly appearing in NFRs that related studies have rarely addressed by comprehensively analyzing related work.

Chapter 4

A comprehensive Analysis and Review of the Problems and Solutions in NFR Classification

"If I had an hour to solve a problem, I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions."

Albert Einstein

The previous chapter systematically reviewed existing ML methods proposed for identifying and classifying NFRs. Although there is a great focus on applying recent techniques and datasets, there is a lack of understanding about analyzing the problems raised by the NFR's descriptions to standard ML classifiers. As stated in chapter 1, the classification of NFR using supervised learning tends to suffer from three problems: high dimensionality, class imbalance, and short text classification.

This chapter illustrates these problems; by separately explaining what each problem is and how it hinders supervised ML classifiers' effectiveness. Furthermore, it reviews the common solutions applied to these problems in general, and in NFRs classification, in particular. The supervised NFRs classifiers identified in the previous chapter (listed in Table A.1) are included in this review. We only include the supervised methods since they are more promising in NFRs classification, and, therefore, they are applied in this thesis.

The chapter is organized as follows: Section 4.1 provides an overview of a high dimensionality problem, including 1) a description of the problem, 2) an overview of common solutions, and 3) a review of existing solutions used for NFRs classification. Similarly, Section 4.2 presents an imbalanced class problem, and Section 4.3 is for a short text classification problem. Section 4.4 discusses the limitations of existing solutions in the context of requirements classification and introduces the techniques used to address each problem in this thesis. Section 4.5 provides a summary of the chapter.

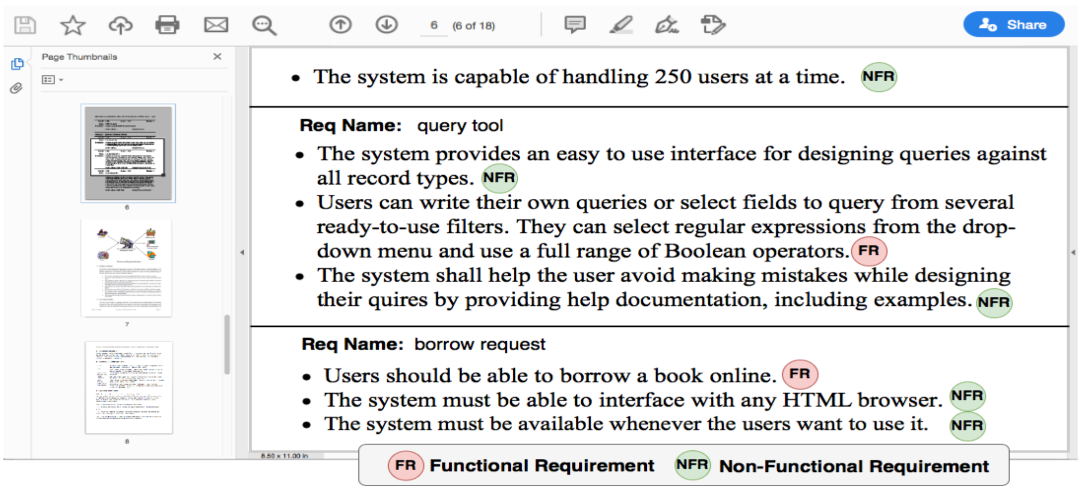
4.1 High Dimensionality

4.1.1 Learning with High Dimensional Dataset

High-dimensionality is a well-known problem that appears in different tasks, including text classification, face recognition, speech recognition, and many other tasks that deal with data described by too many attributes (input variables) and, consequently, represented in a high-dimensional vector space. Requirements are often represented as texts having many unique words that can easily number in the hundreds or even thousands. For example, Figure 4.1 shows that even a small example of a requirements document contains 74 unique words. Each unique word or phrase in the document is a potential feature treated as a single dimension in the data space. Therefore, each textual document (a requirement in our case) is represented by an extensive number of features, which are used to build classification models.

Several problems may arise in supervised classification due to high dimensionality. For example, the extensive number of words can lead to high complexity in the time and memory of the classification process [KJMH$^+$19]. Moreover, a small number of samples in high-dimensional data will be prone to overfitting (i.e., the model is overfit to the training data), leading to poor generalization and increased difficulty comprehending the model [PP14]. High-dimension low-sample dataset has been characterized by a large number of features p and a small number of samples n, (i.e., p»n) [PP14]. High-dimension low-sample dataset is prevalent in NFRs since requirements are written in NL documents, and most of the requirements datasets are small (see Table 3.7). For example, PROMISE, most frequent used dataset in NFRs classification, is a high-dimension low-sample dataset, containing of $n = 626$ requirements and $p = 2204$ unique words, where $p >> n$.

Another problem that may arise due to high dimensionality is that high-dimensional



- The system is capable of handling 250 users at a time. NFR

Req Name: query tool

- The system provides an easy to use interface for designing queries against all record types. NFR
- Users can write their own queries or select fields to query from several ready-to-use filters. They can select regular expressions from the drop-down menu and use a full range of Boolean operators. FR
- The system shall help the user avoid making mistakes while designing their quires by providing help documentation, including examples. NFR

Req Name: borrow request

- Users should be able to borrow a book online. FR
- The system must be able to interface with any HTML browser. NFR
- The system must be available whenever the users want to use it. NFR

FR **Functional Requirement** NFR **Non-Functional Requirement**

Figure 4.1: An example of requirements document

datasets often contain a high degree of irrelevant and redundant information [YL03]. For example, by analyzing the second requirement provided in Figure 4.1, "The system provides an easy-to-use interface for designing queries against all record types", we found that only a few words are necessary to represent the category type (e.g., provides easy-to-use interface). Keeping all the features increases the difficulty of identifying the relationships between variables and eventually deteriorating the accuracy of many classification algorithms [YL03].

The impact of learning from a high-dimensional space differs from one ML algorithm to another. For example, SVMs are less affected by this problem as they can discard some data points and reduce dimensionality [Joa98]. However, NBs are heavily affected, as they deal with all features equally; that is, they give the same importance to all features when making classifications, which is hard to hold in high dimensional data. Moreover, high dimensional space contains noise features, which do not contribute to class prediction, and redundant features, which are not conditionally independent, increasing the difficulty of using NB in learning from high dimensional data [CW12]. This noise also affects DTs' classification performance, as noise leads to generate noisy trees that make wrong predictions for new instances [LWDD10]. In addition, noise can decrease the generalization performance of DTs [LT16]. Similarly, high dimensional space poses difficulties for KNNs distinguishing relevant and irrelevant data points, which affects the accuracy of KNN [TM11].

4.1.2 Overview of Solutions for High Dimensionality

Dimensionality reduction methods are commonly used to address the high-dimensionality problem. These methods can include feature selection or feature extraction [IKT05]. Both methods are briefly described in the previous chapter (2 & 3); a detailed description is provided in the following subsections.

Feature Selection Methods

Feature selection methods select a (best) subset of the original features; the subset has fewer dimensions and contributes most to learning accuracy. In mathematical terms, feature selection can be defined as $|t| \ll |T|$, where t represents terms in the original dataset (i.e., T) [Seb02]. The main advantages of feature selection are its simplicity and accuracy in text classification [JGDE08]; however, important information might be lost, where some features have to be omitted [KKN14].

Feature selection methods have been broadly classified into three groups, depending on how the feature selection search is combined with the classification model: filter, wrapper, and embedded approaches.

The filter approach selects features independently of the ML algorithm by looking only at the data's intrinsic properties, such as information, distance, consistency, and correlation [Seb02]. Examples of filter methods are TF, TF-IDF, information gain, and CHI-Square. The main advantage of the filter approach is that it is computationally fast and simple. Moreover, due to the separation between feature selection methods and classifiers, this approach is performed once and can be applied in many different classifiers. However, this separation ignores the interaction with classifiers (i.e., the effect of a selected feature on ML algorithms' performance), leading to worse classification performance [SIL07].

The wrapper approach uses a learning algorithm to select a good subset of features, and the goodness is determined according to a classifier's effectiveness. In other words, this approach searches for various subsets of the whole feature set and evaluates each subset by training and testing a specific classifier. The main advantages of wrappers are that they consider the interaction between feature subset and model induction algorithm, leading to better classification accuracy than filter methods. However, the wrapper approach is more prone to overfitting than the filter approach and is more time-consuming and computationally intensive, especially with a high number of features [SIL07]. Thus, wrappers are not considered to be suitable for text classification

[CHTQ09].

The embedded approach is similar to wrapper methods; however, feature section is built into the classifier construction (i.e., guided by the learning model). Embedded methods have the advantage of both the wrapper approach—in that they interact with the classification model—and filter models—in that they are less computationally intensive than wrapper methods [SIL07].

Feature Extraction Methods

Feature extraction methods constructs (synthesizes) new features based on the original ones. In practice, feature extraction methods transform original input spaces into low-dimensional spaces that preserve most of the relevant information in the original set [KKN14, Seb02]. In mathematical terms, feature extraction was defined by Sebastiani [Seb02] as $|T'| \ll |T|$; T represents terms in the original dataset, while T' "synthetic" terms that maximize effectiveness. T' and T are not the same types (e.g., if T are words, T' may not be words at all); however, they are a combination or transformation of the original ones. The main advantage of feature extraction is reducing the dimensions (features) without losing information from the original feature space. However, the synthesized features are often not intuitively interpretable, and the original features are often lost [KKN14].

The terms "feature extraction" and "dimensionality reduction" have been used interchangeably in the literature. For example, Kowsari et al. [KJMH$^+$19] used "feature extraction" to define vector representation (i.e., converting unstructured data space into structured data space using mathematical modeling). In contrast, they used "dimensionality reduction" to describe feature extraction (i.e., synthesizing new features from original ones). Sebastiano [Seb02] divided the process of extract features for text classification tasks into two steps: 1) extracting new features from old ones and 2) converting the extracted features into new representations. Sebastiano provided two examples of feature extraction methods: terms clustering and latent semantic indexing. Terms clustering methods group semantically related terms together, where the groups (or their centroids, or their representatives) are used as dimensions of the vector space instead of original terms. Latent semantic indexing produces new concepts (dimensions) that are latent in a text. These new features are obtained by analyzing word co-occurrence patterns in a corpus to capture the hidden related meaning. Latent semantic indexing is developed to address the problems deriving from using synonymous and polysemous words as dimensions of document representations. More common feature extraction methods are provided in a recent review by Kowsari et al., [KJMH$^+$19], such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and non-negative matrix factorization (NMF).

Word embedding techniques can also be considered a feature extraction technique as they can dramatically reduce dimension size. An example of these techniques is the word2vec technique proposed by Mikolov et al. [MCCD13]. This technique takes a corpus with an N-dimensional space, applying a NN model to learn word representation and creating a semantic space with a lower K-dimensional space ($N > K$). For example, word vectors trained on a part of Google News dataset (about 100 billion words) contains only 300-dimensional vectors [1]. Word2vec is also categorized as a text representation method and unsupervised learning method to measure text similarity, as shown in Chapter 3. We will back to define and uses Word2vec in Chapter 6.

4.1.3 Techniques for Handling High Dimensionality in NFR Classification

This section reviews feature reduction techniques applied to address the high dimensionality problem in supervised NFRs classification. Out of 45 studies that applied supervised learning, 27 studies applied feature reduction techniques. These techniques can be divided into three categories, statistical-based (also known as ranked-based or ML-based) techniques, linguistic-based (also known as rule-based or NLP-based) techniques, and a combination of linguistic and statistical-based techniques. Figure 4.2 shows the distribution of studies per each category. The description of each category is provided in the following subsections.

Statistical-Based Techniques

Statistics-based methods analyze feature frequency using functions (or metrics) that score the importance of terms in the classification task and then select those with the highest score. An intuitive example of this method is term frequency, where the most common words or phrases are selected. Term frequency has been used by S3, S7, S27, S35, and S50. Another example of a statistical-based technique is TF-IDF used by S9, S20, S33, S41, S50. CHI-Square and information gain are also statistical-based techniques used by S19, S50 for CHI, and S5 and S11 and S16 for information gain.

[1] `https://code.google.com/archive/p/word2vec/`, last accessed March 2021.

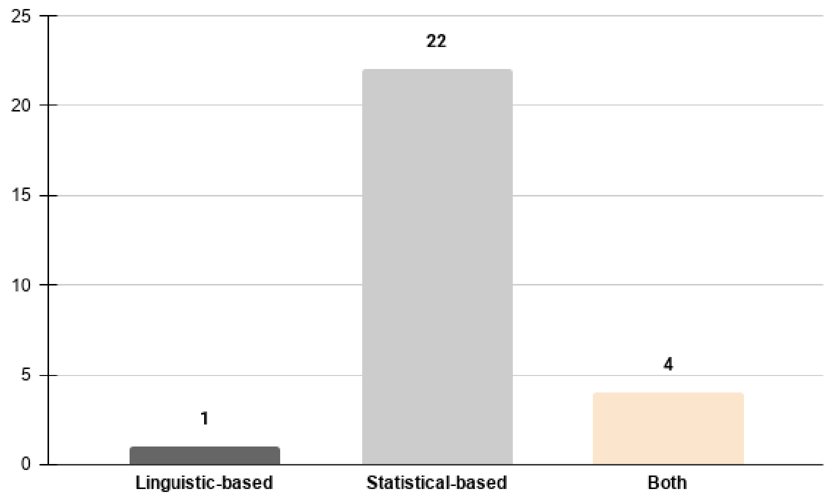


Figure 4.2: The distribution of studies per feature selection technique

All the studies above selected features from a training dataset. However, some studies selected features from another resource. For example, S29 conducted a mapping study to systematically extracted the most frequent keywords from Softgoal Interdependency Graph (SIG) catalogues for each NFRs category. In total, they found 87 words for three requirement classes (usability, security, and performance); each category has 29 words.

Besides using existing methods (such as term frequency), some RE researchers have developed their statistics-based methods to select features used in requirements classification. For example, Cleland-Huang et al.[CHSZS07] (S1) proposed a probability function that selects a set of features for each NFR class (known as indicator terms) based on their occurrence. S45 has reused this technique for detecting security requirements.

Seven studies (S25, S62, S31, S40, S46, S49, and S51) used word2ve. Apart from S31, all the studies used word2vec for NN-based classifiers. Word2vec, or NN-based classifier in general, required the availability of a large, domain-related dataset [LLHZ16, KJMH$^+$19]. Thus, the studies used pre-trained models, by Google news (e.g., S31 and S51) or Wikipedia articles (e.g., S7 and S46), to generate semantic space for representing requirements. Among all those studies, only S25 and S40 clearly mentioned that they used wor2vec techniques to capture enough features for the automatic NFR classification method.

Statistics-based techniques can be effective and flexible in different domains, languages, and writing styles. However, the main drawback of these methods is the need for a large document to perform some statistical functions effectively.

Linguistic-Based Techniques

Linguistic-based methods select features based on syntax or semantic relationships in requirements using NLP techniques. Syntactic relationships are extracted based on the structure of a dataset (i.e., grammatical relationships). Examples of these techniques are dependency parsing, POS tagging, and entity name extraction. Comparatively, semantic relationships are usually extracted using an external source, such as WordNet in S48, to identify semantically related words. A simple example of a linguistic-based technique is selecting features with particular POS tags (i.e., noun, adjective, and determiner). This technique was applied by S37; however, this study applied unsupervised learning.

Only one linguistic-based technique has been used with a supervised NFRs classifier. This technique was proposed by S48 to select features in security requirement classification. The technique is started by extracting security keywords from security taxonomies and security standards. Then, they replaced the matched words in the training dataset with the extracted keywords using WordNet. Besides, they used a syntactic structure of requirement through linguistic rules to define a set of syntactic patterns. In total, they defined 35 linguistic rules and the 140 keywords as linguistic features of security requirements. By evaluating their method, they found a higher F1-score is achieved when a training dataset and a testing dataset come from different application domains. This finding confirms the benefit of reducing the number of features in generalizing the classifier's performance.

Linguistic-based methods can extract good features from small corpora by exploiting morphological structures in a text. However, these methods are challenging as they often require manual or automatic analysis to generate patterns or set linguistics rules. In addition, most of these methods require proper use of grammar and language syntax to match such patterns.

Both Statistical and Linguistic -Based Techniques

The last category of feature reduction techniques applied in NFRs classification is a combination of statistical and linguistic techniques. Several methods used these techniques with supervised learning. For example, Hussain et al. [HKO08] (S2) extracted a POS tag for each term and computed the probability of POS tags occurrence in a training dataset. S23 used Hussain et al.'s method with an additional pre-processing step that included removing the inconsistency in a requirement description by replacing words that have the same meaning with a single term. S22 used NLP techniques (e.g., POS and n-gram) to identify features that were then ranked based on importance scores. The importance scores of features were calculated using an ensemble of tree classifiers, and the top features are selected. S34 also used probability based on linguistic features (e.g., dependency types) to extract three lists of features. Then, they applied interpretable ML techniques that provide linguistic rules to analyze predictions made by each set of features. Finally, 15 significant types of features were selected to classify requirements into Functional and Quality requirements.

This combined technique minimizes the disadvantages of each technique. For example, in a small dataset, the frequency of a specific POS tag is more likely to be higher than the frequency of a specific word. Thus, it might work better, in a small dataset, than statistical-based techniques. Besides, the combined techniques would reduce the need for manual analysis or proper use of language required for linguistic-based techniques.

4.2 Imbalanced Data

4.2.1 Skewed Distribution of Training Dataset

An imbalanced dataset occurs when one of a target class (known as majority class) contains a significantly greater number of examples than another class (minority class) [SWK09]. Learning from imbalanced datasets poses a serious problem to the ML models for two reasons: 1) the prevalence of this problem in many domains, including requirements classification [KM17]; and 2) the inadequacy of ML algorithms for dealing with this problem [SWK09]. Imbalanced datasets hinder learning algorithms' effectiveness by increasing their bias toward the large classes because these classes are well-defined, while the minority classes are not well-defined; thus, they are ignored. In most cases, the class with the lowest number of instances is the class of interest. Misclassifying these classes is more serious than misclassifying the majority classes, which have the largest number of instances. For example, in cancer diagnoses, the number of patients with cancer is less than the number of healthy people. Misdiagnosing healthy people to be sick can cause a huge amount of stress and more payment for further diagnosis; however, it is much less than misdiagnosing sick people to be healthy, which could lead to losing patients' lives.

The main characteristic of an imbalanced dataset that influences classification performance is the rarity (rare classes). However, rare class (skewed data distribution) is not the only parameter that hinders ML models' capability to predict rare classes. Rare case (known as within-class imbalances or small disjuncts) is another important factor in the difficulty of learning from smaller classes [Wei04, SWK09]. Rare cases could be a subset, sub-concept, or subclass that infrequently occurs within a single class (e.g., diagnosing rare forms of cancer). Jo and Japkowicz [JJ04] found that the skewed distribution is not the main issue; rather, small and complex data combined with rare cases are responsible for the degradation in performance of standard classifiers. Overlapping among imbalanced classes is also an important factor hindering the performance of ML classifier [PBM04, SWK09, GSM07]. It increases the difficulty of separating the minority class from the majority classes, leading to an overlapping feature space. Such space loses the intrinsic properties of minority cases, making them redundant or irrelevant to help recognize good decision boundaries between classes [SWK09].

Imbalance classes pose a problem not only during classifier building but also in performance evaluation. Classification accuracy, referring to the fraction of examples correctly classified over all the examples, is the most common metric for assessing a classification task's performance. This metric is not proper with a class imbalanced problem as it is biased toward the majority classes [Wei04, SWK09]. For example, in the case where only 1% of the training dataset represents a rare class, the classification accuracy could achieve 99% only by correctly classifying all examples from the majority class while few examples of the minority class are misclassified. Thus, accuracy is meaningless to some applications when rare cases are a major concern. In general, metrics that cannot distinguish between the numbers of corrected labels from different classes are less effective with an imbalanced dataset. Alternatively, receiver operating characteristics (ROC), G-mean, and F-score are the prominent performance metrics for models on imbalanced datasets because they have less bias toward the majority class [SWK09, Les04, Cha09, WY13].

Most learning algorithms face difficulties due to imbalanced data. For example, learning from imbalanced data increases the probability of test instances being assigned to the majority class in NBs [ZSM15], as it is hard to decode dependency patterns in small classes [SWK09]. Imbalance data also increases the probability of the majority class being the dominant class in the leaf nodes in DTs [BPM04], and the leaves predicating small classes are prone to be pruned [SWK09]. Therefore, to distinguish the small classes in DT, many splits (conditions) are required [SWK09]. KNNs are also affected by learning from imbalanced datasets, as the high majority vote (or K) will go to the sample of major classes, while the samples of the small class occur sparsely [SWK09]. SVMs suffer from a lack of small class samples at the margin; consequently, the decision boundary is inherently biased toward the majority classes to minimize the high error rates of misclassification, as such classes are more prevalent than minor classes [HG09]. However, it is believed that SVMs are less affected by imbalanced classes than other classification learning algorithms, as they only consider a few training samples to identify boundaries between classes, and class size may have little effect on class boundary [SWK09]. However, if the imbalance ratio of large classes to small classes is too high (e.g., 10,000:1), SVMs might be ineffective at determining class boundaries [SWK09].

4.2.2 Solutions for Imbalanced Data

A number of solutions have been proposed to address the imbalanced classification problem. These solutions could be categorized into two major groups: data-level (internal) and algorithm-level (external) [ASR$^+$15, SWK09, LD13].

Data Level

A data-level method is a preprocessing step that is applied to balance class distribution by either adding new instances to the minority classes (oversampling strategy) or removing existing instances from the majority classes (undersampling strategy) [ASR$^+$15]. Both sampling techniques are independent of the classifiers, easy to implement and understand, and flexible enough to be applied to any algorithm. The main weakness of undersampling strategies is low performance because of the probability of discarding useful majority-class examples that could be important for the learning process [Wei04, Ste16]. Meanwhile, oversampling strategies increase computation costs and memory usage because of the addition of new training cases. Furthermore, oversampling increases the risk of overfitting due to the new cases, which are exact copies of examples from minority classes. [Wei04]. Synthetic minority oversampling technique (SMOTE) is an advanced oversampling strategy proposed to overcome the overfitting of oversampling. SMOTE basically creates new minority class examples synthetically rather than duplicating them as random oversampling. However, there is no guarantee that new samples belong to minority classes, especially when there is an overlap between the minority and majority classes [DKC14].

Feature selection methods have also been considered as a solution to address the imbalance problem at pre-processing steps [ASR+15]. These methods [Gro99, ZWS04, CW08, WC09, YL03] aim at handling the biases in selecting features from an imbalance dataset since most traditional feature selection methods are mainly influenced by the majority class. This bias could lead to more classification errors by selecting features rather than by not selecting features [YGX+13]. An example of feature selection methods that especially address imbalanced class distribution is the decomposition-based method by Yin et al. [YGX+13]. They applied traditional feature selection methods on an imbalance dataset where the majority class is decomposed into smaller pseudo-subclasses with relatively uniform sizes and pseudo-class labels. They also demonstrated a new Hellinger distance-based method for selecting features in an imbalanced dataset. Zheng et al. [ZWS04] proposed a feature selection method that separately combines positive and negative features for each class by simply adjusting existing techniques (e.g., information gain, chi-square, correlation coefficient, and odds ratio).

Algorithm Level

Algorithm-level solutions aim to fine-tune algorithm bias to improve the learning task, especially relative to the smaller class [ASR+15]. These solutions could be categorized into 1) algorithm modification, 2) one-class learning classifications, 3) cost-sensitive learning, and 4) ensemble methods [ASR+15, SWK09].

Algorithm modification method. This category modifies classification algorithms to learn directly from imbalanced data distribution. The methods that belong to this category are developed by first learning the impact of imbalance distribution on an algorithm and an application domain (i.e., the importance of correctly classifying minority classes in a specific domain) [ASR+15]. Next, the inductive algorithm is adjusted to extract important information, which is used to build a model based on the target objective (i.e., predicting the minority class). An example of these methods is z-SVM, proposed by Imam et al. [ITK06] to adjust the hyperplane position using a new parameter z, which maintains a good margin from the data of both classes (minority and majority). Li and Zhang [LZ11] proposed KNN with exemplar generalization (i.e., kENN) to selectively enlarge the minority instances and generalize them using Gaussian balls. These main weakness of this method is the complexity since it requires extensive knowledge about the training algorithms and the application domain.

One-class learning method. These methods (known as recognition-based) create a classifier model based only on examples of the minority class (or a target class) [ASR$^+$15]. These methods establish boundaries surrounding the minority examples rather than separating positive from negative examples [ASR$^+$15]. A boundary threshold plays an important role in the effectiveness of one-class learning methods [ASR$^+$15]. In classification tasks, these methods basically measure the similarity between query objects and the minority class and make a classification based on a predefined threshold on the similarity value [SWK09]. These methods have been applied with SVM [MY01], and neural network [MY07], but they are less effective with other approaches, such as decision trees, NB, and KNN [SWK09, ASR$^+$15]. This drawback makes these methods less popular and limits their use in certain learning algorithms.

Cost-sensitive learning method. Cost-sensitive learning considers the cost of misclassification during the training of ML models. Each class (or instance) is given a misclassification cost (or weight), where the misclassification cost of the minority class is higher than that of the majority class, to change the bias of the learning algorithm to favor the minority class [LD13]. This method can be applied at the data level by assigning the cost as a sample weight [LD13]. It can also be incorporated with the process of optimizing the algorithm by, for example, using misclassification cost to choose the best feature to split the data in building a DT classifier [SWK09]. The main problem of cost-sensitive learning is the difficulty of obtaining an accurate misclassification cost [GFB$^+$11]. Moreover, cost-sensitive learning may cause overfitting [ASR$^+$15].

Ensemble method. These methods combine several classifiers to make a new classifier that outperforms each individual classifier [GFB$^+$11]. Bootstrap aggregation (bagging) and boosting are two main examples of the ensemble method (both are illustrated in Section 3.4.2 and Figure 3.5) [ASR$^+$15, SWK09]. Although bagging was not developed to handle the skewed class distribution, it performs well with an imbalanced dataset [GFB$^+$11]. The same with boosting, where each sub-classifier is built according to the previous classifier's output, and, in each iteration, the weight of incorrectly classified samples is increased. The incorrectly classified samples are commonly the minority classes [Wei04]. The main weakness of boosting is the difficulty of understanding the classifier error diversity, especially with the use of more classifiers [ASR$^+$15]. In addition, standard ensembles, such as AdaBoost (Adaptive

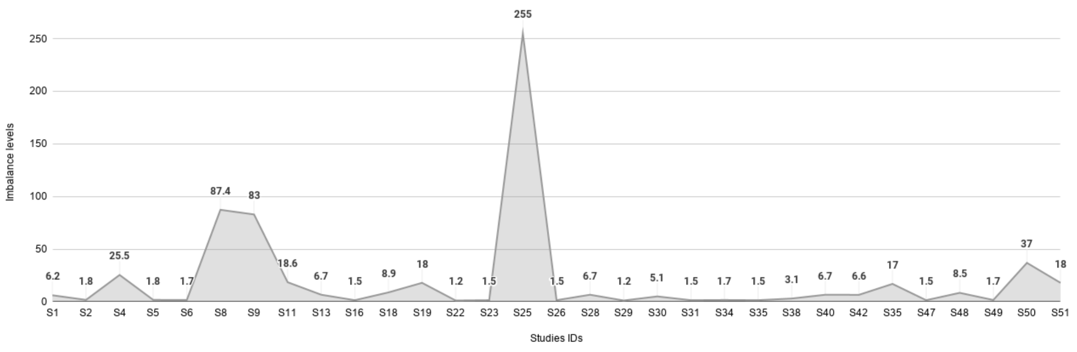


Figure 4.3: Imbalance levels of the studies identified in the previous chapter

Boosting), boosting, and bagging, are accuracy oriented, when the class distribution is uneven, these techniques bias the learning (the weights) toward the majority class since it contributes more to the overall accuracy [GFB$^+$11]

4.2.3 NFR Classification with Class Imbalance Problem

The imbalance problem is prevalent in requirements classification, as requirements categories are naturally imbalanced, with an uneven distribution of the requirements among different categories [CHSZS07, EVF16]. Figure 4.3 shows the imbalance levels of the studies identified in the previous chapter; only the studies that showed clearly the distribution of a training dataset are included. Imbalance level is the relation of the majority class size to the minority class (i.e., the size of the majority class divided by that of the minority class) [LWZ08]. More details are illustrated in Table B.2 in appendix B . As the figure shows, the smallest imbalance level was 1.2 (67: 54) by S22 and S29, and the highest imbalance level was 255 (255:1) by S25.

Among these studies, only five studies (S22, S30, S40, S47, and S51) clearly address the imbalanced class problem. All these studies applied re-sampling techniques (i.e., data-level techniques) to handle the imbalance problem. For example, S22 used over-and under-sampling techniques with an SVM classifier to classify requirements into usability and performance. This study also evaluated the usefulness of using user reviews as a resource for oversampling technique, where the additional samples are extracted from Amazon software reviews. S47 applied under- and oversampling strategies with BERT to classify requirements into FN and NFRs. S51 used undersampling and oversampling (i.e., SMOTE) on an imbalanced dataset of 6 classes, with an imbalance level =1119: 62. S30 also used SMOTE to handle the class imbalance with

SVM, NB, and KNN. S40 mentioned that they applied a re-sampling strategy without specifying which strategy they used.

As mentioned in the previous section, one-class learning methods and ensemble methods are among the common solutions to imbalance problems. Although these methods have been used in NFRs classification, the purpose of their use was not for handling the problem. For example, S27 used one-class SVM to classify requirements into security and non-security requirements. However, this method aims to evaluate the effectiveness of using a domain-independent dataset in building a security classifier, not handling the class imbalance problem. S11, S19, S28, S33, and S48 used ensemble learning for different classification tasks: identifying security requirements (S48), classifying security requirements (S11), or classify requirements into multiple NFRs (S19, S28 and S33). However, the use of this learning method in all the studies was to compare such method with other ML methods empirically; and select the method with high performance.

Besides the studies that used well-known solutions for different purposes, some studies handle this problem differently (not well-known solutions). For example, S5 selected only three classes from a detest (i.e., PROMISE) to build binary classifiers to reduce the imbalance between classes. S45 argued the purpose of their use of the 10-fold strategy is to have a balanced training set. Although such attempts are not well-known solutions, they show an awareness of the problem among the researchers.

4.3 Short Text Classification

4.3.1 Short Text Characteristics and Impacts on Supervised Learning

Short text classification has recently gained great interest coincided with an increase in short texts available in many fields such as mobile short messages, news titles, product reviews, tweets, and Q&A scripts [CJS11]. The shortness is the principal difference between short text than normal (traditional) text. Short texts consist of very few words, from a dozen words to a few sentences [SYD$^+$14]. Conversely, normal (traditional) texts have more text, including entire web pages, email contents, or news articles. Thus, normal texts provide clear enough context for automatic analysis, whereas short text does not. Due to a few words in short texts, there are no enough information or sufficient word co-occurrences to allow meaningful statistics to be gathered or a strong linkage to certain topics to be established [LHW+17, SYD+14]. Moreover, due to the nature of the diversification of language, the same concept can be expressed in different ways, therefore, the possibility of certain features frequently appearing in different short text is reduced [WHYL12]. This increases the difficulty of extracting invalid features and fails to improve classification accuracy [WHYL12, SYD+14, ZW15].

As short texts have not enough contextual information, they suffer from sparseness (i.e., a given short text span over a wide range of words). The sparsity in feature space means, that the feature matrix is very sparse, each document contains a small percentage of a total number of terms that appeared at least once in a full set of documents [SYD+14]. Technically speaking, sparseness means that there is a lot of zeros elements in a vector space [SFHA14]. The sparseness of features increases the difficulty in managing and analyzing the features based on simple text representation (such as BOW) which regards each word as a separated individual. This affects the performance of standard ML classifiers since most of them perform classification by establishing a correlation between features and categories [SFHA14, AG19].

Sparse data has various impacts on ML algorithms, with some performing better than others, such as SVM and NB. SVMs can reduce the dimensions of training datasets by discarding instances that are not close to the boundary; this reduction might reduce data sparseness [Joa98]. NBs assume the independency of features; thus, they simply ignore the zero elements when making probability estimate calculations [RRSK10]. However, DTs have difficulty learning from sparse data, as such data always leads to imbalanced and extremely deep trees [SZK+17]. KNNs also find it difficult to measure the similarity between two data points in a sparse dataset, which leads to unreliable neighborhoods due to the lack of overlapping values [GMFG05].

4.3.2 Solutions for Short-Text Classification

As the main problem of short texts is feature sparsity, most existing solutions address this problem through feature extension [SHA12]. As the name suggests, feature extension solves the feature sparsity problem by augmenting the feature space model with additional features extracted from internal or external language resources. This may involve adding more features to the feature space or replacing features in the feature space with a richer set of new features [SHA12]. The techniques for identifying additional features vary; in the following, we review some techniques grouped by the resources of additional features (i.e., external and internal sources).

Internal resources include a training dataset (e.g., the dataset used to build the model), where additional features are extracted from a training dataset based on a semantically or statistical analysis of this dataset. Different methods have been used to extract related features from an internal dataset. For example, Man (2014) [Man14], Fan et al. (2010) [FH10], and Zhang et al. (2015) [ZW15] expanded short text with the most related terms. The related terms were extracted based on an analysis of word occurrence in a training dataset. Krzywicki et al. (2018) [KHB+18] expanded rare words in a short text with semantically similar words in a training dataset using word embedding (i.e., word2vec). [ZLS+18]

In contrast, external resources are outside knowledge resources (not training dataset). The methods that belong to these resources can be divided into four categories. The first category is web search-based methods (e.g., web search results are added to short texts). An example of this category is the web search results used by Sahami and Heilman in 2006 [SH06] to provide greater context for short text. They treated each short text as a query in a web search engine and then used the returned documents to expand the short text. The second category is a short text's context. For example, Jiang et al. (2010) [JYZ+11] used related tweets in tweet classification. The related tweets could be replied tweets, retweeted tweets, or tweets by the same person published in a short timeframe. The third example of external resources methods are taxonomy-based methods. For example, Hu et al. in 2009 [HSZC09] used Wikipedia and WordNet for expanding short texts. They extracted seed features from short texts to generate new features based on the semantic similarity of the data in the external resources (Wikipedia and WordNet). The new features were then integrated with the original feature to build the feature space. The forth category are corpus-based methods (analyzing external datasets to discover relevant features). For example, Phan et al. [PNH08] built large-scale external data (called "universal dataset") in their method, and used topic-model (LDA) to inferred hidden topics from the external dataset. The extracted topics were integrated with the original documents. Chen et al. [CJS11] used the same external dataset to generate multi-granularity topics that are combined with word features to address the sparseness in short-text classification.

Among these methods, internal resources methods are easy to implement, less time-consuming and less prone to noise [LHW+17, WWZY17]. However, they may not have enough features, especially when training datasets are small [LHW+17]. Conversely, external language resources offer large text corpora and can handle feature sparseness better than internal resources [SH06, HSZC09, CJS11, PNH08]. Nevertheless, external resources are generic, and thus contain more noises and irrelevant features than internal resources. Per the experiments of Dave et al. [DLP03] and Bollegala et al. [BAMK18], using an external resource (i.e., WordNet) produces more noise, which leads to an unmanageable number of features and low classification accuracy. Thus, most of the methods that use external resources either go through a long vetting process to select the related features [SH06, HSZC09] or require large, rich, and relevant datasets that are consistent with the training dataset (e.g., [CJS11, PNH08]).

To handle such threats, a combination of different resources or methods has been used. For example, Wang et al. [WWZY17], applied a hybrid of feature extension methods; taxonomy-based and corpus-based methods. Their method first carries out feature selection and feature extension and then combines the selected and expanded features with pre-trained word vectors, to capture additional features through multiple degrees of similarity between pairs of words [MYZ13]. Zeng et al. (2018) [ZLS$^+$18] extracted latent topics from an internal resource. The extracted topics were then integrated with short-text embedding to generate the feature space. Although such an approach has been achieved significant improvement over state-of-the-art methods for short text classification, using word embedding models for short-text classification requires the availability of a large, domain-related dataset [LLHZ16], as they perform poorly with new or rare data [WWZY17].

4.3.3 Short Text in NFRs Classification

Most requirements statements are written in a short and concise form to support their understandability [Mey85]. According to Hooks [Hoo94], over-specification and unnecessary descriptions of requirements could result in cost overruns on a software project. The average length of requirements used in NFRs classification, as reported by S22 and S31, was about 20 words for FRs, and between 14 and 28 for NFRs. Nevertheless, little attention has been paid to handling this problem in classifying NFRs using supervised learning.

To the best of our knowledge, only one study (S19 by Lu and Liang) has clearly handled this problem. S19 proposed a feature extension method for classifying user reviews based on NFR types (reliability, usability, portability, and performance). They expanded each user review with the most K semantic similar words, extracted using word embedding (i.e., Word2vec). They found that the feature extension method with a Bagging classifier outperformed the other existing techniques (e.g., TF, TF-IDF, and CHI).

Besides S19, few studies consider this problem in determining text processing techniques. For example, S5 used a Boolean weighting schema due to the short length of requirements. The authors of S5 argued that NFRs are very short; thus, there is no need for a complex weighting schema such as TF-IDF. S37 concatenates each requirement by itself to enhance the process of measuring semantic similarity between requirements. However, S37 classified NFRs based on their similarity using Word2Vec (unsupervised method).

4.4 Discussion

This section discusses the existing solutions, identifies existing researches limitations and gaps, and highlights our research motivations.

4.4.1 Small and High-dimensional Requirement Datasets

High dimensionality is a prevalent problem in NFRs classifications. Although the statistical-based methods have been widely used to address this problem in NFRs classification, their shortcomings have also become apparent. First, their accuracy depends on the input of a large training corpus of annotated texts, as they use word frequencies in the text to establish the relevance of the words to the categories [DLWZ19]. In the domains like RE, this is not always possible due to the scarcity of the datasets [FDE+17, ZAF+21]. Second, these methods are semantically weak, as they ignore the morphological structures and semantic relationships in the text and only consider isolated, unconnected words present in a document text. Therefore, they do not account for semantic relationships and similarity of the words [KJMH+19]. For example, words "airplane," "aeroplane," "plane," and "aircraft" are often used in the same context, but TF-IDF cannot account for the similarity between these words in the document since each word is independently presented as an index [KJMH+19].

Word embedding methods, such as word2vec, has been used instead to reduce feature space size and discovering semantic relationships in the text corpus. However, such method is also statistical-based methods requiring large related dataset, as they rely on word co-occurrence frequencies to identify the syntactic and semantic similarity between words. Most of requirements datasets are too small (see Table 3.7) for word embedding. We tried to train a word embedding model on PROMISE-exp, but its performance was very poor. We also collected 3009 unlabeled requirements randomly from the internet and used them to train an embedding model, but also did not produce good performance in measuring similarity between words, and consequently to learn feature for feature selection (more in Appendix 2). We noticed that a small dataset, coupled with short-text requirements, leads to insufficient word co-occurrences in the dataset—increasing the difficulty of selecting informative features; this consequently hinders the effectiveness of the statistics-based feature selection method [MKIZ14, SYD$^+$14].

Another drawback of word embedding methods is that they are computationally expensive for training. For example, using one CPU, a word2vec model was trained on a subset of the Google News data in about a day to three days [MCCD13].

Despite the weaknesses of statistical-based methods, they were used most frequently than linguistic-based methods (see Figure 4.2). Linguistic-based feature selection methods could be the more suitable choice for selecting features for NFRs classification since these methods are more effective with small datasets and can understand the relationships between words. Motivated by these observations, we address high dimensionality by using a linguistically-based feature selection technique: a subset of features are selected based on the semantic and syntactic analysis of a set of requirements.

. This technique is described and evaluated in Chapter 5.

4.4.2 Imbalanced Multi-Class Requirements Classification

Although the diversity of solutions proposed to handle imbalance learning problems, most of the existing solutions are designed to address binary-class problems (i.e., imbalances exist between two classes) [HG09]. These solutions are less effective when dealing with multi-class classification tasks. Per experiments of Wang and Yao [WY12], resampling techniques (i.e., under- and over- sampling) showed strong negative effects in multi-class imbalanced classification. Oversampling causes overfitting the minority classes with low recall and high precision, while undersampling leads to loss of the performance of majority classes.

Resampling techniques have been used widely to handle the imbalance problem in NFRs classification. In many cases, these techniques were applied to the binary classification task. However, some studies applied such techniques in multi-class classification. These studies (S30, S40, and S51) did not provide detailed results (i.e., minority classes performance or overfitting level). Thus, according to Wang and Yao experiments [WY12], these techniques are more likely ineffective for multi-class NFRs classification.

One of the methods applied to address the multi-class imbalance problem is to use class decomposition, which converts a multiclass imbalance problem into a series of binary-class sub-problems. Then, a set of binary classifiers are applied to classify these sub-problems [GFB$^+$11]. Each classifier is individually trained without full data knowledge. Consequently, class decomposition can cause ambiguity in classification or uncovered data regions [WY12]. Besides, it might lead to an extremely uneven distribution in sub-problems. For example, by applying the one-versus-all class decomposition method, the combination of all classes that are not the most minor class causes the highest imbalance level [LSZS20].

Originally designed for classification with hierarchical class structures, hierarchical classification [KMNF06] has also shown promise for class imbalance problems in text classification [GIS10, ZZ20]. In hierarchical classification, classes are organized into a tree structure with levels and nodes [KMNF06]. Accordingly, the classification task is also divided into a set of sub-tasks corresponding to the number of nodes. The construction of a classification hierarchy can be informed by domain knowledge (e.g., relationships between the classes [GIS10]) or constraints (e.g., cost-sensitive factor [ZZ20]), with an aim to address the class imbalance problem. Ghazi et al. [GIS10] compared the performance of the flat and hierarchical classification approaches to an emotional classification in blog sentences. In the hierarchical approach, they used a two-level method. The first level contained the emotional and non-emotional categories, and the second represented six categories of emotional classes, e.g., happiness, sadness, and anger. They found that the hierarchical classification approach was better at dealing with highly imbalanced data.

Due to the difficulty of algorithm-level solutions, the ineffectiveness of data-level solutions, and the promising results of a hierarchical classification in handling the imbalanced class problem, we propose a hierarchical classification technique (i.e., data decomposition technique) to address the imbalanced multi-class NFRs classification. To the best of our knowledge, such a technique has not yet been applied in the context of requirements classification. Chapter 5 describes our technique and evaluation details.

4.4.3 Feature Extension in Requirements Classification

Requirements are short; expanding requirements have improved some RE tasks, such as requirement similarity (S37) and requirements classification (S19). Moreover, the idea of expanding requirements semantically has been widely used to handle problems with mismatches in requirements tracing [GF94, GCCH10, BDLO$^+$13, GCCH17]. For example, the requirements (or trace query) were expanded with new terms that were retrieved from web-mining results [GCCH10, BDLO$^+$13], domain knowledge and ontology [GMPCH14], and/or others [MN15, GCCH17]. These techniques were designed with the primary intent of adding terms to the original dataset to link two system artifacts together (e.g., the source code and requirement description). However, feature explanation in short text classification aims to add features that link requirements to each other with less noise; this handles sparsity and improves classification accuracy.

Expanding requirements from external resources would not be effective in requirement classification. As requirements documents are domain-specific texts, finding relevant terms from generic texts is difficult. For example, in WordNet, the terms "system" and "organization" are synonymous, but in requirement text, the term "system" is normally referred to as "software system", "software", and "application." In addition, using external resources (e.g., WordNet) for identifying synonyms can contribute to negative correlations between different terms in different requirements categories and thus lead to misclassification. For example, adding "late" as a synonym of "recent" can lead to the misclassification of an FR as an NFR and vice versa. To avoid these problems, techniques using external resources need to go through a careful vetting process to identify suitable datasets and suitable features.

Due to the difficulty of effectively use an external resource for handling the short text problem in NFRs classification, we use an internal resource instead. This thesis reports two proposed (internal-resource based) feature extension techniques: the first technique is reported in (Chapter 5), while the second technique in (Chapter 6). Each technique addresses the problem differently; however, both use the internal resource to expand features.

4.5 Summary, Main Conclusions, and Findings

This section provides an overview of three problems in NFRs classification. These problems are high dimensionality, imbalanced class problem, and short text classification. Each problem has specific causes and techniques of being addressed. For example, large numbers of words in the corpus cause high dimensionality, skewed class distribution causes a bias toward the majority class, while short requirements cause sparsity. To handle high dimensionality problem, existing methods are roughly based on feature reduction by either applying feature selection or feature extraction methods. Existing solutions for imbalance problems can be divided into two major groups: data sampling and classifier modification. Most of the existing solutions of short text classification focus on feature extension to overcome the sparseness problem.

The review of existing NFRs classifiers identified in the previous chapter shows that the high dimensionality problem is the problem that has been most commonly addressed, where statistical-based feature selection methods are the widely used solution. Followed by the imbalanced classes problem, which is clearly addressed by class data resampling techniques. In contrast, short text classification gets less attention with only one study that clearly addresses this problem.

In analyzing existing techniques and solutions applied in NFRs classification, we found that statistical-based methods for selecting features in NFRs classification are not often effective due to the small labeled requirements datasets. Data sampling techniques can also negatively affect requirement classification, as they cause loss of performance and overfitting. Finally, semantic expansion is useful to support several RE tasks (e.g., measuring semantic similarity and generating traceability links); however, using external resources is difficult, as requirements are domain-specific and naturally overlapped.

Based on this chapter's findings, the following chapter describes a proposed method that consists of three techniques, each of which addresses a single problem. High-dimensionality is addressed based on the linguistic-based feature selection method, class imbalance by using hierarchical classification, and semantic expansion using internal resources for short text classification problem.

Chapter 5

Dealing with Imbalanced, High Dimensional and Short Text Data in Machine Learning-Based Requirements Classification

"Problems Are Not Stop Signs, They Are Guidelines."

Robert Schuller

The previous chapter reviews three important problems related to ML-based requirements classification—high dimensionality, short text, and class imbalance problems—indicating the gaps in the existing solutions. Motivated by these gaps, we propose a decomposition-based requirements classification method to solve the three problems. The proposed method, called "ML4-RC" (Machine Learning For Requirements Classification), integrates three techniques: dataset decomposition, feature selection, and feature extension. Each technique is designed to address a single problem. For example, dataset decomposition for the class imbalance problem, feature selection for high dimensionality, and feature extension for short text problem. To assess the effectiveness of ML4RC, we conducted two experimental studies. The first study compares the entire ML4RC method with four closely related methods and the second evaluates the three key techniques individually against a baseline method, related NFRs classifiers, and common solutions.

This chapter is organized as follows: Section 5.1 introduces the ML4RC method in detail, explaining its processes and techniques. Section 5.2 implements ML4RC using four different algorithms and selects the ML algorithm with the best performance. Section 5.3 presents the experiment conducted to evaluate the performance of the ML4RC method as a whole against related work, and Section 5.4 evaluates each ML4RC technique to show how each technique works individually. Based on the experimental results, Section 5.5 discusses the key findings together with the opportunities and challenges for requirements classification. Section 5.6 discusses threats to validity. Finally, Section 5.7 presents a summary of this chapter.

5.1 The ML4RC Method

The ML4RC method is purposely designed to deal with the three problems—high dimensionality, short text and imbalanced data in requirements classification. It solves these problems using three key techniques, which are summarized below:

1. *Dataset Decomposition*: This technique aims to solve the imbalanced class problem by decomposing a given training dataset into two subsets, one containing the majority classes and another with the minority classes. In doing so, it breaks down the classification problems into two smaller problems. The hierarchical classification technique is then applied to perform the classification task.

2. *Semantic Role-based Feature Selection*: This technique aims to address the high dimensionality problem by identifying a small number of semantic roles to select the most relevant features from the requirements.

3. *Feature Extension*: This technique handles the short text problem by extending the features identified for each requirement with additional semantically related features.

These techniques are integrated into a coherent process depicted in Figure 5.1. The process resembles the text classification process shown in Figure 2.4 but with two additional steps: dataset decomposition in the training phase and feature extension in both the training and testing phases. The process steps of the ML4RC method and their enabling techniques are detailed in the sections below.

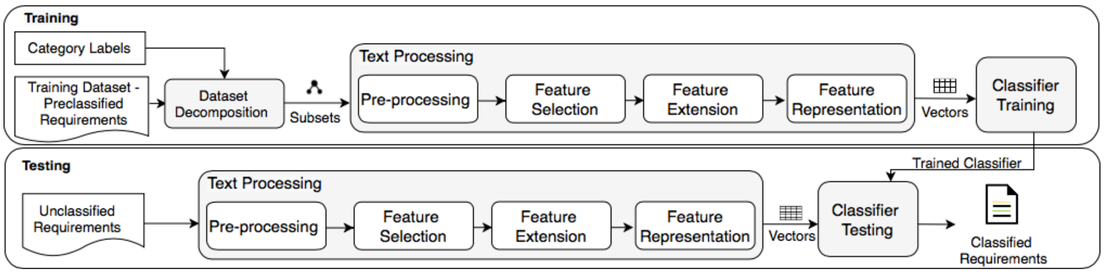


Figure 5.1: Overview of ML4RC - A decomposition-based machine learning method for requirements classification

5.1.1 Dataset Decomposition

This step, which only applies to the training phase, is to decompose the input training dataset into two subsets and one superset. The method for performing this decomposition is sketched in Algorithm 1. The input to this algorithm consists of a set of preassigned requirement-category pairs and the size of each category (i.e., the number of requirements in each category). Based on this input, the algorithm first sorts the requirements classes in descending order based on the category size. Then, the largest requirement-category pair (the one with the most number of requirements) are assigned to the subset S_{maj} (called the *major class subset*). This is repeated until the size of the major subset is greater or equal to half of the size of the training dataset for getting a relatively similar size for each subset. The remaining requirement-categories pairs (those with the smallest number of requirements) are assigned to the subset S_{min} (called the *minor class subset*).

After the subsets are divided, the algorithm assigns the subset-label (maj or min) to each requirement based on which subset it belongs to in order to identify the location of the original category. The created pairs (requirements, subset-label) are used to produce a superset S_{super}, as Figure 5.2 shows. These three sets, i.e., S_{maj}, S_{min} and S_{super} , will be used to train three classifiers for requirements classification. Figure 5.3 shows an example of applying the decomposition dataset technique on a training dataset.

5.1.2 Text Pre-Processing

This step applies to the training and testing phases. As described in Section 2.2, text pre-processing involves the application of various NLP techniques to the text data to remove unwanted, redundant, and superfluous words, and in doing so, to improve the

Algorithm 1: Dataset Decomposition Algorithm

```
Input :
    S={(r'_1,c_1),(r'_2c_2), ....(r'_m,c_k)}
        The set of preassigned requirement-category pairs in the training set.
    D={c_1=d_1,c_2=d_2,....,c_k=d_k, }
        The size of each category in the training set.
Output:
    $S_{min} \subset S$ and $S_{maj} \subset S$
        $S_{min}$ is the subset of the requirements in the minority categories in $S$ and $S_{maj}$ is the subset of
        the requirements in the majority categories in $S$.
        $S_{min} \cap S_{maj} = \{\Phi\}$ and $S_{min} \cup S_{maj} = \{S\}$
    $S_{superset}$ = {(r'_1,c'),(r'_2,c'), ....(r'_m,c')}, c' ∈{maj,min}
        $S_{superset}$ is a superset of all requirements associated with subset labels
1  $h = m/2$
       Compute $h$,which is the potential size of each subset.
2  $S_{min}$ = { }, $S_{maj}$ = { }.
       Initialize two subsets.
3  $D = \sum_{i=1}^{k} d, \{d \in D : d_i \geq d_{i+1}\}$
       Sort D in descending order.
4  for i=1 to i=k do
5      if $|S_{maj}| > h$ then
6          $S_{maj} = S_{maj} \cup \sum_{d=1}^{D[c_i]} (r', c_i)$.
7      end
8      else
9          $S_{min} = S_{min} \cup \sum_{d=1}^{D[c_i]} (r', c_i)$.
10     end
11 end
12 end For.
13 $S_{super}$ = { }.
       Initialize a super-set .
14 for j=1 to j=m do
15     if c from $(r',c)_j \in S_{maj}$ then
16         $S_{super}$= $S_{super}$ ∪ (r'_j,maj)
17     end
18     else
19         $S_{super}$= $S_{super}$ ∪ (r'_j,min)
20     end
21 end
22 end For.
23 Return $S_{super}$, $S_{min}$ and $S_{maj}$
```

* Time complexity = $O(n^2)$, Space complexity = $O(n)$

performance of feature selection and classification. Some commonly used NLP techniques are described in Chapter 3. In the ML4RC method, we apply the following NLP techniques to text pre-processing in order: tokenization, POS tagging, named entity recognition (NER), dependency parsing, lowercase conversion, lemmatization,

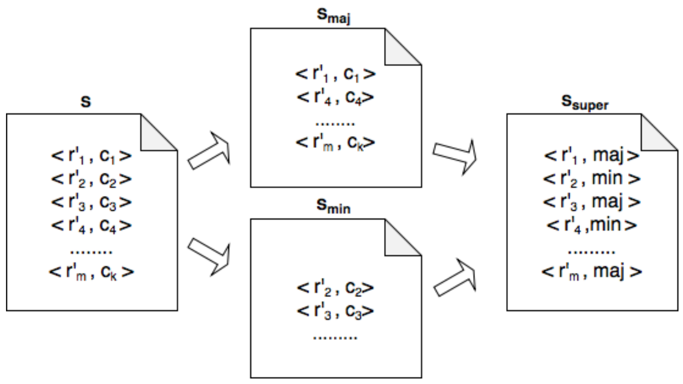


Figure 5.2: The dataset decomposition process. The original dataset is decomposed into two subsets: one with the majority classes and one with the minority classes. The superset is the union of the two subsets. The superset and two subsets are structured as a class hierarchy.

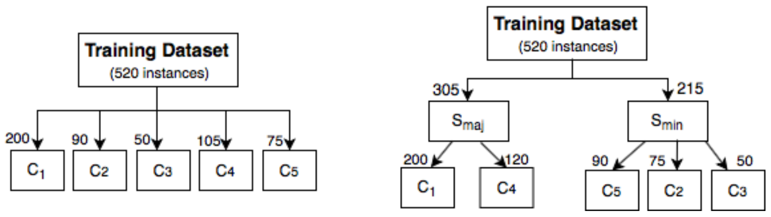


Figure 5.3: An example of flat and hierarchical structure of a training dataset; before and after dataset decomposition technique

stop-words removal, and finally, short-words removal (i.e., removing the words containing fewer than three characters). Dependency parser and POS tagger are applied here and before lemmatization and stop word removal to ensure the proper use of the grammatical structure of requirements.

5.1.3 Feature Selection

This step aims to select the most relevant features for each requirement. Each set of the selected features, known as a *feature set* [BKL09], should capture the basic information about the corresponding requirement. They can, therefore, be used to classify the requirement. Based on the findings of chapter 3 and 4, feature selection methods commonly used for requirements classification are based on statistical analysis techniques. Our feature selection method, on the other hand, is based on the identification of *semantic roles* in the requirements sentences.

In linguistics, semantic roles are shallow semantic representations that express the role of arguments or adjuncts of a predicate take in some event [JM20]. Arguments of a predicate are subjects and objects of verbs, whereas adjuncts of a predicate are adverbs, adjectives, and prepositional phrases. Semantic roles provide a level of representation for capturing the semantic commonality between sentences and help generalize over different surface realizations of predicate arguments or adjuncts [JM20]. Although there is no universally agreed-upon set of semantic roles, most of today's semantic role labeling methods [GJ02, Xue08] are based on the semantic roles used in FrameNet [BDCD19] and PropBank [PGK05]. To illustrate semantic roles, consider the following examples:

1. The system sent a message to the user.

2. The system should be easy-to-use.

3. The system should be available 24/7.

In the first sentence, the three arguments of the verb "*sent*" respectively play three semantic roles AGENT ("the system"), THEME ("a message") and GOAL ("the user"). In the second sentence, "the system" takes the semantic role AGENT, whereas the adjunct "easy-to-use" is an adjective playing the role MANNER. In the third sentence, "the system" takes the semantic role AGENT, whereas "available" is an adjective playing the role MANNER and "24/7" is an adverb playing the role DEGREE. These examples show that we can identify semantic roles through their grammatical features:

- AGENT/Subject
- GOAL/Object
- THEME/Object
- MANNER/Adverb, Adjective, Proposition
- DEGREE/Adjective, Number

We notice that the above five semantic roles are sufficient to answer the question "*Who (AGENT) did what (THEME) to whom (GOAL), how (MANNER) and how much (DEGREE)*". This question is central in understanding natural language [JM20] as it explains semantic relationships between a verb and its constituents. Such understanding is also key to requirements analysis. To assess the sufficiency of these roles for predicting requirement type, we manually analyze a set of requirements to identify the main element of each requirement. This analysis considers the predefined components of NFRs types (e.g., usability [GWSH09] and security [Fir04]) and lessons learned from related work to identify the main syntactic elements of each requirement [HKO08, AKG$^+$17]. The results of such analysis confirmed that these five semantic roles are relevant to feature selection.

However, the above five roles do not explicitly describe the event itself as none of them are realized by verbs—indeed, current semantic roles do not explicitly include verb roles. In RE, verbs that refer to actions, states, and events are central to functional requirements as these requirements specify *what* the system should *do*. Such *action verbs* are, therefore, the most relevant to requirements classification. Thus, we introduce ACTION as an additional role to our feature selection method. In total, we use six semantic roles for feature selection. These are:

1. **AGENT** - the volitional causer of an event [JM20]. This role is realized by the subjects. We notice that, in requirements modeling, the AGENT is also sometimes called ACTOR [RP92]. For example, in the requirement "**The system** shall send a verification email to the user when they log on to their account from an unfamiliar computer", "the system" plays the AGENT role.
2. **ACTION** - the cause of an action, event, or state. This role is realized by the verbs. For example, in the requirement "The system shall **send** a verification email to the user when they **log on** to their account from an unfamiliar computer", both "send" and "log on" play the ACTION role.
3. **THEME** - the participant most directly affected by an event [JM20] or who undergoes a change of state [Xue08]. In requirements modeling, THEME is referred to as *key object*; the subject matter of the essential system transaction, therefore, undergoes state change [SM98]. This role is realized by *direct objects*. For example, in the requirement "The system shall send a verification **message** to the user when they log on to their account from an unfamiliar computer", "message" takes the THEME role.
4. **GOAL** - the destination of an object of a transfer event [JM20] or a new state. In requirements modeling, GOAL describes a future, a required state which the system should satisfy, maintain, or sometimes avoid [SM98]. This role is realized by *indirect objects*. For example, in the requirement "The system shall send a verification email to the **user** when they log on to their account from an unfamiliar computer", "user" is the GOAL.
5. **MANNER** - the manner in which an action takes place [Xue08]. This role is

realized by adjectives, adverbs, determiners, or preposition phrases. For example, in the requirement "The system should be **easy-to-use**", "easy-to-use" is MANNER.

6. **DEGREE** - the degree of control exerted by the stimulus [JM20]. This role is typically realized by adverbs (e.g., rather) or numbers (e.g., 99%). For example, in the requirement "The system must be available to the users **98 %** of the time every month during business hours," 98 % is DEGREE.

These six roles can be automatically extracted using the combination of a POS tagger, a dependency parser, and NER tagger. In particular, the POS tagger is used to identify the part-of-speech tags of each requirement, the dependency parser produces a syntax tree for the requirement, and the NER tagger identifies special terms of the requirement. The syntactical features obtained from these tools can then be used to extract the six semantic roles as follows: POS tags for extracting these five roles: AGENT/ related to verbs, ACTION/action verbs, THEME/related to verbs, MANNER/adjectives, adverbs and preposition, and DEGREE/adverbs. Dependency parser helps in extracting all the six roles. For example, subject is used for extracting AGENT, object for both THEME and GOAL, and head-dependent relation for ACTION, MANNER, and DEGREE. The NE tagger is only used to extract DEGREE by identifying predefined entities such as date, time, number, and percent.

Table 5.1 summarizes these six semantic roles, their grammatical features and extraction rules for automatic identification. Figure 5.4 shows examples of how each main element of NFRs is selected.

However, in some cases, the same words may have more than one semantic role, and the extraction rules may result in the same words being extracted more than once. For example, “within 2 minutes” in “User can successfully register within 2 minutes” could be extracted as a MANNER (within) or DEGREE (within 2 minutes). To avoid to have duplicate features, we remove duplicate words from the selected features.

For the training phase, this step will be applied to each of the three decomposed datasets (S_{maj}, S_{min} and S_{super}), whereas for the testing phase, as the testing data are not decomposed, this step is applied only once to the entire testing set.

5.1.4 Feature Extension

This step aims to extend each feature set (the set of features selected for each requirement) with additional features to address the problem of feature sparsity caused by the

Role	**Grammatical Features**	**Extraction Rules**	**Explanation**
AGENT	Subjects	Identified as the subjects of the main verbs (the verbs with the head verbs)	If a term is the subject of the head verb, then, this subject is AGENT.
ACTION	Action verbs	Identified as the verbs with the head verb	If a term is the head verb, then this term is ACTION.
THEME	Direct objects	Identified as the objects of ACTION (main verbs)	If a term is the direct object of the main verb, then this object is THEME.
GOAL	Indirect objects	Identified as the objects of a dative preposition	If a term is an indirect object that its head is dative, then this object is GOAL.
MANNER	Adverbs, adjectives, determiners, proposition phrases	Identified as adjectives, adverbs, and determiners with their headwords, or prepositions with their dependents (i.e., children)	If a term is an adjective, adverb, or determiner, then this term and its headwords represents MANNER; otherwise, if a term is a preposition (e.g., from, with, without, after), then the preposition and all its dependents are MANNER.
DEGREE	Adverbs, numbers	Identified as the numbers and all its dependents (i.e., children) or adverbs and their headwords	If a term is a named entity (e.g., data, time, percent, money, and cardinal), then the term and all its dependents represents DEGREE; otherwise, if the term is an adverb, then this term and its headwords are DEGREE.

Table 5.1: The semantic roles used for feature selection in the ML4RC method

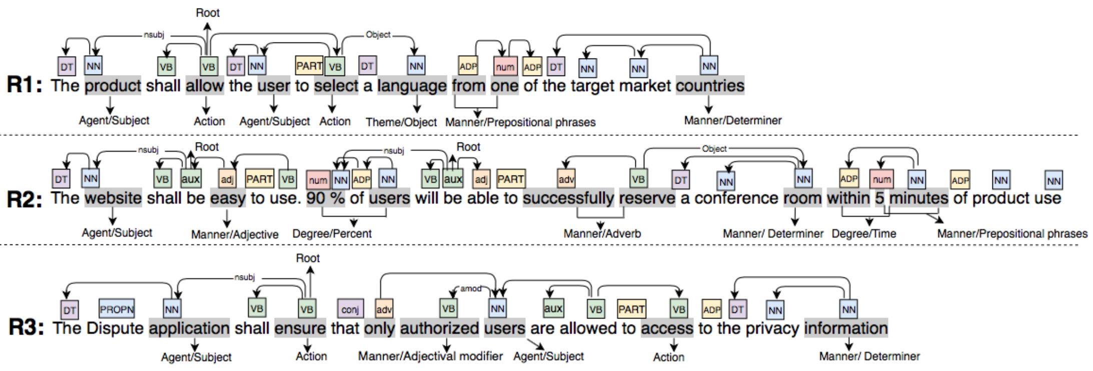

R1: The product shall allow the user to select a language from one of the target market countries
Agent/Subject
Action
Agent/Subject
Action
Theme/Object
Manner/Prepositional phrases
Manner/Determiner
R2: The website shall be easy to use. 90 % of users will be able to successfully reserve a conference room within 5 minutes of product use
Agent/Subject
Manner/Adjective
Degree/Percent
Manner/Adverb
Manner/ Determiner
Degree/Time
Manner/Prepositional phrases
R3: The Dispute application shall ensure that only authorized users are allowed to access to the privacy information
Agent/Subject
Action
Manner/Adjectival modifier
Agent/Subject
Action
Manner/ Determiner
Root
nsubj
Object
amod

Figure 5.4: Examples of how an input requirements are processed by feature extracting. The outputs of these Inputs are: "product allow user select language countries" , "website easy 90% users successfully reserve room 5 minutes", "application ensure only authorized users access information".

short text nature of the requirement. Ideally, the extended features should have a strong relationship with the original features [PNH08, SYD+14]. Thus, in this step, the extended features are synonyms of the original features; they are words with either the same or a highly semantically similar meaning that do not alter the general meaning of the requirement [MBF+90]. These synonyms are extracted using WordNet, a popular lexical database (thesaurus) for the English language. Examples of synonyms provided by WordNet for *error* are *mistake*, *erroneousness*, *computer error*, and *fault*.

Since not all the synonyms extracted from WordNet are relevant to requirements, our feature extension only uses the synonyms of a feature if they are also present in the training dataset. Practically, each word in the requirement tagged as a verb, noun, adjective, or adverb is used by WordNet to obtain its synonyms. Then, the extracted synonyms are assessed on whether they meet the condition of being present in the training dataset. If yes, they will be added at the end of each requirement.

Moreover, WordNet often repeats the synonyms retrieved for a particular term. For example, the synonyms extracted for the noun "System" are *system*, *system*, *arrangement*, *organization*, *habit*, and *system*. Keeping all these synonyms has a detrimental effect on classification accuracy as this redundancy might increase the weight of unimportant features and ignore important ones. Therefore, our feature extension technique ensures that duplicate synonyms are not added to the feature set.

To illustrate our feature extension method, assume that we have a very small training dataset consisting of two requirements:

1. "The system must be usable for people with visual impairment."

2. "The system shall provide handicap access."

An original requirement to be expanded with additional features is:

- "The system shall conform to the Americans with Disabilities Act."

The synonyms of each unique feature in this requirement are extracted from WordNet. These synonyms are then filtered by removing duplicates and selecting only those that exist in a training dataset. The selected synonyms will be added at the end of the original requirements to be:

- The system shall comply with the Americans with Disabilities Act system handicap impairment

"System" is a synonym of system, and "handicap" and "impairment" are synonyms of disability.

For the training phase, this step will be applied separately to each of the three decomposed datasets (S_{maj}, S_{min} and S_{super}). In the testing phase, this step is also applied by expanding the testing data with the synonyms of the training data used to train a specific classifier (either S_{maj}, S_{min} or S_{super}). The testing phase is detailed in Section 5.1.6.

Feature Representation

The features obtained in the previous steps need to be transformed into a vector representation that can be directly interpreted by a ML algorithm. This entails converting the features obtained from each requirement j into a vector of numerical features $\vec{V}_j = (w_{t_1,j}, w_{t_2,j}, w_{t_3,j}, ..., w_{t_k,j})$. In this vector, $w_{t_k,j}$ is the weight of feature t in requirement j, and k is the total number of unique features in a dataset. The feature weight $w_{t_k,j}$ is calculated using TF-IDF as follows:

$$w_{t_k,j} = tf_{t_k,j} \times \log \frac{N}{n_{t_1}} \tag{5.1}$$

where $tf_{k,j}$ is the frequency of feature t in requirement j, N is the total number of the requirements in a dataset, and n_{t_1} is the number of requirements containing feature t.

In the training phase, since we have three training datasets (S_{super}, S_{maj}, and S_{min}), we need to represent them separately. The resulting feature vectors from each dataset are arranged into a feature matrix. This leads us to three matrices M_{super}, M_{maj}, and

M_{min}. In these matrices, each row represents an individual instance (i.e., a particular requirement) in the corresponding training set with the columns being populated with the feature weights. As the three matrices represent three datasets, they have different sizes (i.e., different numbers of rows and columns) depending on the number of requirements and of features in their corresponding dataset. Clearly, M_{super} is the largest as it contains all the requirements combined from M_{maj} and M_{min}.

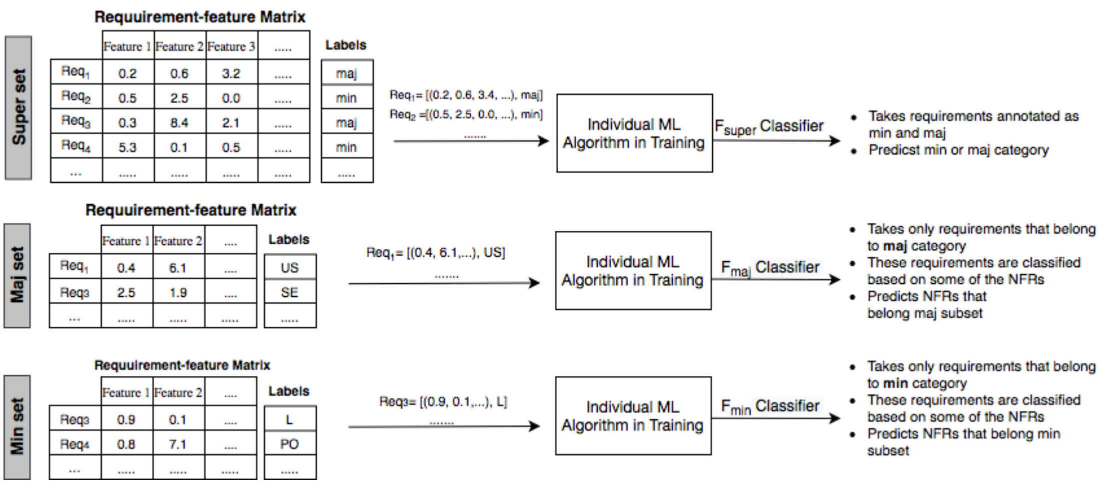
In the testing phase, the three training matrices are used to represent the testing data separately to form three corresponding testing matrices (T_{super}, T_{maj}, and T_{min}). Each testing matrix contains the same number of columns as the corresponding training matrix, but has a different number of rows, as the rows in the testing matrix represent the specific testing requirements in the testing data. The cells or elements of each matrix are populated by the TF-IDF weights of the features extracted from the testing requirements. Each feature weight is calculated by multiplying the feature frequency (TF) in a testing requirement by the feature IDF weight computed from the training dataset. The features in a testing requirement that have not been seen during the training phase are discarded and will not be represented in the testing matrices.

5.1.5 Classifier Training

This step is to train a hierarchical classifier F using a specific ML algorithm. The classifier consists of one super-classifier F_{super}, and two sub-classifiers F_{maj} and F_{min}, which are trained on the three feature matrices M_{super}, M_{maj}, and M_{min}, respectively. F_{super} is a *binary classifier* for classifying each requirement into either a major class subset (S_{maj}) or a minor class subset (S_{min}), whereas F_{maj} and F_{min} are two *multi-class classifiers* for predicting individual requirements categories. We will show how we select a ML algorithm for training in Section 5.2; here, we focus on the training process (see Figure 5.5)

The three classifiers are trained individually using a separate feature matrix as follows:

1. F_{super}: This binary classifier is trained on matrix M_{super} to learn the associations between the requirements in dataset S_{super} and their subset labels (i.e., maj and min). Based on the learning, this classifier is then applied to assigning a label "maj" or "min" to each requirement in the new data (e.g., the testing or application data). In doing so, this classifier divides the new data into two categories:


Super set
Requuirement-feature Matrix
Req₁= [(0.2, 0.6, 3.4, ...), maj]
Req₂ =[(0.5, 2.5, 0.0, ...), min]
....
Individual ML Algorithm in Training
F super Classifier
Takes requirements annotated as min and maj
Predicst min or maj category
Maj set
Requuirement-feature Matrix
Req₁= [(0.4, 6.1,...), US]
....
Individual ML Algorithm in Training
F maj Classifier
Takes only requirements that belong to maj category
These requirements are classified based on some of the NFRs
Predicts NFRs that belong maj subset
Min set
Requuirement-feature Matrix
Req3= [(0.9, 0.1,...), L]
....
Individual ML Algorithm in Training
F min Classifier
Takes only requirements that belong to min category
These requirements are classified based on some of the NFRs
Predicts NFRs that belong min subset

	Feature 1	Feature 2	Feature 3	.....	Labels
Req_1	0.2	0.6	3.2	.....	maj
Req_2	0.5	2.5	0.0	.....	min
Req_3	0.3	8.4	2.1	.....	maj
Req_4	5.3	0.1	0.5	.....	min
...	.....	.....	.....	.....	.....

	Feature 1	Feature 2	....	Labels
Req_1	0.4	6.1	....	US
Req3	2.5	1.9	....	SE
...	.....	.....	.....	.....

	Feature 1	Feature 2	....	Labels
Req3	0.9	0.1	....	L
Req4	0.8	7.1	....	PO
...	.....	.....	.....	.....

Figure 5.5: Classifier training process. The ML4RC classifier consists of one super-classifier and two sub-classifiers, which are trained respectively on the three decomposed datasets.

maj and *min*. This classifier thus performs the dataset decomposition conceptually.

2. F_{maj}: This multi-class classifier is trained on matrix M_{maj} to learn the associations between the requirements in dataset S_{maj} and their preassigned categories (i.e., the requirement-category pairs). Based on the learning, this classifier is then used to classify each requirement in the new data (e.g., the testing or application data) into a specific category label (e.g., functional, security or usability), thus performing the classification task on the new data.

3. F_{min}: This multi-class classifier is trained on matrix M_{min} to learn the associations between the requirements in dataset S_{min} and their preassigned categories (i.e., the requirement-category pairs). Based on the learning, this classifier is then used to classify each requirement in the new data (e.g., the testing or application data) into a specific category label that was not used in training F_{maj} (e.g., legal, portability, or maintenance), thus performing the classification task on the new data.

5.1.6 Classifier Testing

This step is to test the three pre-trained classifiers using a testing dataset, as illustrated in Figure 5.6. The testing process is described as follows.

First, the testing set will undergo the same text preparation steps as shown in Figure 5.1. Then the three classifiers will be tested as follows:

1. F_{super} is used to assign a label "maj" or "min" to each requirement in the testing set represented by matrix T_{super}.

2. For the requirements with the label "maj", F_{maj} is used to predict their categories. However, before applying the classifier, the preprocessed and filtered version of these requirements will undergo the feature extension and feature representation steps to ensure the testing data are expanded and transformed in a similar way as maj training data. This results in matrix T_{maj}.

3. Similarly, for the requirements with the label "min", F_{min} is used to predict their categories. As was done above, before applying the classifier, the preprocessed and filtered version of these requirements will undergo the feature extension and feature representation steps. This results in matrix T_{min}.

4. The predicted category labels from F_{maj} and F_{min} will then be compared with the original labels in the testing data to establish the performance of the classifiers. The performance measures used in the ML4RC method will be described in Section 5.2.

After testing, the three classifiers are ready for use. The application process is similar to the testing process, except that the application data are completely new and unrelated to the datasets used to train and test the classifiers.

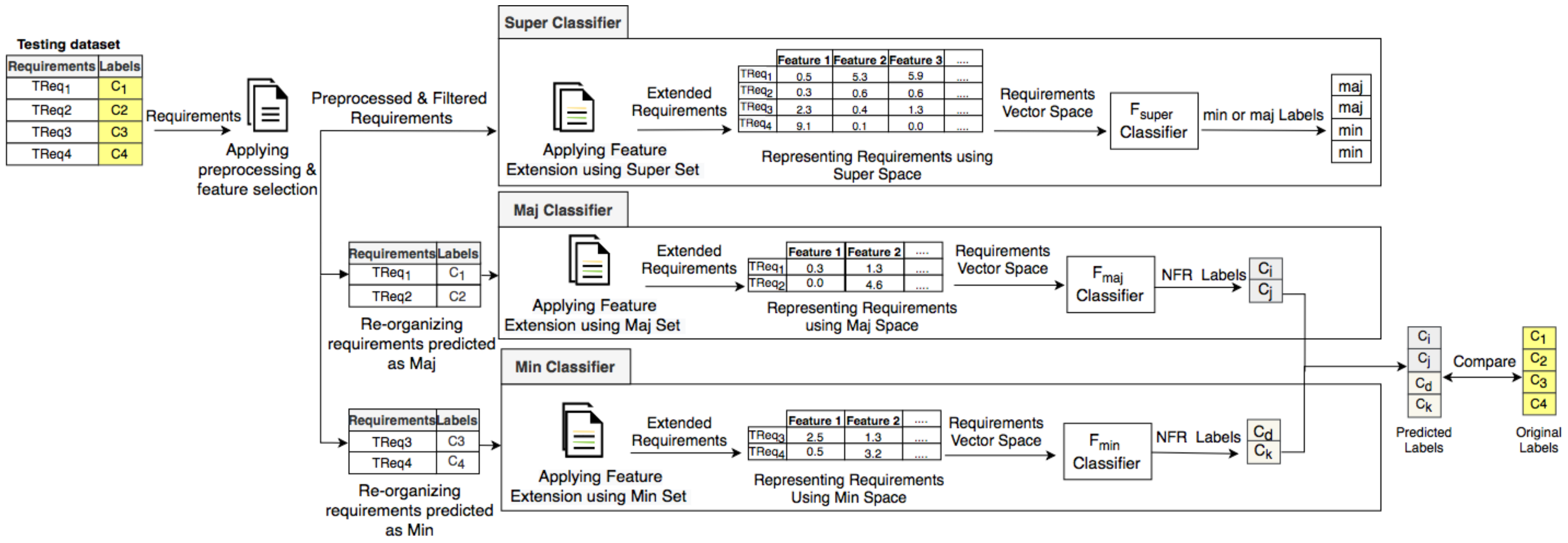

Testing dataset
Requirements
Labels
TReq1 C1
TReq2 C2
TReq3 C3
TReq4 C4
Requirements
Applying preprocessing & feature selection
Preprocessed & Filtered Requirements
Super Classifier
Applying Feature Extension using Super Set
Extended Requirements
Representing Requirements using Super Space
Requirements Vector Space
Fsuper Classifier
min or maj Labels
maj
maj
min
min
Maj Classifier
Re-organizing requirements predicted as Maj
Applying Feature Extension using Maj Set
Extended Requirements
Representing Requirements using Maj Space
Requirements Vector Space
Fmaj Classifier
NFR Labels
Ci
Cj
Min Classifier
Re-organizing requirements predicted as Min
Applying Feature Extension using Min Set
Extended Requirements
Representing Requirements Using Min Space
Requirements Vector Space
Fmin Classifier
NFR Labels
Cd
Ck
Compare
Predicted Labels
Original Labels

Figure 5.6: Classifier testing process. Three sub-classifiers are tested respectively on the three decomposed datasets.

5.2 Training and Testing ML Algorithms for ML4RC

Before we can evaluate the ML4RC method and its key techniques, we evaluate ML4RC with different ML algorithms (SVM, NB, DT, and KNN) to select the most effective one. SVM, NB, DT and KNN are the most frequent ML algorithms used in NFR classification (see Table 3.5), which showed promising results (see Table 3.18).

5.2.1 Dataset and Experimental Settings

We used the PROMISE-exp dataset to train the four algorithms. PROMISE-exp is one of the publicly available NFRs datasets listed in Table 3.7. We chose this dataset as it was released recently and is publicly available [1]. In addition, it is an extended version of PROMISE [CHMLP07], a widely used dataset in NFR classification (see Table 3.6). The original PROMISE dataset contains 12 categories and 625 instances (i.e., requirements) collected from 15 requirements documents, whereas the PROMISE-exp dataset has 12 categories and 969 requirements collected from 47 requirements documents. Table 5.2 illustrates the categories of PROMISE-exp and provides examples for each category.

A summary of PROMISE-exp statistics is given in Table 5.3. It is worth noting from Table 5.3 that the PROMISE-exp dataset exhibits the problems of class imbalances, short text, and high dimensionality. More specifically, 1) the requirements in this dataset are unevenly distributed across different categories, with "Functional" being the largest category containing 444 instances, and "Portability" being the smallest with only 12 instances; 2) the average length of the requirements in this dataset is 22.4 words (with a standard deviation of 17.03), which is rather short; 3) the number of unique features of this dataset is 2129 which results in a high dimensional feature space of 969×2129.

[1]https://tinyurl.com/PROMISE-exp

Category	Definition	Examples of Requirements Statements
Functional (F)	Requirements that specify what a system or system component should do [15990].	F1: The user shall be able to download appointments and contact information for clients. F2: The system shall filter data by: Venues and Key Events. F3: The product shall store new conference rooms.
Security(SE)	A system's ability to protect data and information against any unauthorized access or modification while maintaining usability for authorized persons or systems [II04].	SE1: The system shall prevent malicious attacks including denial of service. SE2: Only authorized users shall have access to clinical site information. SE3: The product shall free of computer viruses.
Usability (US)	The capacity of the system to allow a specific user to effectively and efficiently achieve precise goals with satisfaction [ISO18].	US1: The product shall use a standard navigation menu familiar to most web users. US2: The system shall be used by realtors with no training. US3: The product shall be intuitive and self-explanatory. 90% of new users shall be able to start the display of Events or Activities within 90 minutes of using the product.
Operational (O)	This category (known as environmental requirements) specifies the environment in which the system works and considers some circumstances created by the environment which might affect the system operation [RR12].	O1: The product shall run on the existing hardware for all environments. O2: For estimators, the product shall be able to be operated in a repair facility during dirty and noisy conditions. O3: The product must support Internet Explorer 5.5 and above.

Category	Definition	Examples of Requirements Statements
Performance(PE)	A requirement that specifies the conditions for a certain function, such as speed, level of accuracy, or memory usage [15990].	PE1: The System shall allow for a minimum of 6 users to work at the same time. PE2: The product shall respond fast to keep up-to-date data in the display. PE3: The product shall synchronize with the office system every hour.
Look and feel (LF)	Requirements that specify a system's appearance and the impression the user has of the system when using it [RR12].	LF1: The website shall be attractive to all audiences. The website shall appear to be fun and the colors should be bright and vibrant. LF2: The website design should be modern clean and concise. LF3: The system shall have a professional appearance.
Availability (A)	A software's ability to work as required at a given point in time [II04].	A1: The system shall be available for use between the hours of 8am and 6pm. A2: All movies shall be streamed on demand at any time of the day. A3: The software is available for use from the supermarket opening time to the closing time.
Maintainability (MN)	A system's ability to be modified. Examples of this type of modification can include corrections, improvements, or changes to the system's environment and requirements [II04].	MN1: Promotional updates to the website should take a day to update. MN2: The application should be easy to extend. The code should be written in a way that it favors implementation of new functions. MN3: Changes made to the Manage My ID website can be adopted without altering the iOS application.
Scalability (SC)	A system's capacity to tolerate the differences or scales in certain characteristics that affect its execution [DRW07].	SC1: The server will support a maximum of 1,000 simultaneous users. SC2: The system should be able to scale up to 500 concurrent users (if there is a need in the future) by installing additional hardware components. SC3: The system shall be capable of processing 100% of nursing students and their classes for the next 10 years.

Category	**Definition**	**Examples of Requirements Statements**
Fault tolerance (FT)	A system's ability to maintain a certain level of performance in case of failure [II04].	FT1: The product shall retain user preferences in the event of a failure. FT2: The website shall continue to operate if the payment gateway goes down. FT3: 100% of saved user preferences shall be restored when system comes back online.
Legal (L)	Requirements that specify the laws and standards that are applied to a system [RR12].	L1: The product must comply with Sarbanes-Oxley. L2: The PHP code will comply with PEAR standards. L3: The product shall comply with insurance regulations regarding claims processing.
Portability (PO)	The ability of a system to run and transfer from one environment to another [II04].	PO1: The product is expected to run on Windows CE and Palm operating systems. PO2: The application should be portable with iOS and Android. PO3: The system will support mobile users in some way.

Table 5.2: The categories of requirements in PROMISE-exp dataset

Category	#Reqs.	Percent	Avg Len.	#Feat.
Functional (F)	444	45.82	16.43	1070
Security (SE)	125	12.90	18.08	554
Usability (US)	85	8.77	21.36	467
Operational (O)	77	7.95	19.12	450
Performance (PE)	67	6.91	22.58	390
Look & feel (LF)	49	5.06	19.57	330
Availability (A)	31	3.20	18.64	186
Maintainability (MN)	24	2.48	25.12	251
Scalability (SC)	22	2.27	19.27	156
Fault tolerance (FT)	18	1.86	18.03	167
Legal (L)	15	1.55	16.08	115
Portability (PO)	12	1.24	13.25	85
Total	**969**	**100.00**		

Table 5.3: Summary of the PROMISE-exp dataset

We trained these algorithms on a standard laptop with an Intel Core i5 1.6 GHz and 8 GB RAM. The entire process of the ML4RC method was mapped onto a pipeline system [MKMG97] and each process step onto a software component. The implementation was carried out using the Python programming language (version 3.6.5), with the support of Python's NLP and ML toolkits for performing the following tasks:

- Pre-processing and Feature Extension were implemented by NLTK's text processing module and POS tagger [2].
- Feature Selection implemented by SpaCy's dependency parser [3].
- Feature Representation, Classifier Training and Classifier Testing implemented by scikit-learn [PVG+11][4].

The four ML algorithms were implemented using the corresponding scikit-learn's pre-trained ML classification models (known as estimators):

- The Linear Support Vector Classification model for the SVM algorithm
- The Gaussian Naive Bayes model for the NB algorithm
- The Decision Tree Classifier (with gini impurity splitting criterion) for the DT algorithm

[2]https://www.nltk.org
[3]https://spacy.io
[4]https://scikit-learn.org/stable/index.html

- The KNeighbors Classifier (with Euclidean distance measure) for the KNN algorithm

Each of these classification models provides a *fit* method, which can accept an input data array and an array of labels for supervised learning as its arguments [PVG+11]. Thus, through this method, each model can be fitted to different training datasets.

5.2.2 Training and Testing Process

We apply a 10-fold cross-validation process [BG04, AC+10] to train and test each of the four ML model as follows:

1. Apply scikit-learn's StratifiedKFold tool to divide the PROMISE-exp dataset randomly into 10 folds (i.e.10 portions), nine portions for algorithm training and one portion for testing;

2. Decompose each portion of the training data into three training sets (i.e., S_{super}, S_{maj}, and S_{min}), respectively used to train the three classifiers (i.e., F_{super}, F_{maj}, and F_{min}).

3. Carry out Pre-Processing, Feature Selection, and Feature Extension tasks as detailed in Section 3 (A total of 1365 features were selected from 2129 unique features; Table 5.4 lists the top 10 features selected for each classifier.)

4. Use scikit-learn's TfidfVectorizer tool for Feature Representation, which automatically converts each feature set into a TF-IDF vector.

5. Train three classifiers for each algorithm and use scikit-learn's GridSearchCV with a 5-fold cross validation to tune the parameters of the model under training to maximize the performance score of each classifier. These classifiers are then trained using the optimal parameters chosen in the grid search.

6. Test the trained classifiers on the testing set as described in Section 5.1.6, which produces the predicted classification results (i.e., a set of predicted category labels).

7. Compare the predicted labels with the true labels provided in the original PROMISE-exp dataset and measure the difference between the true label and the predicted label for each category using the Precision (P), Recall (R), and F1-Score ($F1$)

Table 5.4: Top 10 features used by the three ML4RC classifiers

Classifier	**Top 10 Features for Each Classifier**
F_{super}	theft, password, edit, sensitive, easy, feel, environment, error, ethnic, confirm
F_{maj}	sensitive, securely, unauthorized, password, transaction, address, store, feel, feedback, time
F_{min}	space, support, available, feel, easy, degradation, environment, time, simultaneous, delay

metrics. The overall performance (P, R, and $F1$ metrics) of the classifier is then measured by computing the macro and micro average of all the classes.

This process is repeated 10 times for each algorithm—nine times on the nine portions of the training data and one time on the testing data. Afterwards, the mean P, R, and $F1$ values for each category as well as overall macro (mean) and micro averages for all categories are computed. In Table 5.5, we list both macro and micro averages of the four algorithms, but only use the macro averages as the basis for comparison of classification performance. The micro-average of P, R, and $F1$ are equal in multi-class classification as it is computed over all classes; thus, the incorrectly predicated samples represent both false positive and false negative. For example, in the case of "A is misclassified as B", it is false positive for B and false negative for A. A more detailed explanation of macro and micro is provided in Section 2.2.3.

During the incremental training of each algorithm, we plotted its training curve to show how well the algorithm can be adapted to different numbers of training samples by measuring the performance of the trained algorithm in classifying the training set. Similarly, we also plotted a testing curve for each algorithm to show how well the algorithm can generalize to new data [Bro14]. These learning curves are depicted in Figure 5.7.

In the section below we discuss the performance and learning curves of the SVM, NB, DT, and KNN algorithms.

5.2.3 Results

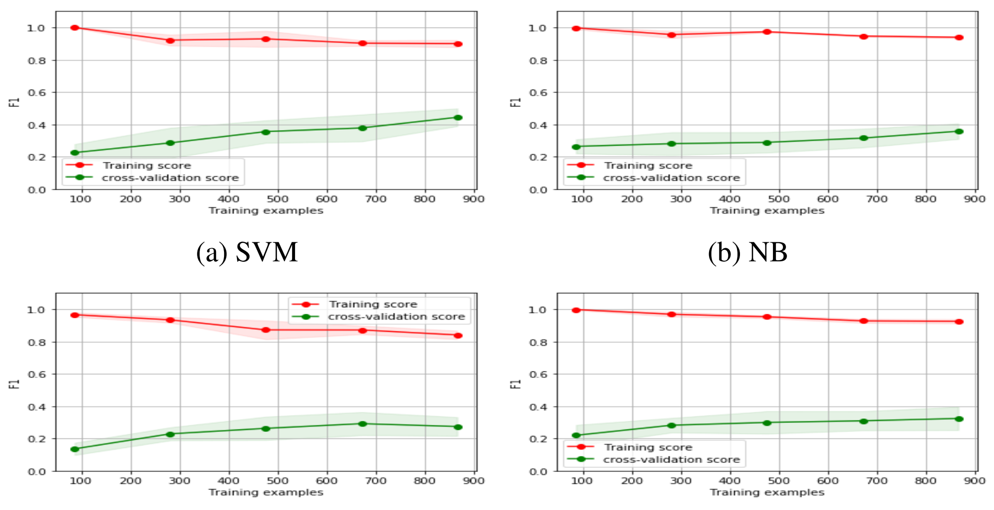
Classification Performance

It is clear from Table 5.5 that overall, among the four algorithms used to support the ML4RC method, SVM has achieved the best test results, with a macro average of P = 49%, R = 47%, and $F1$ = 46%. The results confirmed previous findings that SVM

Cat.	**SVM**			**NB**			**DT**			**KNN**		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
F	0.76 ±0.07	0.82 ±0.13	0.78 ±0.08	0.68* ±0.06	0.60* ±0.14	0.63* ±0.10	0.66* ±0.12	0.69* ±0.18	0.67* ±0.14	0.69* ±0.8	0.85 ±0.13	0.76 ±0.09
SE	0.66 ±0.19	0.52 ±0.20	0.57 ±0.18	0.38* ±0.14	0.53 ±0.17	0.43* ±0.12	0.51 ±0.12	0.42 ±0.19	0.44* ±0.14	0.69 ±0.27	0.41 ±0.23	0.49 ±0.22
US	0.53 ±0.14	0.54 ±0.19	0.53 ±0.16	0.43 ±0.22	0.41 ±0.17	0.41* ±0.18	0.37* ±0.15	0.39* ±0.17	0.37* ±0.16	0.53 ±0.10	0.52 ±0.16	0.52 ±0.12
O	0.37 ±0.12	0.53 ±0.19	0.43 ±0.12	0.30 ±0.16	0.26* ±0.13	0.27* ±0.13	0.20* ±0.16	0.29* ±0.28	0.22* ±0.20	0.36* ±0.16	0.47 ±0.27	0.39* ±0.20
PE	0.72 ±0.10	0.68 ±0.15	0.69 ±0.12	0.37* ±0.14	0.50* ±0.17	0.42* ±0.15	0.56 ±0.20	0.54 ±0.22	0.57 ±0.18	0.49* ±0.26	0.42* ±0.27	0.42* ±0.24
LF	0.61 ±0.29	0.47 ±0.30	0.49 ±0.25	0.42* ±0.18	0.42* ±0.18	0.40* ±0.17	0.19 ±0.20	0.12 ±0.13	0.14 ±0.15	0.45 ±0.43	0.22* ±0.26	0.28* ±0.29
A	0.74 ±0.39	0.67 ±0.39	0.67 ±0.36	0.72 ±0.33	0.36 ±0.18	0.45 ±0.17	0.35* ±0.25	0.50 ±0.37	0.40* ±0.28	0.39* ±0.30	0.60 ±0.36	0.45* ±0.30
MN	0.25 ±0.34	0.20 ±0.26	0.21 ±0.27	0.09 ±0.18	0.15 ±0.32	0.11 ±0.21	0.10 ±0.15	0.13 ±0.22	0.11 ±0.17	0.10 ±0.03	0.05 ±0.15	0.07 ±0.20
SC	0.44 ±0.42	0.42 ±0.42	0.41 ±0.39	0.42 ±0.42	0.37 ±0.33	0.38 ±0.35	0.37 ±0.44	0.28 ±0.33	0.30 ±0.35	0.38 ±0.38	0.35 ±0.33	0.33 ±0.28
FT	0.25 ±0.40	0.20 ±0.33	0.22 ±0.35	0.25 ±0.40	0.20 ±0.33	0.20 ±0.31	0.20 ±0.33	0.25 ±0.40	0.22 ±0.35	0.20 ±0.40	0.10 ±0.20	0.13 ±0.27
L	0.45 ±0.47	0.40 ±0.44	0.40 ±0.42	0.43 ±0.47	0.35 ±0.39	0.37 ±0.41	0.18 ±0.32	0.20 ±0.33	0.17 ±0.26	0.10 ±0.30	0.05 ±0.15	0.07 ±0.20
PO	0.05 ±0.15	0.10 ±0.30	0.07 ±0.20	0.28 ±0.39	0.35 ±0.45	0.30 ±0.40	0.02 ±0.08	0.05 ±0.15	0.3 ±0.10	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
Mic.	**0.65** ±0.08	**0.65** ±0.08	**0.65** ±0.08	**0.49** ±0.07	**0.49** ±0.07	**0.49** ±0.07	**0.50** ±0.10	**0.50** ±0.10	**0.50** ±0.10	**0.60** ±0.07	**0.60** ±0.07	**0.60** ±0.07
Mac.	**0.49** ±0.22	**0.47** ±0.21	**0.46** ±0.20	**0.40** ±0.17	**0.38** ±0.13	**0.36** ±0.13	**0.31** ±0.19	**0.32** ±0.18	**0.32** ±0.17	**0.37** ±0.22	**0.34** ±0.25	**0.33** ±0.22
Ex. Time	2 Minutes			53.62 Seconds			57.66 Seconds			56.1 Seconds		

* The Wilcoxon signed-rank test indicates that the performance of the SVM algorithm is statistically significantly higher than the other three algorithms in at least 4 categories, with the probability $p < 0.05$.

Table 5.5: Test results of the SVM, NB, DT, and KNN algorithms with the ML4RC method based on 10-fold cross-validation



(a) SVM

(b) NB

(c) DT

(d) KNN

Figure 5.7: Learning Curves of the SVM, NB, DT, and KNN Algorithms Based on Macro F1-score using 10-Fold Cross-Validation.

performs better than KNN and NB for requirements classification [SW13, LHG$^+$18]. SVM has also been widely recognized as one of the most effective algorithms for text classification tasks [YL99].

For individual requirement categories, SVM performs well on Functional, Performance, Availability, Security, and Usability, achieving over 60% on the first three categories across the board and over 50% on the last two categories across the board. For Look & Feel, SVM achieves a good precision (61%), but with a lower recall (47%) and, subsequently, a lower F1-score (49%). SVM performs poorly on the remaining six categories, with the worst performance in Portability ($P = 5\%$, $R = 10\%$, $F1 = 7\%$). We noticed that, among the six poorly performed categories, five are the minor classes with an average number of instances = 18.2, and one (Operational) is the major class with 77 instances. We also noticed that the worst-performed category, Portability, is the smallest class with only 12 instances. These observations suggest that, under the ML4RC method, SVM performs well on predicting the major classes (except the Operational category) and not well on the minor classes (except the Availability category). These two exceptional categories are interesting because, while Operational is a major class, its features may not be intuitive enough for the classifier; on the other hand, while Availability is a minor class, its features are the most intuitive. We will return to this discussion in Section 5.4 when we assess the effectiveness of our feature selection and feature extension techniques.

To confirm the statistical significance of the performance of the SVM algorithm over the other three, we use the Wilcoxon signed-rank test [Wil92] to statistically compare the performance metrics between SVM and the other three algorithms. The test indicates that the performance of SVM is statistically significantly higher than the other three algorithms on at least four requirements categories, with the probability $p < 0.05$ (see the stared numbers in Table 5.5).

However, while SVM is more effective than the other three algorithms, it incurs a slightly higher computational time as it takes more than two minutes to train. By contrast, the training of the other three algorithms takes less than one minute. There is, therefore, a trade-off between the effectiveness and efficiency of these algorithms.

Learning Curves

The training curves of the four algorithms (i.e., the red curves) in Figure 5.7 show that all four algorithms start with a high training score (with $F1 = 0.95$ for DT; $F1 = 1.0$ for SVM, NB, and KNN). These curves gradually slide downwards as the training progresses (i.e., with more training samples), resulting in a slight decrease in $F1$-score (with $F1 = 0.95$ for NB and KNN; $F1 = 0.90$ for SVM and DT). By contrast, the testing curves of these algorithms (i.e., the green curves) start with a low score ($F1 \leq 0.20$ for SVM, DT, and KNN; $F1 = 0.25$ for NB) and gradually climb upwards as the training proceeds (with more training samples). Among them, SVM moves upwards faster than other algorithms, with $F1 > 0.45$ for SVM at the end of the training and $F1 < 0.40$ for the other three algorithms. This suggests that SVM is generalizing better than the other algorithms. Thus, potentially adding more training examples can help improve the performance of SVM.

5.3 Experimental Comparison of Related Methods

This section compares the performance of the ML4RC method that incorporates SVM as the classification algorithm with related methods. It seeks to fulfill the following research objective:

> To investigate the effectiveness and efficiency of ML4RC (i.e., the combination of the three techniques) compared with closely related methods.

To do so, we compare ML4RC with the following four methods:

1. The method by Cleland-Huang et al. [CHSZS07]

2. The method by Kurtanović and Maalej [KM17]
3. The method by Yin et al. [YGX$^+$13]
4. The method by Lu and Liang [LL17]

There are two main reasons for selecting these four methods for comparison: First, the process description of these methods is relatively clear or can be inferred from the description so that we can reconstruct these methods based on their description and objectively assess the performance of these methods against ours. Second, these methods are closely related to ours in the following ways: The method by Cleland-Huang et al. [CHSZS07] has been used as a basic method for evaluating ML-based requirements classification by the RE community [ROW13, HKO08, ZYWS11]. We thus use it as our basic method. The methods by Kurtanović and Maalej [KM17], and Yin et al. [YGX$^+$13] have explicitly addressed the class imbalance problem. Although the method by Yin et al. [YGX$^+$13] was originally designed for high-dimensional data classification in general, we re-purposed it for requirements classification as it uses a class decomposition strategy that complements ours. Finally, the method by Lu and Liang [LL17] seems to be the only method that has explicitly addressed the short text problem in the context of requirement classification. These methods are summarized in Table 5.6.

5.3.1 Experiment Execution

To conduct a fair comparison, we re-implemented all three methods using the same environment as described in Section 5.2. The PROMISE-exp dataset and 10-fold cross-validation were used for classifier training and testing. We implemented these methods from scratch based on their original descriptions [KM17, YGX$^+$13, LL17]. Below, we present how we execute each of these three methods through software implementation.

Implementing the Basic Method

The basic method by Cleland-Huang et al. [CHSZS07] uses a probability function to determine the likelihood that each new requirement might belong to a certain NFR type. A multi-class classifier is trained to classify a given requirement into one of the 10 categories on the original PROMISE dataset. Similar to our method, this method is also divided into training and testing phases. During the training phase, the probability function computes a probabilistic weight for each potential indicator term (i.e.,

Ref.	Dataset	ML Algorithm	Key Techniques	Classifier Training & Testing
Basic Method				
Cleland-Huang et al. [CHSZS07]	PROMISE dataset (10 categories, 684 requirements)	Probability function	*Feature selection:* a probability function to calculate the probabilistic weight for each potential indicator term for each category	A multi-class classifier is trained to classify a given requirement into one of the 10 categories; leave-one-out cross-validation for classifier training and testing, with an average of $P = 0.14$ and $R = 0.77$
Addressing the class imbalance problem				
Kurtanović and Maalej [KM17]	PROMISE dataset	SVM with linear kernel (binary version)	*Solution for imbalanced classes*: Over-sampling used to balance the major NFR classes; *Feature selection*: Different types of features (meta-data, texts, and syntax features) are selected according to their importance.	A multi-class classifier is trained to classify a requirement into one of these four categories: Usability, Security, Operational, and Performance; 10-fold CV for classifier training and testing, with the highest $F1 = 0.74$ for classification of Usability, 0.74 for Security, 0.71 for Operational, and 0.82 for Performance.
Yin et al. [YGX$^+$13]	4 datasets (a total of 433 samples from medical diagnosis problems & other sources	SVM with linear kernel	*Solution for imbalanced classes*: Use *k-means clustering* to divide the major classes into smaller pseudo-subclasses with relatively uniform sizes.	A binary classifier is trained to classify an input text into one of two categories; 5-fold CV for classifier training and testing, with $F1 = 0.65$ for minor classes and $F1 = 0.91$ for major classes
Addressing the short text classification problem				
Lu and Liang [LL17]	4000 user review sentences from iBooks and WhatsApp	DT with Bagging method	*Feature extension*: Word2Vec used to expand each user review with the most similar words; *Feature representation*: Bag of Words	A multi-class classifier is trained to classify a given user review into one of these four types of NFRs: Reliability, Usability, Portability, and Performance; 10-fold CV for classifier training and testing, with $P = 0.71$, $R = 0.72$, $F1 = 0.72$

Table 5.6: Related Methods Used to Compare ML4RC

keyword), which represents the importance of the term relative to a particular NFR category. The weight given to each term is calculated based on the standard information retrieval assumption that the term relevant to a certain NFR type must also be present in the requirements document to be classified. Consequently, the frequency at which the term occurs in the document is an important factor in determining the weight of the term.

During the testing phase, each requirement is classified into a certain NFR type if it contains a certain number of indicator terms belonging to that type. Requirements receiving the classification scores above a certain threshold for a given NFR type will be classified into that type, and all unclassified requirements are assumed to be functional requirements.

Cleland-Huang et al. [CHSZS07] employed leave-p-out cross-validation (LPOCV) [AC^{+}10] for classifier training and testing (where $p = 1$). Like k-fold CV, LPOCV creates k-fold partitions to be fed to the classifier k times; $(k - 1)$ fold for training and 1 fold for testing. The split of the training and testing in each fold is based on the number of requirements documents (i.e., projects) in the PROMISE training set, rather than the number of requirements; thus, k equals the number of requirements documents.

We implemented this method as faithfully as possible. As no implementation details were provided by its authors, we used Python as the main implementation tool, complemented by NLTK for the text pre-processing task (stop-word removal and stemming). The test results of this method are presented in Table 5.7. The learning curves of this method are plotted in Figure 5.8.

It should be noted that we could not use 10-fold cross-validation for this basic method due to its probability function's dependency on the number of requirements documents. Therefore, we cannot split the dataset based on the number of requirements in the dataset as we would have done with 10-fold cross-validation. Consequently, we had to use the same cross-validation technique (LPOCV) as in the original method. To be able to compare our method with the basic method, we retrained it using LPOCV (as our method was originally trained using 10-fold cross-validation, as described in Section 5.2). The test results of our method using LPOCV are presented alongside those of the basic method in Table 5.7. The learning curves of our method are plotted in Figure 5.8.

Cat.	**Basic (Probability-based)**			**ML4RC (SVM-based)**		
	P	*R*	*F*1	*P*	*R*	*F*1
F	0.46 ±0.44	0.08*±0.14	0.13*±0.02	0.72 ±0.15	0.78 ±0.21	0.71 ±0.15
SE	0.41*±0.20	0.44*±0.18	0.39*±0.16	0.74 ±0.19	0.65 ±0.19	0.68 ±0.16
US	0.18 ±0.16	0.46 ±0.36	0.25 ±0.21	0.35 ±0.31	0.42 ±0.02	0.33 ±0.24
O	0.12*±0.13	0.41 ±0.32	0.16*±0.12	0.27 ±0.17	0.60 ±0.33	0.32 ±0.14
PE	0.47 ±0.38	0.43 ±0.33	0.42 ±0.30	0.49 ±0.38	0.49 ±0.37	0.46 ±0.34
LF	0.15*±0.30	0.16 ±0.23	0.09*±0.12	0.42 ±0.39	0.34 ±0.38	0.33 ±0.32
A	0.29 ±0.29	0.60 ±0.43	0.37*±0.31	0.53 ±0.40	0.55 ±0.42	0.53 ±0.40
MN	0.21 ±0.33	0.19 ±0.31	0.16 ±0.21	0.23 ±0.4	0.18 ±0.31	0.19 ±0.32
SC	0.08 ±0.23	0.02 ±0.08	0.04 ±0.11	0.16 ±0.33	0.07 ±0.14	0.09 ±0.18
FT	0.00 ±0.00	0.00*±0.00	0.00*±0.00	0.21 ±0.33	0.18 ±0.22	0.18 ±0.25
L	0.05 ±0.15	0.02 ±0.05	0.02 ±0.08	0.10 ±0.03	0.01 ±0.04	0.02 ±0.08
PO	0.05 ±0.15	0.10 ±0.30	0.07 ±0.20	0.3 ±0.10	0.10 ±0.30	0.5 ±0.15
Mic.	**0.26** ±0.07	**0.26** ±0.07	**0.26** ±0.07	**0.58** ±0.15	**0.58** ±0.15	**0.58** ±0.15
Mac.	**0.21** ±0.16	**0.24** ±0.20	**0.18** ±0.14	**0.38** ±0.20	**0.36** ±0.24	**0.36** ±0.21
Ex.	39 Minutes			1.8 Minutes		

* P-values significant at alpha < 0.05

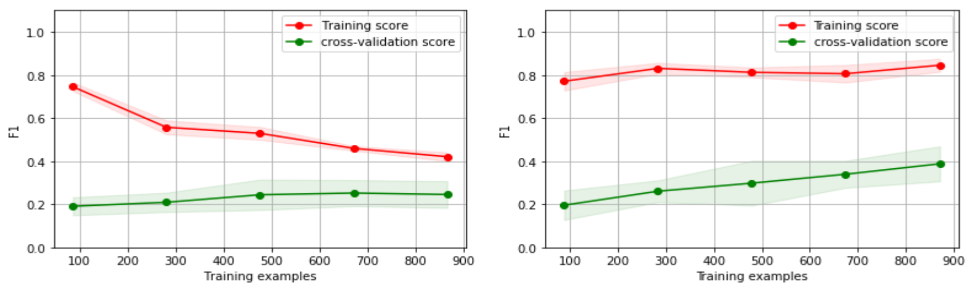
Table 5.7: Test results of the basic [CHSZS07] and ML4RC methods using Leave-*P*-Out Cross-Validation

Implementing the Method by Kurtanović and Maalej [KM17]

This method classifies a given requirement into one of these four categories: Usability, Security, Operational, and Performance. This method has explicitly addressed the class imbalance problem. However, in contrast to our method, it only addresses the two-class imbalanced problem. The two classes were derived from the PROMISE dataset: the majority class consists of 67 Usability requirements and the minority class consists of 54 Performance requirements. The majority class is reduced through under-sampling, which randomly removes some samples (i.e., requirements) from the class, whereas the minority class is enlarged through over-sampling by adding supplementary samples derived from Amazon software reviews.

For text pre-processing, Kurtanović and Maalej applied NLTK and the Stanford Parser [5] for removal of punctuation, stop-words, and lemmatization. This method selects 11 types of feature, including textual features (e.g., word collections, such as bigrams and trigrams), meta-data (e.g., text length), and syntax features (e.g., POS tags). These features are ranked according to their importance, which is assessed using an ensemble of tree classifiers, and the top 10 features are selected and used for classification. The built-in SVM model in scikit-learn is used to train four binary classifiers

[5]http://nlp.stanford.edu/software/lex-parser.shtml



(a) Basic (b) ML4RC

Figure 5.8: Learning curves of the basic and ML4RC methods Based on Leave-P-Out Cross-Validation.

and one multi-class classifier that work in unison to classify the four most frequent NFRs in the PROMISE dataset: Usability, Security, Operational, and Performance.

To implement this method, we adapted the source code provided by Dalpiaz et al. [DDAÇ19] [6]. As this source code was for a binary classification system, we changed it into a multi-class classification system. We noticed that Dalpiaz et al. made several modifications to the original method by Kurtanović and Maalej. One was feature selection, as they could not reproduce one specific type of feature, called CP unigrams, due to lack of sufficient description on how this type of feature can be identified. Another modification was the tool for building the parse trees, as they used Berkley's benepar library rather than the Stanford parser. The third modification was over-sampling. Since the Amazon software reviews used for over-sampling in the original method were artificial, they did not implement this technique. We adapted their first two modifications in our implementation, but implemented a random over-sampling technique based on imbalanced-learn, a python package offering different re-sampling techniques [7], due to the unavailability of the Amazon software reviews dataset used in the original dataset.

We trained the SVM model of Kurtanović and Maalej's method using 10-fold cross validation and the test results of this method are presented in Table 5.8.

Implementing the Method by Yin et al. [YGX$^+$13]

This method is closely related to ours in that it is also based on data decomposition. However, while our method decomposes the dataset, Yin et al.'s method decomposes the majority classes into relatively balanced pseudo-subclasses. For feature selection,

[6]`https://github.com/explainable-re/RE-2019-Materials`, last accessed December 2020
[7]https://pypi.org/project/imblearn/

Yin et al. apply the goodness of feature with pseudo-labels (e.g., Fisher-based or mutual information method). Different numbers of features are selected and released from the pseudo-labels to the original labels. A binary classifier is built using the selected features and applied to different high-dimensional datasets (including texts).

As this method does not provide implementation details, we made the following modifications in our implementation:

- Implementation tools: We used Python as the main implementation tool, NLTK for pre-processing and clustering the major classes with cosine distance (via the KMeansClusterer module), and scikit-learn for feature selection, representation and classification.

- Pre-processing: We applied stemming and stop-word removal for pre-processing and representation, as the original method did not include pre-processing and transformation steps.

- Feature selection: We use the mutual information method for feature selection and SVM for classification. These techniques were chosen because they showed high performance and suitability in our preliminary experiments when compared to other techniques mentioned in Yin et al.'s study (e.g., Fisher and correlation-based methods for feature selection, DT and NB for classification).

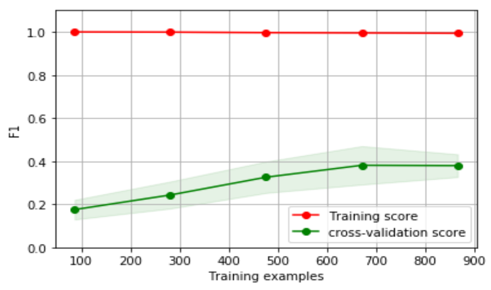
- Classifier training: We trained a multi-class classifier based on SVM instead of a binary classifier as in the original method.

Table 5.8 displays the test results of this method.

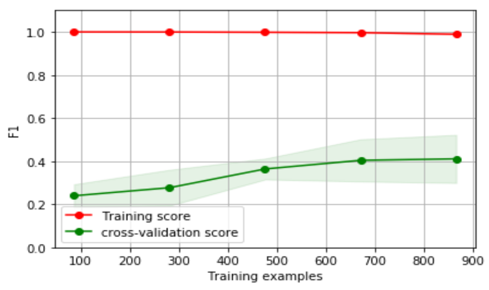
Implementing the Method by Lu and Liang [LL17]

This method classifies a given user review into one of these four types of NFRs: Reliability, Usability, Portability, and Performance. To the best of our knowledge, it is the only requirements classification method that has explicitly addressed the issue of short text classification. It does so by expanding each user review with the most related words in an external dataset using a word2vec model [8] that has been trained using 21969 user review sentences. Then, top *N* words that are ranked according to their similarity values are added to the end of the user review. *N* is the user review length

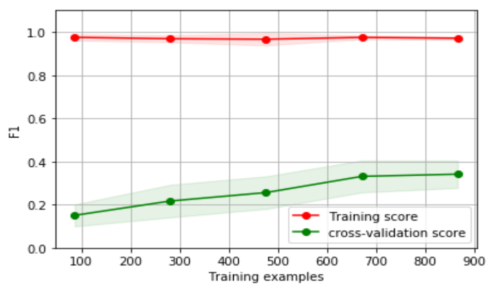
[8]https://code.google.com/archive/p/word2vec/



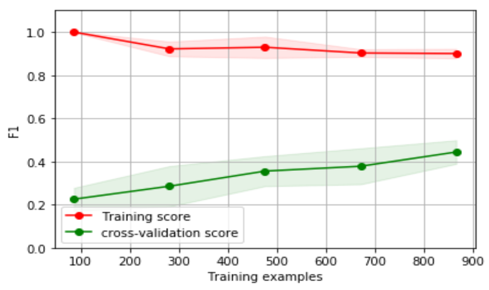
(a) Kurtan Maalej



(b) Yin et al.



(c) Lu and Liang



(d) ML4RC

Figure 5.9: Learning Curves of Kurtanović and Maalej [KM17], Yin et al. [YGX$^+$13], Lu and Liang [LL17], and ML4RC Methods Based on 10-Fold Cross-Validation.

multiplied by θ. This feature extension method is used to classify user reviews into NFR categories using a DT-based bagging classifier.

The original method was implemented using Weka [9]. To be consistent with our implementations of the other methods (including ours) and due to the unavailability of its source code, we re-implemented this method using Python complemented with NLTK and scikit-learn. In addition, as the word2vec model used by Lu and Liang is not available, we replaced it with a pre-trained Google News word2vec model released by Mikolov et al. [MCCD13]. This model is publicly available and showed higher performance in our preliminary experiments for classifying NFRs than another available model that was proposed by Efstathiou et al. [ECS18].

We trained the DT model of this method using 10-fold cross validation and the test results of this classifier is displayed in Table 5.8.

[9]https://www.cs.waikato.ac.nz/ml/weka/

C	**Kurtanović&Maalej**			**Yin et al.**			**Lu & Liang**			**ML4RC**		
	P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$
F	0.73 ±0.11	0.81 ±0.11	0.76 ±0.05	0.71 ±0.07	0.83 ±10	0.77 ±0.07	0.69* ±0.04	0.88 ±0.09	0.77 ±0.04	0.76 ±0.07	0.82 ±0.13	0.78 ±0.08
SE	0.72 ±0.19	0.56 ±0.15	0.61 ±0.14	0.69 ±0.22	0.60 ±0.17	0.63 ±0.17	0.63 ±0.15	0.56 ±0.15	0.57 ±0.10	0.66 ±0.19	0.52 ±0.20	0.57 ±0.18
US	0.49 ±0.15	0.42 ±0.18	0.43 ±0.16	0.56 ±0.12	0.40* ±0.22	0.44* ±0.19	0.45 ±0.20	0.40* ±0.20	0.41* ±0.18	0.53 ±0.14	0.54 ±0.19	0.53 ±0.16
O	0.37 ±0.26	0.29* ±0.18	0.28* ±0.14	0.40 ±0.18	0.43 ±0.24	0.40 ±0.17	0.34* ±0.15	0.30* ±0.16	0.31 ±0.15	0.37 ±0.12	0.53 ±0.19	0.43 ±0.12
PE	0.59* ±0.11	0.60 ±0.12	0.58* ±0.09	0.73 ±0.23	0.57 ±0.21	0.60 ±0.17	0.76 ±0.21	0.61 ±0.21	0.66 ±0.18	0.72 ±0.10	0.68 ±0.15	0.69 ±0.12
LF	0.41 ±0.36	0.26* ±0.22	0.28* ±0.21	0.52 ±0.21	0.33 ±0.20	0.36 ±0.13	0.36 ±0.39	0.14* ±0.13	0.18* ±0.17	0.61 ±0.29	0.47 ±0.30	0.49 ±0.25
A	0.40* ±0.27	0.44* ±0.31	0.38* ±0.23	0.45 ±0.32	0.60 ±0.36	0.49 ±0.31	0.49 ±0.34	0.44* ±0.27	0.34* ±0.26	0.74 ±0.39	0.67 ±0.39	0.67 ±0.36
MN	0.30 ±0.31	0.28 ±0.24	0.28 ±0.24	0.52 ±0.45	0.40 ±0.38	0.43 ±0.37	0.35 ±0.45	0.17 ±0.21	0.22 ±0.28	0.25 ±0.34	0.20 ±0.26	0.21 ±0.27
SC	0.32 ±0.38	0.37 ±0.40	0.30 ±0.32	0.40 ±0.38	0.40 ±0.38	0.38 ±0.35	0.45 ±0.47	0.22 ±0.22	0.29 ±0.30	0.44 ±0.42	0.42 ±0.42	0.41 ±0.39
FT	0.28 ±0.39	0.20 ±0.24	0.22 ±0.28	0.23 ±0.40	0.20 ±0.33	0.18 ±0.28	0.20 ±0.40	0.10 ±0.20	0.13 ±0.27	0.25 ±0.40	0.20 ±0.33	0.22 ±0.35
L	0.43 ±0.47	0.40 ±0.44	0.41 ±0.44	0.28 ±0.39	0.35 ±0.45	0.28 ±0.37	0.38 ±0.43	0.40 ±0.44	0.37 ±0.40	0.45 ±0.47	0.40 ±0.44	0.40 ±0.42
PO	0.03 ±0.10	0.10 ±0.30	0.05 ±0.15	0.10 ±0.20	0.20 ±0.40	0.13 ±0.27	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.05 ±0.15	0.10 ±0.30	0.07 ±0.20
Mic	**0.60** ±0.06	**0.60** ±0.06	**0.60** ±0.65	**0.63** ±0.08	**0.63** ±0.08	**0.63** ±0.08	**0.62** ±0.04	**0.62** ±0.04	**0.62** ±0.04	**0.65** ±0.08	**0.65** ±0.08	**0.65** ±0.08
Mac	**0.42** ±0.19	**0.39** ±0.19	**0.38** ±0.19	**0.47** ±0.19	**0.44** ±0.17	**0.42** ±0.18	**0.43** ±0.20	**0.35** ±0.24	**0.35** ±0.21	**0.49** ±0.22	**0.47** ±0.21	**0.46** ±0.20
Ex. Time	15.2 minutes			47.7 seconds			3.1 minutes			2 minutes		

* Indicates a significant statistical difference between the ML4RC and the other classifiers with p<0.05. Wilcoxon signed-rank test has been applied.

Table 5.8: Test results of the four related methods: Kurtanović & Maalej [KM17],Yin et al. [YGX$^+$13], Lu abd Liang [LL17] and ML4RC based on 10-fold cross-validation

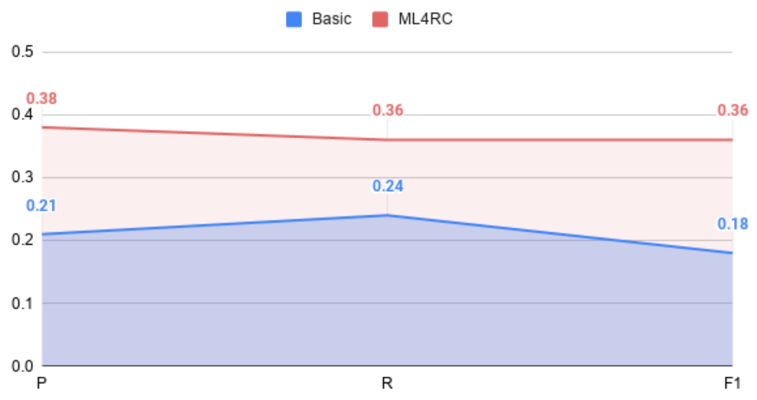


Figure 5.10: Overall performance comparison between basic and ML4RC.

5.3.2 Experimental Results

Comparison with the Basic Method

Classification Performance. As shown in Table 5.7, apart from the bottom two categories (Legal and Portability), ML4RC outperforms the basic method in all the other categories. ML4RC performs particularly well on the Functional and Security categories, and produces encouraging results on the Availability category. A comparison of the overall performance of basic and ML4RC is depicted in Figure 5.10, which shows that ML4RC has made a substantial improvement over the basic method across the board with a 17% increase in precision, 12% increase in recall, and 18% increase in F1-score.

For individual requirement categories, ML4RC performs far better than the basic method on Functional, Security, and Availability, and slightly better on the Performance category. By itself, ML4RC performs well on the major classes (except the Operational category) and not well on the minor classes (except the Availability category). The same observation is reported in Section 5.2.3.

Statistically, the Wilcoxon signed-rank test shows that ML4RC has made a significant improvement over the basic method in the F1-score for the Functional, Security, Operational, Look & feel, and Availability categories. We noticed that the basic method was unable to identify any Look & feel requirements, due to the limitation in its feature selection technique, that is, the relevance of a term (feature) depending on its occurrence in the number of requirements documents. Finally, in terms of the execution time, it took about 60 minutes to cross-validate the basic method, whereas it only took about two minutes for ML4RC.

Learning Curves. When examining the learning curves of the basic method (Figure 5.8a), it can be observed that its training curve starts with a high score (F1 $\geq 0.75\%$) and declines gradually to ($F1 > 0.4$) throughout the training; its validation curve starts with a low score (F1 $\leq 0.20\%$) and slightly increases to (F1 $\leq 0.25\%$). It then stays fixed. This indicates the inability to learn from the training data (i.e., fit the training data), known as the underfitting problem. Adding more examples to the basic model is unlikely to lead to improvement in its performance, but changing the model by, for example, using a different algorithm, will.

On the other hand, it can be observed in Figure 5.8b that the training curve of the ML4RC method starts with a high score ($F1 <= 0.80$) and slightly increases to (F <= 0.86) as the training progresses (i.e., with more training samples); by contrast, its validation curve starts at a very low score ($F1 = 0.20$) but gradually moves upwards as the training proceeds ($F1 > 0.40$). This indicates that ML4RC suffers from high variance (a large gap between training and validation score) and bias (low validation score), known as the *overfitting*. Adding more training samples is very likely to lead to improving ML4RC's performance.

Comparison with Three Related Methods

Classification Performance. Table 5.8 shows that ML4RC has the best performance, followed by Yin et al's method, then Kurtanović and Maalej's method, and finally Lu and Liang's method. This seems to suggest that the two methods with explicit data decomposition perform better than the two without, as the method by Lu and Liang is based on DT, whereas the remaining three are SVM-based. The relatively poor performance of Lu and Liang's method may also be due to its learning algorithm (see Section 4 for the discussion on the performance of different algorithms). A comparison of the overall performance of these methods is depicted in Figure 5.11.

For individual requirement categories, all four methods perform similarly on most categories. Notably, all four methods are very good at identifying the Functional category, with a high recall value (over 80%); ML4RC stands out in classifying the Availability category and produces good results compatible with Lu and Liang's method on the Performance category. Similar to ML4RC, the other three methods also show a tendency toward performing better on the major classes and worse on the minor classes.

Statistically, the Wilcoxon signed-rank test shows that ML4RC has made a significant improvement over Kurtanović and Maalej's method on the Operational, Performance, Look & feel, and Availability categories; it also improves upon Lu and

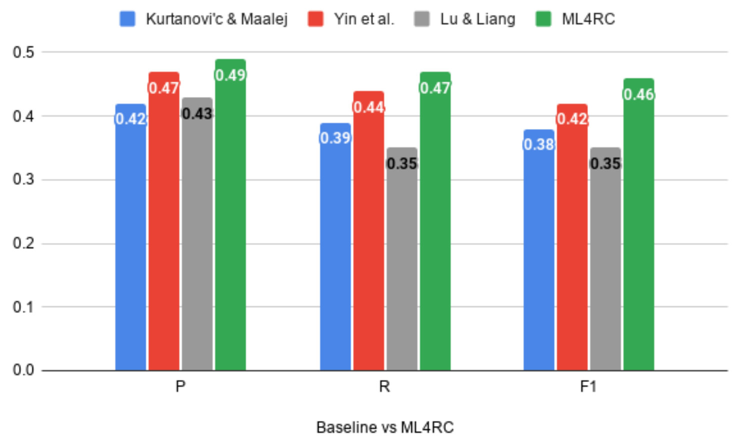


Figure 5.11: Overall performance comparison between four related methods.

Liang's method on Usability, Operational, Look & feel, and Availability, and Yin et al.'s method on the Usability.

In terms of execution time, apart from Yin et al.'s method, which took 48 seconds to run and is the fastest, ML4RC is more efficient than the other two methods. The execution time is based on the time spent on 10-fold cross validation without considering the time used for tuning the parameter, as some classifiers have more parameters than others.

Learning Curves. When examining the learning curves of these methods (Figure 5.7), it can be observed that their training curve starts very high ($F1 = 1.0$); as the training progresses, the training curve of ML4RC declines gradually, whereas the other three training curves stay largely unchanged, resulting in a big gap between training and cross-validation scores (i.e., a high variance), indicating that the other methods fit the training data very well but are difficult to generalize on the data not used for training (known as *over-fitting*).

The testing curves of all the four methods show an upward tendency in model generalization. The curve of ML4RC, however, is more prominently going upwards, suggesting that ML4RC has more potential to generalize well, if the size of the training data continues to increase.

5.4 Experimental Evaluation of the Key Techniques of ML4RC

In Sections 5.2 and 5.3, we evaluated the performance of the ML4RC method as a whole, whereas in this section we evaluate the performance of the three key techniques underpinning ML4RC: Dataset Decomposition (DD), Feature Selection (FS), and Feature Extension (FE). The main objectives of this evaluation are:

- To investigate the usefulness of the three proposed techniques (HC, FE, and FS) versus a baseline ML approach applied with no additional technique for addressing the three problems (high dimensionality, short-text classification, and class imbalance).
- To demonstrate the effectiveness of the techniques in in relation to a widely used technique applied to address the problems of high dimensionality and imbalance class.

To fulfill these objectives, we incorporate each of these techniques into a multi-class classifier and then experimentally compared each performance with a baseline classifier and related techniques. Therefore, our evaluation contains two main parts: a comparison with the baseline method and one with other widely used techniques. The following sections are organized based on these parts (aforementioned objectives).

5.4.1 Comparison with the Baseline Method

In this section, we experimentally evaluate each of the three key techniques of ML4RC against a baseline method to assess the performance of each technique. We implement the following classification systems to facilitate our evaluation:

1. *The Baseline Classifier*: This is the baseline classifier for comparisons. This classifier does not include any of the three key techniques in ML4RC.
2. *The DD Classifier*: This classifier only includes the dataset decomposition technique in its implementation.
3. *The FS Classifier*: This classifier only includes the semantic role-based feature selection technique in its implementation.

4. *The FE Classifier*: This classifier only includes the feature extension technique in its implementation.

In the subsections below, we present this experimental study and the results obtained.

Experiment Execution

The same implementation environment as described in Section 5.2.1 was used. We implemented the four classifiers individually, each trained and tested using the 10-fold cross validation process. The implementation process for the Baseline classifier is given below:

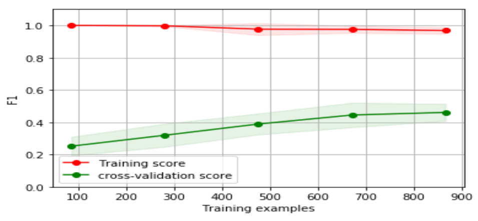
1. PROMISE-exp is randomly divided into 10 folds (nine folds for training and one for testing).
2. For each fold of the training data, the requirements in that fold are pre-processed and converted into TF-IDF vectors, i.e., all the words are used as features in the learning space with no feature selection or extension.
3. The converted requirements and associated labels are used to tune SVM hyperparameters (e.g., C parameter) by using grid search with 5-fold cross-validation. Then, the classifier is trained using the optimal hyperparameters.
4. The testing dataset is also pre-processed, converted into TF-IDF, and used to test the trained classifier, which produces a set of predicted requirement category labels.
5. The predicted labels are compared with the true labels provided in the original PROMISE-exp dataset, and the difference between the true and predicted labels for each category is measured using P, R, and $F1$ metrics. Besides, the macro-average and micro-average of these values (all classes) are calculated.
6. Steps 2-5 are repeated nine times. Afterward, the average P, R, and $F1$ values for each category are computed, as well as the mean of these average values.

This process was adapted to the implementation of the remaining three classifiers by adding a new step appropriate for each classifier, adding a dataset decomposition step for the DD classifier, a feature selection step for the FS step, etc. The 10-fold cross validation test results of these classifiers are presented in Table 5.9.

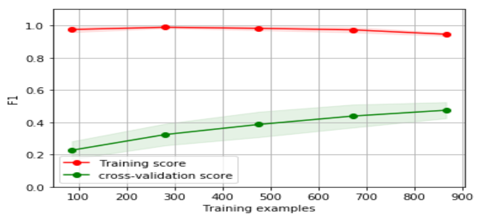
Cat.	**Baseline Classifier**			**DD Classifier**			**FS Classifier**			**FE Classifier**		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
F	0.69 ±0.04	0.83 ±0.12	0.75 ±0.05	0.79* ±0.05	0.74* ±0.17	0.75 ±0.09	0.68 ±0.03	0.85 ±0.01	0.76 ±0.05	0.71 ±0.05	0.87 ±0.12	0.78* ±0.05
SE	0.69 ±0.19	0.65 ±0.15	0.64 ±0.13	0.74* ±0.15	0.63 ±0.16	0.66 ±0.09	0.64 ±0.19	0.60 ±0.17	0.60 ±0.16	0.66 ±0.02	0.63 ±0.15	0.62 ±0.14
US	0.62 ±0.22	0.51 ±0.23	0.55 ±0.22	0.52 ±0.10	0.62 ±0.22	0.55 ±0.12	0.59 ±0.18	0.49 ±0.20	0.53 ±0.18	0.63 ±0.21	0.54 ±0.26	0.57 ±0.22
O	0.41 ±0.13	0.46 ±0.26	0.42 ±0.17	0.36 ±0.15	0.55 ±0.24	0.43 ±0.16	0.46 ±0.14	0.42 ±0.21	0.42 ±0.15	0.44 ±0.15	0.43 ±0.26	0.42 ±0.20
PE	0.88 ±0.13	0.65 ±0.22	0.71 ±0.18	0.67* ±0.19	0.69 ±0.13	0.66 ±0.14	0.87 ±0.18	0.69 ±0.16	0.76 ±0.15	0.86 ±0.17	0.63 ±0.18	0.71 ±0.16
LF	0.69 ±0.37	0.39 ±0.19	0.46 ±0.22	0.51 ±0.23	0.57 ±0.30	0.49 ±0.21	0.60 ±0.39	0.33 ±0.25	0.39 ±0.26	0.74 ±0.24	0.45 ±0.20	0.54 ±0.20
A	0.70 ±0.38	0.67 ±0.37	0.66 ±0.35	0.72 ±0.38	0.70 ±0.38	0.70 ±0.36	0.76 ±0.40	0.63 ±0.38	0.66 ±0.36	0.74 ±0.39	0.67 ±0.37	0.68 ±0.35
MN	0.45 ±0.47	0.33 ±0.39	0.37 ±0.40	0.44 ±0.42	0.37 ±0.37	0.37 ±0.35	0.35 ±0.39	0.23 ±0.24	0.27 ±0.28	0.63 ±0.46	0.45 ±0.39	0.49 ±0.39
SC	0.55 ±0.47	0.40 ±0.38	0.45 ±0.40	0.55 ±0.45	0.45 ±0.42	0.49 ±0.43	0.57 ±0.47	0.48 ±0.42	0.51 ±0.44	0.67 ±0.45	0.53 ±0.39	0.57 ±0.39
FT	0.25 ±0.40	0.20 ±0.33	0.20 ±0.31	0.25 ±0.34	0.25 ±0.34	0.23 ±0.29	0.25 ±0.40	0.20 ±0.33	0.22 ±0.35	0.25 ±0.40	0.20 ±0.33	0.22 ±0.35
L	0.45 ±0.47	0.40 ±0.44	0.40 ±0.42	0.45 ±0.47	0.40 ±0.44	0.40 ±0.42	0.47 ±0.46	0.45 ±0.42	0.43 ±0.40	0.45 ±0.47	0.45 ±0.47	0.43 ±0.45
PO	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.05 ±0.15	0.10 ±0.30	0.07 ±0.20	0.10 ±0.30	0.10 ±0.30	0.10 ±0.30	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
Mic	**0.65** ±0.07	**0.65** ±0.07	**0.65** ±0.07	**0.66** ±0.05	**0.66** ±0.05	**0.66** ±0.05	**0.65** ±0.05	**0.65** ±0.05	**0.65** ±0.05	**0.68** ±0.06	**0.68** ±0.06	**0.68** ±0.06
Mac	**0.53** ±0.04	**0.46** ±0.05	**0.47** ±0.04	**0.50** ±0.05	**0.51** ±0.07	**0.48** ±0.06	**0.53** ±0.06	**0.46** ±0.05	**0.47** ±0.05	**0.56** ±0.05	**0.49** ±0.05	**0.50** ±0.05
Ex. Time	3.3 minutes			1.3 minutes			2.5 minutes			3.5 minutes		

* Indicates a significant statistical difference between the baseline and another classifier with p<0.05. Wilcoxon signed-rank test has been applied.

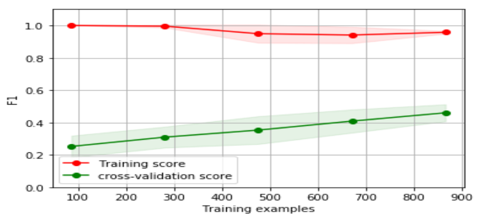
Table 5.9: 10-Fold Cross-Validated Test Results for the Four Classifiers



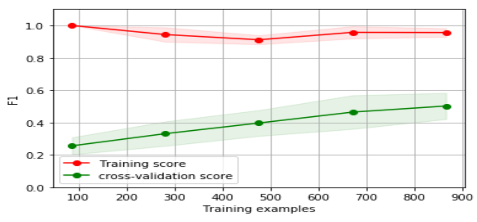
(a) Baseline Classifier



(b) DD Classifier



(c) FS Classifier



(d) FE Classifier

Figure 5.12: Learning curves of the Baseline, DD, FS, and FE classifiers under 10-Fold Cross-Validation.

Experimental Results

Classification Performance. The average performance of the four classifiers, as shown in Table 5.9, is very close, with the Baseline and FS classifiers achieving the same means. In comparison, the FE classifier has the overall best performance, followed by the DD classifier, and the Baseline and FS classifiers in joint third place. These results suggest that, with reference to the Baseline classifier, using feature extension or dataset decomposition has marginally improved the classification performance, but using feature selection has neither improved nor worsened the performance.

Specifically, the FE classifier is the most consistent and robust classifier among the four as it performs well on all but the bottom three categories; in particular, it outperforms the other three classifiers by a great amount on the Look & feel, Maintainability, and Scalability categories—a clear indication of the benefit of feature extension. The performance of the DD classifier is also encouraging. Like the FE classifier, the DD classifier has also made improvements over the Baseline classifier on many minor classes (i.e., Look & feel, Availability, Maintainability, Scalability, and Legal). This clearly shows that the dataset decomposition has a positive effect on the classification of (most) minor classes.

The poorer performance of the FS classifier, in comparison with that of the DD and FE classifiers, suggests that our feature selection technique is unable to correctly identify the distinct features that differentiate the types of requirements. There are many reasons for this failure, including feature sparsity caused by the short text problem in requirements text and inadequate semantic roles captured in our feature selection technique.

A detailed analysis of the classification results for the individual requirement categories brought the same observation reported in Section 5.2.3: the Operational category is the only major class that is poorly performed by all four classifiers, whereas Availability is the only minor class that is well performed by all. As previously noted, intuitive and distinctive features of the Availability category, such as "available", and "24/7", help the classifiers detect the requirements in this category despite the small number of examples in it. By contrast, the features of Operational are not intuitive or distinctive as they also occur in other categories. For example, the features that are frequently used in Operational requirements include "interface" and "server", and these are not unique to the Operational category as they are also frequently used in other categories, such as Look & feel, Functional, and Security. Such indistinguishable features can misguide the classifiers.

Another reason for the classifiers to perform poorly on the Operational category is due to lack of clear distinction between the Operational and Portability categories. For example, the requirement "The system shall be compatible with the Microsoft Windows Operating System" is labeled Portability in the PROMISE-exp dataset, whereas the requirement "The system shall operate on Unix and Windows operating systems" is labeled Operational. This problem makes it difficult to train a classifier that can always correctly classify the requirements in these categories. Perhaps due to this problem, the Baseline and FE classifiers have completely misclassified the requirements in Portability, as their performance metrics all equal zero. It can also be observed that both the DD and FS classifiers also perform very poorly on the Portability category. Since Portability is a minor class (the smallest one in our dataset), and Operational is a major class, test results in Table 5.9 clearly indicate that all four classifiers are biased towards the major class, with the Baseline and FE classifiers completely misclassifying the requirements in the minor class, whereas the DD and FS classifiers only achieve 5-10% of classification accuracy on the minor class.

The problem just demonstrated that a combination of class imbalances and *class ambiguity* poses a serious threat to the performance of a classifier. In RE, class ambiguity is referred to as the representation problem [Gli07]; our classification of a requirement depends on the way we represent it (see Section 2.1.2). This problem prevails particularly in the requirements concerning a system's security, as they can be classified into the Functional or Security category, depending on how we represent them [Gli07].

Finally, in terms of execution time, the DD classifier is the fastest as it took 1.3 minutes to run, followed by the FS classifier which took 2.5 minutes. The Baseline classifier took 3.34 minutes, and the FE classifier took 3.5 minutes. The execution time is based on the time spent on 10-fold cross validation without considering the time used for tuning the parameter.

Learning Curves. As shown in Figure 5.12, the training curves of all the classifiers start very high ($F1 = 1.0$); however, as the training progresses, the training curve of the Baseline classifier stays largely unchanged, while the other three training curves slightly decline to $F1 \leq 0.95$. The testing curves of the four classifiers move upwards gradually with more training examples; however, the testing curve of the Baseline classifier appears to stagnate when the number of training examples reaches 700, whereas the testing curves of the DD, FS, and FE classifiers continue to climb up, exhibiting an upward trend. This seems to suggest that these three classifiers are generalizing well and their performance may continue to improve if they are fed with more and more training examples.

5.4.2 Comparison with Other Techniques

In this section, we evaluate the performance of ML4RC techniques against other related techniques which have been used to address the learning problems in requirement classification. Due to the lack of techniques used in addressing the short text classification problem in requirement classification, our comparison only includes techniques that are handling imbalance class and high-dimensionality problems. This has been done through the following two experiments:

- *Experiment 1* assesses the effectiveness of the DD technique by comparing the implementation of a classifier with the DD technique versus two classifiers implementing over-sampling and under-sampling techniques separately.

- *Experiment 2* assesses the effectiveness of semantic role-based FS by implementing a classifier with our FS technique versus implementing a classifier with a widely-used FS technique (i.e., information gain).

The subsections below are organized based on the experiments. Each section discusses a single experiment by showing its implementation description and results obtained.

Evaluating The Dataset Decomposition Technique

In this experiment, we evaluate the dataset decomposition technique of ML4RC against under-sampling and over-sampling techniques. These resampling techniques are well-known solutions for handling the imbalanced classification problem (as discussed in Chapter 4) and have been used for this purpose in NFR classification [KM17, HKKT20a]. Therefore, they are used to evaluate the effectiveness of the data decomposition technique of ML4RC in this experiment. The following classifiers are implemented to facilitate our evaluation:

1. *The DD Classifier*: This classifier only includes the dataset decomposition technique in its implementation.
2. *The OS Classifier*: This classifier only includes over-sampling technique in its implementation.
3. *The US Classifier*: This classifier only includes the under-sampling technique in its implementation.

Experiment Execution

The DD classifier was implemented using the same setup as described in Section 5.4.1 with no feature selection or extension. The OS and US classifiers were implemented similarly to the steps described in section 5.4.1. However, we applied over-sampling and under-sampling techniques separately after the feature representation step. The *imblearn* [10] package was used to over-sample the minority classes by randomly replicating the minority class examples (over-sampling), and under-sample the majority classes by randomly discarding samples from the majority classes (under-sampling). The three classifiers were implemented individually, each trained and tested using the 10-fold cross-validation process. The results of this experiment are presented in table 5.10.

Experimental Results

Classification Performance. Table 5.10 shows that average performance of DD is similar to the OS classifier, the same F1-score ($F1 = 0.48$), higher Recall ($R = 0.51$), but low Precision ($P = 0.50$). However, the DD classifier clearly outperforms the US with a difference of 15% in Precision, 7% in Recall and 15% in F1-score.

For individual requirements, generally speaking, the DD classifier and the OS classifier show similar performance; high performance for the majority classes (except

[10]https://imbalanced-learn.readthedocs.io/en/stable/api.html

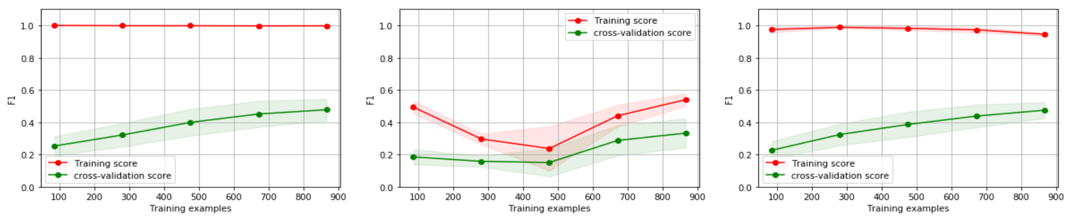
Category	OS Classifier			US Classifier			DD Classifier		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
F	0.71* ±0.04	0.79 ±0.15	0.74 ±0.07	0.80 ±0.21	0.33* ±0.20	0.44* ±0.21	0.79 ±0.05	0.74 ±0.17	0.75 ±0.09
SE	0.61* ±0.19	0.63 ±0.15	0.59* ±0.12	0.52* ±0.26	0.26* ±0.15	0.34* ±0.34	0.74 ±0.15	0.63 ±0.16	0.66 ±0.09
US	0.56 ±0.24	0.51 ±0.25	0.52 ±0.23	0.25* ±0.11	0.44* ±0.21	0.30* ±0.13	0.52 ±0.10	0.62 ±0.22	0.55 ±0.12
O	0.39 ±0.13	0.43 ±0.27	0.39 ±0.18	0.20* ±0.16	0.29* ±0.15	0.22* ±0.15	0.36 ±0.15	0.55 ±0.24	0.43 ±0.16
PE	0.90* ±0.15	0.72 ±0.13	0.78 ±0.10	0.67 ±0.26	0.53* ±0.22	0.55 ±0.22	0.67 ±0.19	0.69 ±0.13	0.66 ±0.14
LF	0.72* ±0.32	0.45 ±0.20	0.51 ±0.20	0.34 ±0.30	0.27* ±0.19	0.27 * ±0.19	0.51 ±0.23	0.57 ±0.30	0.49 ±0.21
A	0.72 ±0.39	0.67 ±0.37	0.68 ±0.36	0.30* ±0.21	0.63 ±0.41	0.40* ±0.27	0.72 ±0.38	0.70 ±0.38	0.70 ±0.36
MN	0.57 ±0.42	0.40 ±0.35	0.44 ±0.33	0.15* ±0.14	0.47 ±0.45	0.22 ±0.21	0.44 ±0.42	0.37 ±0.37	0.37 ±0.35
SC	0.45 ±0.42	0.40 ±0.38	0.40 ±0.36	0.37 ±0.34	0.67* ±0.39	0.41* ±0.29	0.55 ±0.45	0.45 ±0.42	0.49 ±0.43
FT	0.35 ±0.45	0.25 ±0.34	0.27 ±0.33	0.14 ±0.12	0.35 ±0.32	0.18 ±0.16	0.25 ±0.34	0.25 ±0.34	0.23 ±0.29
L	0.45 ±0.47	0.40 ±0.44	0.40 ±0.42	0.35 ±0.39	0.55 ±0.47	0.38 ±0.40	0.45 ±0.47	0.40 ±0.44	0.40 ±0.42
PO	0.03 ±0.10	0.10 ±0.30	0.05 ±0.15	0.11 ±0.13	0.50* ±0.50	0.18 ±0.20	0.05 ±0.15	0.10 ±0.30	0.07 ±0.20
Mic.	**0.64** ±0.06	**0.64** ±0.06	**0.64** ±0.06	**0.36** ±0.11	**0.36** ±0.11	**0.36** ±0.11	**0.66** ±0.05	**0.66** ±0.05	**0.66** ±0.05
Mac.	**0.54** ±0.05	**0.48** ±0.05	**0.48** ±0.05	**0.35** ±0.12	**0.44** ±0.11	**0.33** ±0.13	**0.50** ±0.05	**0.51** ±0.07	**0.48** ±0.06
Ex. Time	14 minutes			5 seconds			1.3 minutes		

* Indicates a significant statistical difference between the baseline and another classifier with p<0.05. Wilcoxon signed-rank test has been applied.

Table 5.10: 10-Fold Cross-Validated test results for the classifiers handling the class imbalance problem using over-sampling (OS), under-sampling (US) and Dataset Decomposition (DD) techniques.

Operability and Look & feel), and low in minority classes (except Availability). In particular, DD outperforms OS in the majority of classes except four: Performance, Look & feel, Maintainability, and Fault tolerance. In addition, the DD classifier outperforms the US classifier in all the classes, except for the most minority class (portability).

Statistically, the Wilcoxon signed-rank test shows that DD has made a significant improvement over on Functional (precision) and Security (precision and F1-score), while the OS classifier significantly improved compared to the DD classifier on Performance (Precision), Look & feel (Precision). In comparison with the US classifier, the DD classifier has made a significant improvement on at least 10 classes, while the US classifier significantly outperformed the DD classifier on Portability (Recall).



(a) OS Classifier (b) US Classifier (c) DD Classifier

Figure 5.13: Learning Curves of the over-sampling (OS), under-sampling (US) and dataset decomposition (DD) Classifiers Based on 10-Fold Cross-Validation.

In terms of execution time, the US classifier is the fastest with 5 seconds, followed by the DD classifier with 1.3 minutes. Finally, the OS classifier took the longest time with 13 minutes.

Learning Curves. It can be observed that the training curves of the OS and DD classifiers start very high ($F1 \approx 1.0$); as the training progresses, the training curve of DD declines slightly ($F1 = 0.95$), whereas the training curves of OS stay largely unchanged, resulting in a bigger gap between training and cross-validation curves (i.e., a high variance) which leads to overfitting. DD also has a big gap between training and testing curves; however, its testing is more prominently climbing upwards, suggesting that the DD classifier has more potential for better generalization if the size of the training data continues to increase. The learning curve of the US classifier, on the other hand, indicates that US suffers from the underfitting problem, low training, and testing scores.

Evaluating the Feature Selection Technique

In this experiment, we evaluate the feature selection technique of ML4RC against information gain technique. Information gain is a widely used features selection technique in text classification tasks that shows good classifier performance and has outperformed other features' selection technique in comparative experimental studies [For03, YP97]. It also frequently used in requirement classification tasks (as shown in Chapter 4); therefore, it has been chosen as the basis for this evaluation. We implement two classifiers to facilitate our evaluation:

1. *The FS Classifier*: This classifier only includes the semantic role-based feature selection technique in its implementation.

2. *The Info-gain Classifier*: This classifier only includes the information gain feature selection technique in its implementation.

Cat.	Info-Gain Classifier			FS Classifier		
	P	*R*	*F*1	*P*	*R*	*F*1
F	0.59*±0.04	0.80*±0.10	0.68*±0.06	0.68 ±0.03	0.85 ±0.01	0.76 ±0.05
SE	0.67 ±0.17	0.54 ±0.18	0.58 ±0.15	0.64 ±0.19	0.60 ±0.17	0.60 ±0.16
US	0.57 ±0.30	0.36*±0.21	0.43*±0.24	0.59 ±0.18	0.49 ±0.20	0.53 ±0.18
O	0.35 ±0.17	0.22*±0.16	0.25*±0.15	0.46 ±0.14	0.42 ±0.21	0.42 ±0.15
PE	0.83 ±0.15	0.68 ±0.21	0.73 ±0.16	0.87 ±0.18	0.69 ±0.16	0.76 ±0.15
LF	0.66 ±0.42	0.30 ±0.21	0.36 ±0.25	0.60 ±0.39	0.33 ±0.25	0.39 ±0.26
A	0.31*±0.34	0.25*±0.27	0.27*±0.29	0.76 ±0.40	0.63 ±0.38	0.66 ±0.36
MN	0.38 ±0.43	0.20 ±0.21	0.25 ±0.26	0.35 ±0.39	0.23 ±0.24	0.27 ±0.28
SC	0.62 ±0.43	0.50 ±0.39	0.53 ±0.38	0.57 ±0.47	0.48 ±0.42	0.51 ±0.44
FT	0.55*±0.47	0.40*±0.37	0.45*±0.39	0.25 ±0.40	0.20 ±0.33	0.22 ±0.35
L	0.43 ±0.47	0.45 ±0.45	0.42 ±0.44	0.47 ±0.46	0.45 ±0.42	0.43 ±0.40
PO	0.03 ±0.10	0.10 ±0.30	0.05 ±0.15	0.10 ±0.30	0.10 ±0.30	0.10 ±0.30
Mic.	**0.59** ±0.15	**0.59** ±0.15	**0.59** ±0.15	**0.65** ±0.05	**0.65**±0.05	**0.65**±0.05
Mac.	**0.50** ±0.04	**0.40**±0.05	**0.42**±0.04	**0.53** ±0.06	**0.46**±0.05	**0.47**±0.05
Ex. Time	10.2 minutes			2.5 minutes		

* Indicates a significant statistical difference between the baseline and another classifier with p<0.05. Wilcoxon signed-rank test has been applied.

Table 5.11: 10-Fold Cross-Validated test results for the two classifiers applying different feature selection methods

Experiment Execution FS was implemented using the same setup as described in Section 5.4.1. The Info-gain classifier was implemented similarly to FS classifier; however, the features were selected based on their information gain value. We computed an information gain value for term t based on the definition provided by Yang and Pedersen [YP97] as follows:

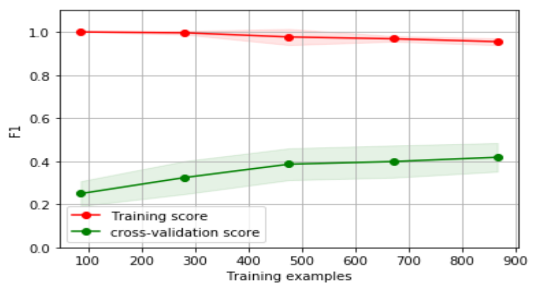
$$\begin{aligned} IG(D,t) = -\textstyle\sum_{i=1}^{k} P(C_i)logP(C_i) + [p(t)\textstyle\sum_{i=1}^{k} P(C_i,t)logP(C_i,t) \\ +P(\bar{t})\textstyle\sum_{i=1}^{k} P(C_i,\bar{t})LogP(C_i,\bar{t})] \end{aligned} \tag{5.2}$$

where k is the number of classes in the dataset (D), and t is the appearance of term t, while $\bar{t}$ is the absence of term t. Then, the terms (features) are ranked in descending order based on information gain value, and top X out of 2129 features were selected to represent the NFRs. X is the number of features that are selected by role-based technique.

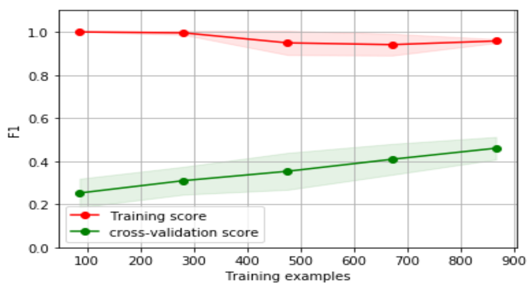
The two classifiers were implemented individually, each trained and tested using the 10-fold cross validation process. The results of this experiment are presented in table 5.11.

Experimental Results

Classification Performance. Table 5.11 shows that the average performance of the



(a) Info-gain Classifier



(b) FS Classifier

Figure 5.14: Learning Curves of the over-sampling (OS), under-sampling (US) and dataset decomposition (DD) Classifiers Based on 10-Fold Cross-Validation.

FS classifier is higher than that achieved by applying the info-gain classifier with a precision difference of 3%, recall 6%, and F1-score 5%. For individual requirements, apart from Fault tolerance and Scalability, the FS classifier outperforms the info-gain classifier in all the categories. Statistically, the Wilcoxon signed-rank test shows that the FS classifier has made a statistically significant improvement in Functional, Usability, Operational, and Availability. In contrast, the info-gain classifier has made a significant improvement in Fault tolerance.

In terms of execution time, FS is faster than the info-gain classifier. Information gain is a supervised feature selection method (i.e., using requirement types). It thus requires computing an information gain value for each term in each iteration. However, FS is an unsupervised feature selection method that can implement one time for all the iterations.

Learning Curves. As shown in Figure 5.10, the training curves of the classifiers start very high ($F1 = 1.0$); however, as the training progresses, the training curve slightly declines to $F1 \leq 0.95$. On the other hand, the testing curves of the two classifiers start low ($F1 = 0.25$) and move upwards gradually with more training examples. The FS testing curve more prominently climbs upwards and ends with a higher score than the info-gain classifier. This suggests that FS classifiers are generalizing slightly better than info-gain. Besides, the FS classifier has more potential to improve by adding more examples as its testing curve moves upward more than info-gain, which is slowly moved upward when the number of examples =500.

5.5 Discussion

Considering the experimental results reported in Sections 5.2-5.4, here we discuss the strengths and weaknesses of the proposed techniques of ML4RC. Then, we discuss some research challenges experienced during the development of the ML4RC method.

5.5.1 On Class Imbalances

As shown in Chapter 4, class imbalance problem is commonly address in requirements classification using data level methods (e.g., over-sampling and under-sampling strategies). Methods belonging to the data level are simple and flexible for any algorithm; however, they either increase the risk of overfilling (e.g., over-sampling) or discarding useful data (under-sampling). Moreover, such solutions are less effective and even cause a negative effect in dealing with multi-class classification tasks [WY12]. Moreover, in ML4RC, we proposed a multi-class imbalanced technique that divides the training dataset into two subsets to reduce class imbalances within each subset. Then, a hierarchical classification, rather than ensemble-based learning, is employed for its simplicity and ease in implementation.

Our empirical validation of this technique in Section 5.4 (see Table 5.9) shows that, in comparison with the Baseline classifier, the classifier based on dataset decomposition (DD) performs better in terms of classification results as well as execution time. Moreover, the validation of the dataset decomposition technique with some related techniques in Section 5.4.2 (see Table 5.10) shows that DD outperforms the classification results of the under-sampling classifier. In comparison with the oversampling technique, although DD shows comparable performance to oversampling, it takes less execution time and suffers less from the overfitting problem. We deduce that the better performance of the DD classifier can be attributed to the benefit of dataset decomposition, as decomposition reduces the feature space and simplifies the classification task, as also noted by other authors [MR05, GIS10]

Through experimental evaluation of the entire ML4RC method against four closely related methods (Section 5.3), we show that ML4RC performs better than these methods (Table 5.8). Significantly, two of these methods also deal with the imbalanced data problem; in particular, Kurtanović and Maalej's method [KM17] uses the over-sampling method whereas Yin et al.'s method [YGX$^+$13] employs class decomposition. However, over-sampling can cause *overfitting* in the training data [Die95], making the classification model more specific to the training data and therefore less generalizable to the unseen or new data. Class decomposition is also problematic as it leads to many small classes. Furthermore, specifically in Yin et al.'s class decomposition method, it uses K-means to cluster the major classes which works well when there are distinctive features between different classes. As NFR classes are not well-defined, K-means clustering cannot clearly differentiate between different classes [AKG$^+$17].

However, dataset decomposition is not perfect either, as the decomposition made in the first level can affect classification performance at the second level. In particular, since classes are grouped into subsets based on their distribution, their classification can be misled by the dominating (major) classes in subsets. To address this problem, Zimek et al. [ZBFK08] suggested decomposing the dataset based on class similarity and not class distribution. We intend to investigate this approach in our future work.

5.5.2 On Feature Selection

The high dimensionality problem in text classification is commonly solved using filter-based methods [CHTQ09]. As shown in Chapter 4, filter-based methods are mainly divided into two main categories: statistical-based techniques and linguistic-based techniques. Whereas statistical-based methods are more efficient, it is less effective with small datasets. Therefore, in this ML4RC, we propose a semantic role-based feature selection technique which identifies the relevant features from each requirement statement according to six semantic roles. This technique is simple and easy to use, and can be implemented using existing NLP tools. It can also be applied to other application domains, as the semantic roles are generic semantic concepts and not specific to requirements text.

Our empirical validation of this technique in Section 5.4 (see Table 5.9) illustrates that, in comparison with the Baseline classifier, the classifier based on feature selection (FS) performs better in terms of execution time; it is also less prone to overfitting. However, it does not outperform the classification results obtained from the Baseline classifier. This agrees with the findings of Kurtanović and Maalej [KM17], who showed that the best performance for the best identification of NFRs with SVMs was achieved by using all the features without selecting a part of them; however, this strategy does increase the risk of overfitting.

Our empirical comparison of a semantic role-based feature selection technique with info-gain, that most frequently used feature selection methods in requirement classification, shows that our feature selection methods outperform the info-gain in terms of classification performance and execution time. Information-gain is a method based on statistics, and, as we previously said, ideal for selecting good features. However, most of the available requirement datasets are relatively small (see Section 3.7).

Through experimental evaluation of the entire ML4RC method against related methods (see Section 5.3), we show that ML4RC performs better than other methods that also explicitly address the high dimensionality problem, which are the methods by Cleland-Huang et al. [CHSZS07], Kurtanović and Maalej [KM17] and Yin et al. [YGX$^+$13] (Table 5.8 and 5.7). All of these methods used co-occurrence-based methods. For example, Cleland-Huang et al. used proposed probability-based function to select the features, Kurtanović and Maalej's method [KM17] method uses a scoring function to determine the importance of the top 500 features, and Yin et al.'s method [YGX$^+$13] applies the mutual information measure to determine the relevance of the words. Our evaluation also further confirms that feature selection methods based on statistics cannot perform well in requirements classification due to the short-text requirements and small datasets.

Since our feature selection technique uses existing NLP tools such as the dependency parser and the POS tagger, its performance is bound by the performance of these tools. In particular, if sentences are poorly formed, NLP tools may incorrectly remove some relevant features. For example, the requirement "90 of maintenance software developers are able to integrate new functionality into the product within two working days" should be classified as a Maintenance requirement, but the feature set selected by our feature selection technique is {developer new functionality product working day}, which has removed the most relevant feature "maintenance". Consequently, this requirement has been misclassified as Operational due to the poor grammar of the original requirement. We believe our feature selection technique can be improved by combining semantic roles with RE keywords such as those identified by Cleland-Huang et al. [CHSZS07].

5.5.3 On Feature Extension

As shown in Chapter 4, the common methods applied to handle the short text classification is feature extension, where the features are extracted from an internal or external dataset. Although the use of external sources can handle sparsity more than internal sources, it is time-consuming and prone to noise. The classification of NFRs, in particular, is highly sensitive to noise due to the nature of overlapping between requirements [SRS14]. Therefore, in ML4RC, we propose an internal resource feature extension technique for handling short text problems in NFR classification. This feature extension method has fewer requirements and can also reduce potential noise that might be obtained from the expanded features. Due to the overlapping nature among non-functional categories, we restricted the expanded features to be synonyms that appear in a training dataset to ensure the least amount of noise. WordNet is used for identifying the synonyms.

Our empirical validation of this technique in Section 5.4 (see Table 5.9) shows that, in comparison with the Baseline classifier, the classifier based on feature extension (FE) performs better in terms of classification results. We posit that the better performance of the FE classifier can be attributed to our semantic role-based feature selection as it helps identify related semantic features (via semantic roles they play). Based on these related semantic features, our feature extension can identify additional semantic features that are also related. This helps the ML algorithm to learn the associations between them and thus improves classification performance. This observation is similar to the finding reported in [Man14], which indicates that the performance of SVM in short text classification can be improved by expanding the text with related terms. The main limitation of our feature extension technique is that the expanded words are restricted to only those appearing in the training dataset, which are inadequate.

As shown in Section 5.3, Lu and Liang [LL17] also address the short-text problem. Their technique differs from ours in that they use word embedding to measure the similarity of the words and add similar words as additional features. Since a large number of potentially similar words can be identified using this measure, Lu and Liang use term frequency of similar words to rank their relevance to the requirements. This leads to the identification of a few similar words. Our experimental validation in Section 5.3 (see also Table 5.8) shows that Lu and Liang's overall method ML4RC performs worse than ML4RC. Part of the reason for this is that their feature extension technique fails to identify many relevant words, due to the restriction of their similarity measure.

5.5.4 On Evaluation Practices and Performance Benchmarks

Although ML4RC showed an improvement over other related methods (see sections 5.3), the results are still unsatisfactory (F1 $\leq$ 50). Based on our understanding of the difficulty of NFRs classification and the limitations of existing datasets, such performance is expected. However. the related works have reported rather high results. For example, Kurtanovic and Maalej [KM17] showed that their method reached F1 > 70 for all classes, and Lu and Liang's method [LL17] also reached an F1 average of 72%, Hey et al.'s [HKKT20a] achieved F1 = 82%.

By examining the related works, we discovered four possible reasons for the noticeable differences in our study's results compared to those reported in other studies. The reasons are explained as follows.

1. The inappropriate use of evaluation metrics. For example, some evaluation metrics used in the others studies were not appropriate assessment measures for imbalanced datasets due to a bias toward the majority class. For example, a weighted average was used in [HKKT20a, ROW13], where each class's performance was weighted (multiplied by) the number of class samples, resulting in high weight for majority classes.

2. The selectivity of the dataset used in the others studies. For example, many studies used only some of the dataset, for example, the four most frequent NFR classes (e.g., [KM17, HKKT20a]). This simplifies the classification task and improves the classification results.

3. The application of a set of separated binary classifiers to perform a multi-class classification (e.g, [DDAÇ19]). The performance of each class is computed separately, and the mean of all classes is reported. Such practice might lead to predicting a requirement differently (multi-label classification), which is possible and correct. However, the dataset applied in this thesis is annotated with single labels.

4. A different implementation of k-fold cross-validation, known as micro-averaging [PA12] or an average of true positives and false positives [FS10]. Instead of separately evaluating each fold, the predicted labels were aggregated to be compared with the true labels after applying a cross-validation method (e.g., as shown in [DCCM20] and the source code [HKKT20b] of Hey et al.[HKKT20a]).

In contrast, the proposed methods use the macro average as the basis for discussion and analysis; the macro average deals with all classes equally. Also, it predicts 12 classes, where one is functional and the others are non-functional. These classes are used, as they are provided in public requirement datasets (i.e., PROMISE or PROMISE-exp).

Table 5.12 shows the differences of F1-score values for Security class and overall performance using PROMISE-exp (see Tables 5.3 and 5.2) with different evaluation approaches. The approach applied in this thesis is multi-class (12 classes) classification with macro-average cross-validation.

Evaluation Approach		Security F1-score	Overall (Macro avg.)	Overall (weighted avg.)
Multi-class (12 classes)	F1-score -average CV	0.64	0.47	0.62
	TP& FP Average CV	0.62	0.51	0.64
Multi-class (4 classes)	F1-score Average CV	0.75	0.70	0.71
	TP& FP Average CV	0.75	0.71	0.71
Binary SE vs. other NFRs	F1-score Average	0.76	0.84	0.89
	TP& FP Average CV	0.77	0.85	0.89

Table 5.12: The changes of F-score for Security and overall performance of a baseline classifier, using different evaluation settings and learning types (binary, multi-class with 12 classes, and multi-class with 4 classes). Note: the dataset used is PROMISE-exp.

5.6 Threats to Validity

In this section, we discuss threats to the validity of the results based on the guidelines proposed by Shull et al. [SSS07], as well as our mitigation strategies.

5.6.1 Construct Validity

This refers to the extent to which the theoretical constructs are correctly interpreted and measured (relationship between the theory and the observation). To minimize threats to construct validity, we employed common performance metrics to measure classification effectiveness (recall, precision, and F-score) [Seb02]. Nevertheless, there was a potential threat to construct validity in our implementation of ML classifiers. All experiments in our research were only conducted in Python, so the results might vary when applied with another programming language. This threat is intensified in implementation-related methods, as there might be biases in selecting a language, since some of them were implemented using another language. However, to mitigate this threat, we used the same parameters in all experiments conducted; thus, the results should be valid for other applications.

Another potential threat to construct validity existed in the dataset used to evaluate the ML models: subjective bias and erroneously labeled data.The dataset used in this chapter is PROMISE-exp, a public dataset that was already classified and validated by its builders. PROMISE-EXP is the expansion of PROMISE, a publicly annotated dataset employed in many related studies (see Table 3.6). PROMISE-exp dataset consists of 47 requirements documents collected from different resources and by different authors, mitigating expected threats (e.g., bias or misunderstanding). Nevertheless, we observed during our research: Some requirements may not be correctly labeled [LHM^{+}14]; the selection of the requirements for the dataset may be biased or mismatch with the real-world requirements; the number of requirements in the dataset is inadequately and too small to represent real software applications, and most of the requirements in the dataset are written by students, so the industrial standards are not necessarily maintained. Unfortunately, these threats exist in all research using external datasets and such threats cannot be easily mitigated.

5.6.2 Internal Validity

Threats to internal validity concern confounding factors that can influence the observed results (without the knowledge of the researchers). To pursue high internal validity, our research design adhered to the standard experimental design guidelines used in software engineering [KPP$^+$02]. A careful analysis was performed to choose the parameters used to build the ML model. However, there is a potential internal validity threat related to ML models. This threat related to the choice of the independent variables used while building the ML model. An example of such a threat was overfitting. To minimize this threat, we used k-fold and p-fold cross-validation and feature selection. In addition, we drew a learning curve for each classifier to observe the overfitting and clearly remark upon this threat. Moreover, we applied ML to another small dataset collected from different sources and annotated externally.

5.6.3 External Validity

This refers to the generalizability of the research and its outcomes in different settings [KPP$^+$02]. External validity is closely related to the replicability or repeatability of research results or observations [Gol03].

An example of such a threat is the lack of generalizability of our findings in other settings. To mitigate this threat, we use a stratified K-fold cross-validation procedure to train and validate all the classifiers used in this research. This is a well-known resampling procedure for evaluating ML models on limited data samples [Bro14]. To further mitigate this threat, we provide a detailed description of this method so that it can be replicated.

Another external validity threat is due to our reconstruction of related methods. To mitigate this threat, we carefully selected the methods that have relatively clear descriptions and followed the descriptions faithfully; we clearly state any changes made to these methods and the rationale behind the changes.

5.7 Summary

In this chapter, we present a NFR classification method called ML4RC. This method consists of three techniques and each technique addresses a single problem when using supervised ML for NFR classification. For example, the dataset decomposition technique is used to address the class imbalance problem. This technique decomposes the classification problem into sub-classification problems with less imbalance level among the classes. A semantic role-based feature selection technique selects the key roles (features) from a requirement to address the high-dimensionality problem. These roles are then selected to be fed to ML classifier, while other features are discards. To address the shortness in requirements, the feature extension method is used. This method extends each feature with synonyms extracted from WordNet.

The method was applied to classify NFRs over PROMISE-exp dataset which consists of 969 requirements, 525 of which are NFRs. It was tested separately over four supervised learning classifiers that are frequently used in RE and compared with the other existing NFRs classification approaches. The results show that ML4RC had the highest performance with SVM and outperformed most of the existing approaches used in the comparison.

Each technique of ML4RC was evaluated separately and compared with a baseline classifier that does not apply any technique of ML4RC. These techniques are also compared with similar techniques that are proposed to address the same problem. The results show that our feature selection cannot improve the ML performance of the basic classifier although it outperforms a similar existing technique. In addition, the results highlight the advantages of the decomposition method in requirement classification: less overfitting, low time consumption, and a slight improvement in the classification performance. One of the the valuable findings is that feature extension shows the highest performance when compared to other techniques and methods, indicating the importance of the short length problem in requirement classification. Based on this finding, we will conduct further investigations regarding extending features in requirement classification by developing a new method known as SE4RC (Semantic Extension for ML classification). This method will be presented and evaluated in the following chapter.

Chapter 6

Semantic Expansion For Short-text Requirements Classification

"It is better to solve one problem five different ways, than to solve five problems one way."

George Pólya

The previous chapter's findings provide insight into the effectiveness of addressing the short text problem in NFRs classification. Motivated by these findings, we propose a method called "SE4RC" (Semantic Expansion for Requirement Classification) to address the short text classification problem in a novel way. This method simulates normal length text classification by expanding short requirements with semantically similar requirements extracted from a training dataset. Thus, the main difference between SE4RC and the feature extension technique in ML4RC is that SE4RC expands short requirements with similar requirements while ML4RC uses similar features (words). In addition, SE4RC uses a corpus-based method (i.e., word embedding) to measure the similarity of the requirements, while ML4RC uses a knowledge-based method (i.e.WordNet).

To assess the effectiveness of SE4RC, we conducted three experimental studies. The first experiment compares the similarity measure of SE4RC with other short text similarity methods in measuring requirements similarity. The second experiment evaluates the performance of SE4RC in relation to a baseline method, and the third compares SE4RC with two related methods.

This chapter is organized as follows: Section 6.1 provides an overview requirement similarity. Section 6.2 introduces SE4RC method. Section 6.3 presents the experiment carried out to compare the similarity measure of SE4RC with other text similarity methods. Section 6.4 evaluates the performance of ER4RE, while Section 6.5 compares the performance of SE4RC against related work. Based on the experimental results, Section 6.6 discusses the key findings together with the opportunities and challenges for requirements classification, followed by Section 6.7 which discusses limitations and threats to validity. Finally, Section 6.8 provides a summary of this chapter.

6.1 Overview of Text Similarity Approaches

Before reviewing text similarity approaches, it important to provide a clear definition of similarity and how it differs from relatedness. When it is said that two items are similar, it means that they can be substituted in a given context without changing the underlying semantics (e.g., “users” and “person”) [NM19]. The relatedness, on the other hand, refers to items that are correlated but not substituted. Semantic similarity is considered a special case of semantic relatedness that contains more semantic relationships between concepts such as is–an–attribute–of, has–part, is made–of, and is–the–opposite [PBP03, NM19]. Examples of semantically related concepts are email and password, person and address, or success and failure. As these relationships could result in unrelated words (i.e. noise for ML classifier), we especially focus on semantic similarity measures in this chapter.

This section provides an overview of the similarity methods that have been used to measure short-text similarity in general and requirements similarity in particular. In addition, it introduces a well-known word embedding model (i.e., Word2vec) which is used in SE4RC to measure the similarity between requirements.

6.1.1 Approaches to Short-Text Similarity Measures

Many similarity measures (i.e. methods) have been proposed to determine the semantic relation between texts. The obvious solution is using the traditional measures which are mainly focused on comparing word co-occurrences in the texts (e.g. lexical overlapping) [QLL^{+}10]. Such techniques can achieve good results in long texts as long texts are likely to share common words; however, this is not the case with short texts. For instance, there is an obvious similarity between the two requirements: “the product shall be free of computer viruses,” and “the system must prevent malicious attacks including the denial of service," but most co-occurrence–based measures will fail to identify any kind of connection between these texts [OSDCI11].

The other approaches of measuring text similarity are based on word-to-word similarity approaches such as corpus-based and knowledge-based approaches [MCS$^+$06]. A knowledge-based approach uses an external lexical recourse to measure the similarity between words (WordNet is a popular English recourse). The main benefits of this approach are that it is reliable and less complex as it is based on expert human judgments. Its drawbacks are its dependence on convergence and its reliance on the quality of the thesaurus used (e.g. some technical terms tend to be underrepresented by WordNet [JM00]). In addition, most of the similarity methods belong to this approach are limited to specific classes of words, only comparing words with others of the same type (e.g., only comparing nouns- nouns and verbs-verbs) [MCS$^+$06].

The measures that belong to the knowledge-based approach can be classified as path-based measures and information content measures. Path-based measures estimate the semantic similarity of two concepts based on the short path connecting them in the WordNet hierarchy [MHG13]. Common examples of path-based measures are Leacock & Chodorow [LC98], Wu & Palmer [WP94], and Path Length. Content-based measures, on the other hand, use the information content of each concept included in WordNet to measure the similarity between concepts; the more common the information concept share, the more similar the concept is [MCS$^+$06, II09]. Common examples of content-based measures are Resnik [Res95], Lin [L$^+$98], and Jiang & Conrath [JC97].

A corpus-based approach measures the semantic similarity between two words according to information derived from a large corpus (i.e, large collections of raw textual data). This approach exploits the statistical information hidden in the corpus to compute the similarity between terms based on the assumption that the semantic properties of a word can be inferred by its context [NM19, LWZ$^+$15]. In other words, terms that have similar co-occurrence behavior in a corpus tend to be semantically similar. The main benefits of the corpus-based approach are that no prior knowledge is needed and that it offers the ability to compare numerous units of language (e.g. to compare texts to words) [JM00]. The limitation of this approach is that the words being compared must occur at least a few times, since these methods measure the similarity of words based on the context of those words [JM00].

The methods that belong to corpus-based approach can be classified into two categories: co-occurrence based methods (i.e., association and probabilistic measures) and distributional representations methods. Co-occurrence methods quantify the semantic similarity between words based on the occurrence in the corpus. Common examples of co-occurrence methods are Pointwise Mutual Information (PMI), Jaccard Index, and Dice's coefficient [NM19, MW16]. Distributional representations methods encode the behavioral use of specific words into vector representation that can either be interpreted directly (explicit representation) or cannot be interpreted (implicit or latent representation) [NM19]. The most common examples of these methods are latent semantic analysis [MCS+06, II09], and Word embeddings (e.g, GloVe [PSM14] and Word2vec [MSC+13]) [NM19].

Although most of the measures mentioned above are used to measure word-word similarity [MCS+06], several methods have been proposed to use these measures in determining short text similarity. These methods can be classified as word-based similarity and representation-based similarity. The first group used the similarity between words to measure text similarity. A well-known example is a semantic metric proposed by Mihalcea et al. [MCS+06], which combines word-to-word similarity metrics and word specificity to indicate the semantic similarity of the two input texts. The second group convert the sentence into a (single) text representation (e.g., by averaging the word vectors) and compute the similarity between sentences by using a metric for measuring distance, such as cosine [KBdR16].

6.1.2 Approaches to Requirements Similarity Measure

Measuring similarity between texts is commonly used for supporting requirement engineering tasks [FDE+17]. For example, Li and Clel and-Huang[LCH13], Lucia et al. [DLDPO10], Marcus and Maletic [MM03], and Hayes et al. [HDS06] used semantic similarity measures to generate trace links between two software artifacts (i.e., tractability). Younas et al. [YJGS20] and Mahmoud and Williams [MW16] discover and classify NFR by measuring semantic similarity between two textual items. Matsuoka and Lepage [ML11] used a semantic similarity measure to detect ambiguity in software requirements specifications. Och et al. [oDRC+01] measured the similarity between requirements to avoid duplicate identification of requirements. Ilyas and Kung [IK09] measured similarity between requirements to support design and code reusability.

None of the semantic similarity measures are primarily used for expanding requirement with additional term(s). Moreover, the quality of the semantic similarity measure used by most of these studies were evaluated regarding the whole task (e.g., traceability and classification). Very few studies measure the effectiveness of semantic similarity measures between requirements. An example of these studies is that conducted by Alhoshan et al. [ABNZ19] who used Word2Vec to detect similarities between the requirements based on semantic frames. Although their approach showed promising results in comparison to the baseline method (averaging word embedding of requirement words with cosine metric), it is labor-intensive because it entails annotating requirements manually based on each frame's elements.

6.1.3 Word Embedding to Measure Text Similarity

Word embedding is a word vector representation that captures both the semantic and syntactic regularities of words from an unlabeled large corpus. The assumption used in word embedding is that words occurring in a similar context have similar meanings. Thus, word embedding methods use a large corpus to generate semantic space (also known as word embedding space), where each word is presented in a dimensional vector of real numbers. In the semantic space, the semantically related words, such as "man" and "moment", are represented close to each other. Besides, it is possible to compute arithmetic expressions in semantic vectors to detect a relationship between words such as “King” - “Man” + “Woman”, which results in a vector very close to “Queen" (known as word analogies) [MYZ13]. Example of word embedding methods are GloVe [PSM14] (co-occurrence based model) and Word2vec [MSC^{+}13] (neural network based model).

Word2vec, introduced by Mikolov [MSC^{+}13], is the most popular method to generate words embedding due to its efficiency and simplicity [LDBT15]. It is built on a two-layer neural network which learns the vector-based representation of each word from an unlabeled text dataset [NM19]. Two variants of word2vec have been proposed: a continuous bag-of-words (CBOW) model and the skip-gram model (SGM). The CBOW model predicts the central word given its context. For example, if the inputs of the model are the words "username", "login" and "enter", the output could be "passwords". The SGM predicts the context given a central word. For example, if the input word is "login," the outputs could be "password" and "username".

Like any word embedding model, only an unlabeled corpus is needed to train a word2vec model for generating a representation for any word (i.e., generating word embedding). The size and domain relevance of the corpus used to train word embedding are important to ensure the quality of the generated word embeddings [LLHZ16]. Mikolov et al. [MCCD13] released a publicly available Word2vec model that is trained

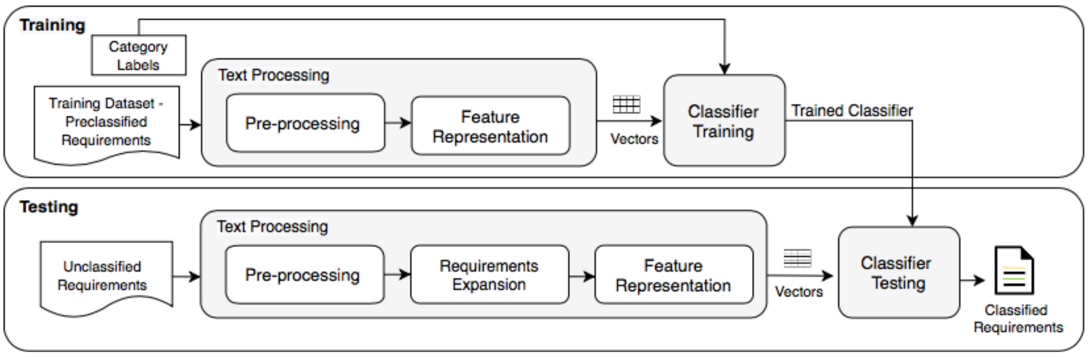


Figure 6.1: Overview of SE4RC-Semantic Expansion For Supervised Requirement Classification

on the Google News data set with approximately 100 billion words [1]. Efstathiou et al. [ECS18] released a pre-trained word-embedding model for the software engineering domain. This model was trained by 15 GB of textual data from Stack Overflow posts.

6.2 The SE4RC Method

The SE4RC method is purposely designed to deal with the short text classification problem by expanding requirements with most semantic similar requirements. Figure 6.1 shows the process of the SE4RC method. This process resembles the text classification process shown in Figure 2.4 with a few changes. These are: no feature selection step in both training and testing phases, and an additional requirements expansion step in the testing phase. The process steps of the SE4RC method and their enabling techniques are detailed in the sections below.

6.2.1 Text Pre-Processing

This step applies for the both training and testing phases. It involves the application of several NLP techniques: tokenization, removal of stop words, lowercase transformation, lemmatization, and short-words removal (i.e., removing the words containing fewer than three characters). The output of this step is sentences consisting of tokens which could be numbers or lemma forms of words.

[1]https://code.google.com/archive/p/word2vec/

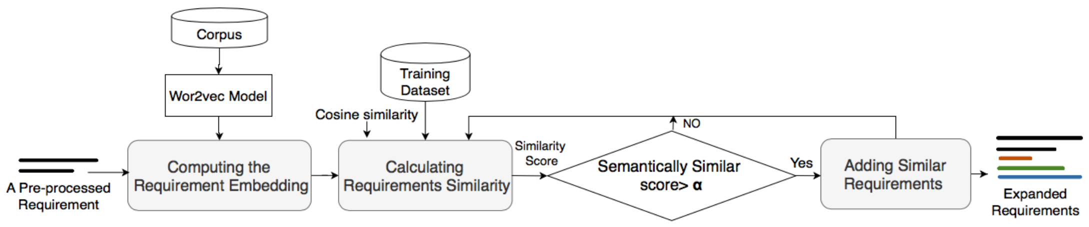

Corpus
Wor2vec Model
Training Dataset
Cosine similarity
A Pre-processed Requirement
Computing the Requirement Embedding
Calculating Requirements Similarity
Similarity Score
Semantically Similar score> α
NO
Yes
Adding Similar Requirements
Expanded Requirements

Figure 6.2: Requirements expansion step

6.2.2 Requirements Expansion

This step, which only applies to the testing phase, aims to expand each requirement in the testing dataset with the most similar requirements in a training dataset through three main sub-steps (as shown in Figure 6.2). Firstly, we compute requirement embedding for each preprocessed requirement. We use weighted averages, generated by a word2vec model [MSC+13], for all word embeddings to compute requirement embedding. Each word embedding is weighted by the inverse document frequency (IDF) [Jon72], which is a measure of the specificity of a word. IDF gives importance to specific words (e.g. reliable and usable) over generic words (e.g. system and product) [MCS+06]. IDF is calculated as an algorithm of the total number of documents in the corpus (i.e., a training dataset) divided by the total number of documents including a specific word. The use of IDF was derived from the work of Mihalcea et al. [MCS+06]. By doing so, the representation of each requirement (R) is calculated as follows:

$$R - embedding = \frac{\sum_{w \in R} V_w \times idf(w)}{\sum_{w \in R} idf(w)} \tag{6.1}$$

In this calculation, (V_w) represented the embedding of word (w) generated by word2vec (i.e., the representation of w in a word2vec semantic space).

In the second sub-step, we use the cosine of the angle between two embedding requirements (an original requirement from a testing data set and a requirement in a training dataset) to measure the similarity between them:

$$\cos(R_1, R_2) = \frac{V_{R1} \times V_{R2}}{||V_{R1}|| \times ||V_{R2}||} \tag{6.2}$$

In the third sub-step, we compare the semantic similarity score with a pre-defined threshold ($0 < \alpha < 1$); if the score is greater than the threshold, we add the requirement extracted from a training dataset at the end of the original requirement. The output of this step is that the requirements are expanded by adding the most similar requirements,

NO	Before		After	
	Pre-processed Requirement	**No Words**	**Extended Pre-processed Requirement**	**No Words**
1	product shall support 2000 concurrent user	6 words	product shall support 2000 concurrent user products shall able support 1000 simultaneous user server support maximum 1000 simultaneous users	21 words
2	system shall refresh display every 60 second	7 words	system shall refresh display every 60 second interface user automated system shall maximum response time second product shall respond fast keep up-to-date data display audit report shall have returned within second system shall allow minimum user work time response time shall fast enough maintain flow game	46 words
3	system back end database shall encrypted	6 words	system back end database shall encrypted secure server required ensure confidentiality customer credit card detail	15 words

Table 6.1: Examples of the requirements before and after applying requirement expansion technique. The most similar requirements added after expansion are highlighted, and each requirement has a different color.

as shown in Table 6.1.

6.2.3 Feature Representation

This setup, which applies to both the training and testing phases, transforms textual requirements into a vector of TF-IDF weights (similar to ML4RC method described in Section 5.1.4). For training phases, the original requirements are transformed, while in the testing phase, the extended requirements that were obtained from the requirement extension step are converted into a vector representation.

6.2.4 Classifier Training

This step is to train a classifier F using the SVM algorithm as it shows the best performance with ML4RC (see Section 5.2). F is a multi-class classifier trained on the TF-IDF matrix to learn the association between the requirements in the training dataset and their preassigned categories (i.e., the requirement-category pairs). Based on the learning, this classifier is then ready to classify each new requirement in the new data (testing dataset) into a specific category label (e.g., functional, security, or usability).

6.2.5 Classifier Testing

This step is to test the trained classifier obtained from the training phases. It includes applying the pre-trained ML classifier to the expanded requirements. The outputs of this classification are the categories of testing requirements (i.e., performance, security,

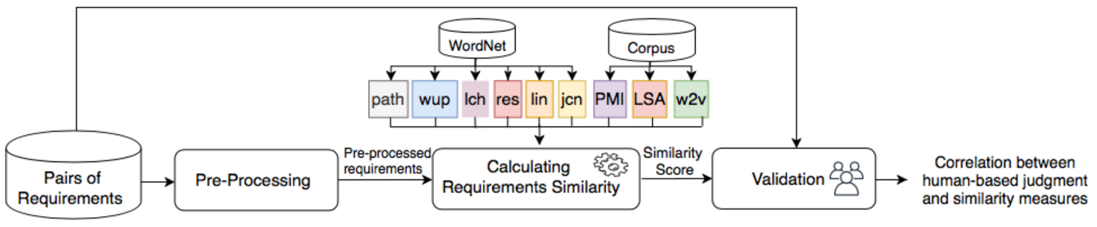

WordNet
Corpus
path
wup
lch
res
lin
jcn
PMI
LSA
w2v
Pairs of Requirements
Pre-Processing
Pre-processed requirements
Calculating Requirements Similarity
Similarity Score
Validation
Correlation between human-based judgment and similarity measures

Figure 6.3: The procedure of the comparison of different similarity measures

usability, etc.).

6.3 Evaluation of Requirements Similarity Measures

Before evaluating the SE4RC method, we evaluate the effectiveness of the similarity measure used in SE4RC (i.e., weighted word embedding with cosine distance). The key objective of this evaluation is:

> To compare the weighted average word2vec with other popular techniques to demonstrate the effectiveness of word2vec in measuring similarity between requirements.

To fulfill this objective, we compare the similarity measure of SE4RC with nine different similarity measures that are commonly used to determine the similarity between short texts [MCS$^+$06, II09]. This section presents the procedure of this evaluation and the implementation environment, and then provides an analysis of the performance of these measures.

6.3.1 Evaluation Procedure

Figure 6.3 shows the procedure of evaluating the similarity measures. The input of this procedure is a set of pairs of requirements. Each pair would have to go through three main steps: pre-processing, calculating requirement similarity, and validation.

Pre-Processing

We use several NLP techniques described in Section 3.4.3 to pre-process the input requirements. These techniques are tokenization, stop words removeable, lower cases transformation, and lemmatization.

Calculating Requirement Similarity

The semantic similarity between pairs of pre-processed requirements is computed using nine different measures (knowledge-based and corpus-based), which have been commonly used to measure short-text similarity. Most of these methods were proposed to measure the similarity between words; thus, we applied Eq. 6.3, which was proposed by Mihalcea in 2006 [MCS+06], to measure the similarity between two requirements (R_1, R_2) as the following:

$$\begin{aligned} sim(R_1,R_2) = \frac{1}{2}(&\frac{\sum_{w\in R_1} maxSim(w,R_2)\times idf(w)}{\sum_{w\in R_1} idf(w)} \\ &+\frac{\sum_{w\in R_2} maxSim(w,R_1)\times idf(w)}{\sum_{w\in R_2} idf(w)}) \end{aligned} \tag{6.3}$$

This function is based on finding the word that has the highest semantic similarity in the other requirement (i.e. maximum similarity). The word similarity is then weighted with its specificity (idf). For knowledge-based measures, maxSim is only used to determine the similarity between nouns and verbs as most of these measures are applied only on these POS groups[CM05]. However, for adjectives and adverbs, lexical matching was performed (similar to Mihalcea et al.'s work).

The nine similarity methods are mentioned below; the first six methods are knowledge-based measures, and the rest are corpus-based measures.

1- Path Length (path) measures the similarity between two concepts (i.e. words) based on the shortest path connecting the concepts in the WordNet taxonomy [MHG13]. The path between one word to another is computed by number of edges between the word (as shown in Figure 6.4).

2- Wu & Palmer (wup) [WP94] measure the similarity based on the depth of two concepts in the WordNet taxonomy and the depth of the Least Common Subsumer (LCS). Figure 6.4 shows an example of LCS.

$$sim_{wup} = \frac{2\times depth(LCS)}{depth(concept_1)+depth(concept_2)} \tag{6.4}$$

3- Leacock & Chodorow (lch) [LC98] measures the similarity between two concepts based on the shortest path between the concepts in the hierarchy of WordNet (length in Eq. 6.5). This path is scaled by the maximum depth of the hierarchy in which the concepts occur (D in Eq. 6.5).

$$sim_{lch} = -\log \frac{length}{2 \times D} \tag{6.5}$$

4- Resnik (res) [Res95] measures the similarity between two concepts based on the information content of the least common subsumer (i.e. LCS).

$$sim_{res}(w_1, w_2) = IC(LCS(w_1, w_2)) \tag{6.6}$$

where IC is defined as:

$$sim_{lch} = -\log P(c) \tag{6.7}$$

P(c) is the probability of finding an instance of concept c in a given corpus (i.e., information content).

5- Lin (lin) [L$^+$98] builds on Resnik's measure by scaling the information content of the LCS with the sum of the information content of two input concepts.

$$sim_{len}(w_1, W_2) = \frac{2 \times IC(LCS(w_1, w_2))}{IC(w_1) + IC(w_2)} \tag{6.8}$$

6- Jiang & Conrath (jcn) [JC97] expands on Resnik's measure by taking the difference of the sum of the information content of the individual concepts and the information content of their LCS.

$$sim_{jcn}(w_1, w_2) = \frac{1}{IC(w_1) + IC(w_2) - 2 \times (LCS(w_1, w_2))} \tag{6.9}$$

7- Pointwise mutual information and Information Retrieval (PMI-IR) [Tur01] measures the similarity between two words based on word co-occurrence over a large corpus; the more often words occur together, the higher the similarity achieved. Given two words, w1 and w2, their PMI-IR is measured as:

$$PMI - IR(w_1, w_2) = \log 2 \frac{p(w_1 \& w_2)}{p(w_1) \times p(w_2)} \tag{6.10}$$

8- Latent semantic analysis (LSA) [LFL98] assumes that there is some underlying semantic hidden relationship (i.e., latent) by the diversity of the context of words. LSA uses singular value decomposition to decompose a term-document matrix with term frequency-inverse document frequency (TF-IDF) [Jon72] weights into a small semantic space. The similarity between two words is calculated by transforming a word representation to LSA representation, and using cosine similarity (see Eq. 6.11)

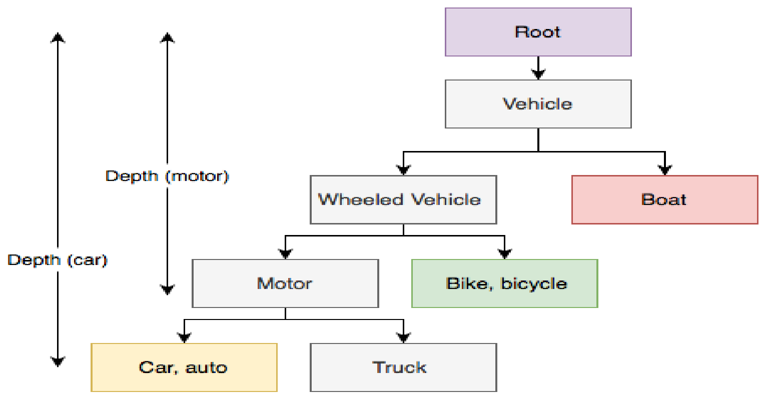


Figure 6.4: An example of the hyponym taxonomy in WordNet. It is shown in this figure that Least Common Subsumer (LCS) of "boat" and "car" is "vehicle", bath between " car" and "Boat" is 4 , maxim depth of "car" is 4.

on the compressed vectors.

$$\cos(\theta) = \frac{v_1 \times v_2}{||V_1|| \times ||V_2||} \tag{6.11}$$

9- Word2vec (w2v)[MSC+13] is measured in two ways: sentence-based and word-based methods. The first method is based on sentence representation by averaging weighted word embedding (using IDF) and computing the distance between these embeddings using cosine (i.e. weighted average `word2vec`). This is the method used by SE4RC. The second way is based on word representation by applying the function from Mihalcea et al. (Eq. 6.3). The similarity between words is measured directly by computing the distance (using cosine) between word embeddings.

Validation

This step validates the quality of each measure in determining the similarity between requirements. The quality of the similarity-measuring methods is based on two main criteria: human judgment (i.e., the ability to provide an effective simulation of human judgment) and execution time (i.e., time taken to perform each similarity measure). Human judgment includes asking the human annotators to measure the similarity between pairs of requirements used in the previous step. Then, the results obtained by the annotators are compared with implementing the nine different measures separately. The measurement method that is more correlated to human judgment performing at a reasonable time is the best.

6.3.2 Implementation Environment

We implemented all the similarity measures on a standard laptop with an Intel Core i5 1.6GHz and 8 GB RAM. The implementation was carried out using the Python programming language with the support of Python's NLP toolkits for performing the following tasks:

- Pre-processing was implemented using NLTK's text pre-processing module.
- Requirements similarity was calculated using different Python toolkits and libraries. For example, NLTK was used to implement similarity measures belonging to the knowledge-based approach and PMI. Sklearn (using truncatedSVD function) was used for implementing LSA, Gensim package for generating word embedding, Numpy to [2] generate requirements embedding, and Scipy [3] to calculate the cosine similarity between the requirements embedding.

Information Content of WordNet are derived from existing variable corpus such as Brown [KF68] and SemCor corpus [MLTB93] through NLTK' corpus package. These corpora are used for calculating the similarity using Knowledge-based measures (e.g., Resnik, Lin, and Jiang & Conrath measures). PROMISE-exp [LVC$^+$19] is used to implement corpus-based measures and to compute IDF in Eq. 6.3.

The available pre-trained word-embeddings model which was proposed by Efstathiou et al. [ECS18] (see Section 6.1.3), is used in generating word embedding. We chose this model as it is more relevant than other available models (such as the model trained using general news articles by Google [4]) and it is larger than the available NFRdataset (i.e., PROMISE-exp).

6.3.3 Implementation Steps

Our implementation of the evaluation procedure as described in Section 6.3.1 is as follows:

1. We sampled 100 pairs of requirements (FRs and NFRs) from PROMISE-exp and the book by Robertson and Robertson [RR12]; about 50% were similar and 50% were dissimilar based on our initial analysis.

[2]https://numpy.org
[3]https://www.scipy.org/
[4]https://code.google.com/archive/p/word2vec/

Threshold	**Proportion of similar pairs**	**Proportion of non-similar pairs**
≥ 0.90	14%	86%
≥ 0.80	27%	73%
≥ 0.70	27%	73%
≥ 0.60	40%	60%
≥ 0.50	40%	60%

Table 6.2: Proportion of similar pairs according to annotation results within different similarity thresholds.

2. Five human users employing Amazon's Mechanical Turk[5] were asked to to scale the similarity of each pair as "Similar" or "Not Similar". The dichotomous scale is used to get clear and binary answers, avoiding respondents being neutral. It also reflects the need to determine the semantic similarity; similar requirements will be used for expansion and non-similar requirements will be discarded.

3. The scales were coded as 1 ("Similar") or 0 ("Not similar"). As there was more than one annotator, we calculated inter-rater agreement using Fleiss's Kappa [Fle71]. The kappa score was 30, which is considered being fair level of agreement [VG+05].

4. We computed the similarity of each pair separately using ten different similarity measures. In this way, 100 similarity scores for each measurement method are obtained. This step is done automatically through the environment described in the previous section.

5. The correlation between the mean of human-assigned similarity scores and the similarity score obtained from each similarity method was measured. A QQ Plot [WG68] suggested non-normal data and a Spearman rank correlation was therefore used [HK11].

Table 6.2 shows the proportion of similar pairs based on the annotation results within different similarity thresholds. The table shows that with threshold ≥ 60, the proportion of similar and dissimilar requirements are close to our initial analysis. This hints at the differences in measuring requirement similarity between people with and without RE knowledge. We will revisit this point in section 6.6.1.

[5]https://www.mturk.com

Measures	**Correlation**	**p-value**	**Exec. time (s)**
Path	0.25 (w+)	0.01	0.86
wup	0.21 (w+)	0.03	2.02
lch	0.24 (w+)	0.02	1.19
res	0.28 (w+)	0.01	0.94
lin	0.28 (w+)	0.01	0.94
jch	0.30 (w+)	0.00	0.94
PMI-IR	0.32 (w+)	0.00	2.43.
LSA	0.25 (w+)	0.01	19.3
`word2vec` (words)	0.33 (w+)	0.00	1.25
weighted avg. `word2vec`	0.45 (m+)	0.00	1.39

Execution Time was measured on a standard laptop with an Intel Core i5 1.6 GHz and 8 GB RAM.

Table 6.3: Correlation, p-value and execution time (exec. time) between the similarity measures and human-assigned similarity scores. Interpretation of raw correlation values given in brackets: w+ (weak positive) and m+ (moderate positive). All p-values significant at the 0.05 level.

Results

Table 6.3 shows the quality of each similarity measure. It is clearly shown in table 6.3 that the similarity method used by SE4RC (Weighted average word2vec) has the highest correction score (0.45), which indicated moderate correlation [Ako18]. Weighted average word2vec also showed a reasonable execution time (i.e., 1.39 s).

Another key observation from Table 6.3 is that corpus-based methods were better than knowledge-based methods in determining requirements similarities. This observation aligns with those of Mahmoud and Williams [MW16], who found that corpus-based methods are more successful than knowledge-based methods in measuring the similarity between words that describe FRs. Nevertheless, knowledge-based methods are more efficient (i.e., incur less execution time) than corpus-based methods. LSA took the longest time, at 19.3 s for 100 pairs.

6.4 Experimental Comparison with Baseline Method

After evaluating the similarity measure in SE4RC, we evaluate the effectiveness of the whole ML4RC in this section. This key objective of this evaluation is:

> To investigate the effectiveness of our SE4RC in relation to a baseline method applied with non-expanded requirements (i.e., not using a feature extension technique).

To fulfill the evaluation objective, we compare the performance of SE4RC with the baseline classifier. This section presents the details of this evaluation, including the implementation of the SE4RC and baseline method, and an analysis of the performance of these methods.

6.4.1 Implementation Environment

We implement the SE4RC using the Python Programming Language with the support of Python's NLP and ML toolkits for performing the following tasks:

- Pre-processing was implemented by NLTK's text pre-processing module.
- Requirement extension was implemented using Gensim (for generating words embedding), Numpy (for computing requirements embedding), and Scipy (for calculating the cosine similarity).
- Feature Representation, Classifier Training, and Classifier Testing implemented by scikit-learn. The Linear Support Vector Classification model (i.e., svm.SVC [6] in scikit-learn) was used for implementing the SVM algorithm.

To conduct a fair comparison, each SVM model (SE4RC and baseline) is trained and tested using the same dataset (PROMISE-exp detest). PROMISE-exp is described in Section 5.2.1 and a summary of the dataset is provided in Table 5.3.

6.4.2 Implementation Steps

We validated the ML4RC method in a 10-fold cross-validation process as follows:

1. Apply scikit-learn's StratifiedKFold tool to divide the PROMISE-exp dataset randomly into 10 folds (i.e. 10 portions), nine portions for algorithm training and one portion for testing.
2. Use NLTK to carry out Pre-processing task, scikit-learn's TfidfVectorizer tool for Feature Representation.
3. Train SVM classifier with optimal hyper-parameters that have been tuned using scikit-learn GridSearchCV with 5-fold cross-validation.

[6]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

4. Apply the requirement extension technique by expanding the testing requirement set with the most similar requirements used to train the classifier (training dataset used in the previous step).

5. Test the trained classifier on the testing set to produce the predicated classification results (i.e., a set of predicated category).

6. Compare the predicated labels with the true labels provided in the original PROMISE-exp, and measure the difference between the true and the predicated labels for each category using the Precision (P), Recall (R) and F1-Score (F1) metrics. The overall performance (P,R, and F1 metrics) of the current classifier is also measured by computing the macro and micro average of all the classes.

The process is repeated 10 times with each of the 10 portions used exactly once as a testing set, while the rest is a training dataset. Afterword, the means P, R. and $F1$ values of each category, macro and micro averages are computed. Moreover, the learning curve of overall F1-score is plotted.

The baseline method was implemented in the same way as the SE4RC, except for step 4 (requirement extension). Table 6.4 shows the results for SE4RC and baseline method. Figure 6.5 shows the learning curves for these methods.

6.4.3 Results

Classification Performance

Table 6.4 shows the overall performance of SE4RC ($P = 0.55, R = 0.48, F1 = 0.49$) as well as the detailed performance of each class. It can be seen from the table that that SE4RC performs well on six categories: Function (over 70 %), Security, Performance and Availability (over 60%), Usability, and Legal (over than 50%). Two out of six classes (Legal and Availability) are minority classes, while the remaining are majority classes. Both of Legal and Availability have intuitive and distinctive features, e.g., such as "available", and "24/7" for the former, and "regulations" and "law" for the latter. However, only the results of the Legal class showed an improvement in comparison with the baseline, indicating the usefulness of SE4RC in classifying Legal requirements.

In compassion with the baseline method, the overall performance of SE4RC outperforms the baseline classifier by 0.02% in P, R, and $F1$ as shown in Table 6.4. For the individual requirement class, ME4RC has an improvement over the baseline in a

Cat.	Baseline Classifier			SE4RC		
	P	R	$F1$	P	R	$F1$
F	0.69*±0.04	0.83 ±0.12	0.75 ±0.05	0.72 ±0.05	0.84 ±0.13	0.77 ±0.05
SE	0.69 ±0.19	0.65 ±0.15	0.64 ±0.13	0.70 ±0.18	0.65 ±0.15	0.65 ±0.13
US	0.62 ±0.22	0.51 ±0.23	0.55 ±0.22	0.60 ±0.19	0.53 ±0.19	0.55 ±0.17
O	0.41 ±0.13	0.46 ±0.26	0.42 ±0.17	0.42 ±0.13	0.46 ±0.26	0.43 ±0.17
PE	0.88 ±0.13	0.65 ±0.22	0.71 ±0.18	0.80 ±0.17	0.68 ±0.15	0.71 ±0.11
LF	0.69*±0.37	0.39 ±0.19	0.46 ±0.22	0.80 ±0.29	0.41 ±0.13	0.51 ±0.17
A	0.70 ±0.38	0.67 ±0.37	0.66 ±0.35	0.69 ±0.38	0.63 ±0.35	0.65 ±0.35
MN	0.45 ±0.47	0.33 ±0.39	0.37 ±0.40	0.40 ±0.44	0.33 ±0.39	0.34 ±0.37
SC	0.55 ±0.47	0.40 ±0.38	0.45 ±0.40	0.55 ±0.47	0.45 ±0.42	0.49 ±0.43
FT	0.25 ±0.40	0.20 ±0.33	0.20 ±0.31	0.35 ±0.45	0.25 ±0.34	0.27 ±0.33
L	0.45 ±0.47	0.40 ±0.44	0.40 ±0.42	0.55 ±0.47	0.50 ±0.45	0.50 ±0.43
PO	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
Mic	**0.65** ±0.07	**0.65** ±0.07	**0.65** ±0.07	**0.67** ±0.05	**0.67** ±0.05	**0.67** ±0.05
Mac	**0.53** ±0.04	**0.46** ±0.05	**0.47** ±0.04	**0.55** ±0.04	**0.48** ±0.05	**0.49** ±0.04
Ex. Time	3.3 minutes			6.5 minutes		

* Indicates a significant statistical difference between the baseline and another classifier with p<0.05. Wilcoxon signed-rank test has been applied.

Table 6.4: 10-Fold Cross-Validated Test Results for the Four Classifiers

half of classes (6 classes) in which the improvement is in $P, R, F1$ for Fault tolerance, Functional, and Legal. The improvement was also noticed in R and $F1$ for Usability, Performance, and in P and $F1$ for Security. The biggest improvements were noticed in Legal and Fault tolerance, and the statistically significant improvements were in the precision of Functional and Look and feel. No changes have been observed with Operational, Scalability, and Portability. However, the performance of Maintenance and Availability was decreased.

It is intuitive to conclude that the improvement and decrease in SE4RC is because of expanding short requirements with similar requirements because it is the only difference between the two methods. By observing the prediction of baseline and SE4RC classifiers, we found that expanding requirements with similar requirements belonging to different classes increased the noise, resulting to misclassification of the expanded requirements. For example, although available requirements have distinctive features, they, as a whole, have many similar requirements belonging to different classes. Examples of these classes are those with the same fit criteria (e.g., period of time), including Performance, Maintainability, and Usability. Performance and Usability are majority

First Example						
Original Requirement	The product shall be available 99 of the time to avoid service interruption during busiest customer service response periods.					
Expanded Requirements	The product shall be able to continue to operate with no interruption in service due to new resource additions. (**MN**) The response time of general student management tasks shall take no longer than 5 seconds and the response time of schedule generation shall take no longer than 30 seconds. (**PE**) The system must be available for use between 12:00AM and 6:00PM all days of the year. (**A**)					
Labels	True	A	Baseline	A	SE4RC	MN
Second Example						
Original Requirement	IzognMovies shall meet the licensing requirements set forth by the appropriate organizations in order to stream African movies.					
Expanded Requirements	The Disputes application must conform to the legal requirements as specified by the Merchant Operating Regulations. (**L**) All business rules specified in the Disputes System shall be in compliance with the Merchant Operating Regulations. (**L**) The product must comply with the intranet page standards and requirements of ENET Securities. 95% of the product look & feel will be approved by the Architecture group prior to implementation. The remaining 5% will be corrected and approved within one month of product release. (**LF**)					
Labels	True	L	Baseline	F	SE4RC	L
Third Example						
Original Requirement	The system shall support the ability to perform a send and receive operation.					
Expanded Requirements	The system shall allow the mediator to send and receive messages from users. **(F)** The system shall allow the initiator to send and receive messages from users. **(F)** The system shall allow users to send and receive messages. **(F)**					
Labels	True	F	Baseline	O	SE4RC	F

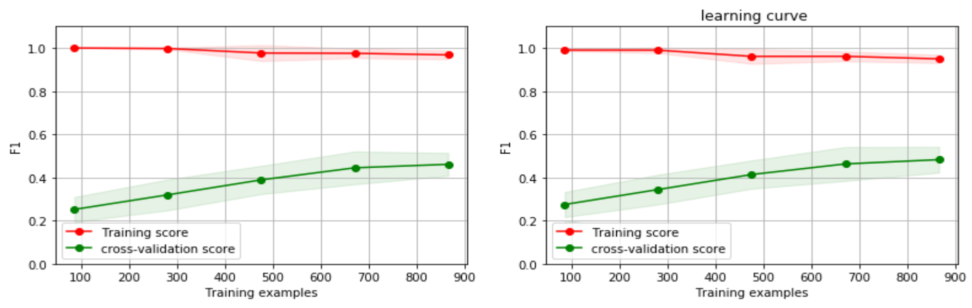
Table 6.5: Examples of the requirement that are added to the original ones and the changes that expansion made in the classification prediction

classes, therefore, they are less affected by the expansion errors as the number of expanded requirements belonging to the same class are more than those belonging to different classes. Therefore, minor classes (Available and Maintainability) were affected more than the majority classes. Table 6.5 shows the prediction of an available requirement by the baseline and SE4RC methods.

On the other hand, the requirement that belongs to the minority classes with less similarity with other classes is significantly improved (e.g., Legal and Fault Tolerance). We noticed that most of the similarity expanded requirements were within the same class, leading to the original requirements being predicted correctly. Table 6.5 shows how requirement expansion improves the classification accuracy of a Legal requirement.

Learning Curves

The training curves of baseline and SE4RC in Figure 6.5 show that all methods start with a high score ($F1 = 1.0$), then gradually slide downward as training progress to ($F1 \geq 0.95$), with the SE4RC curve showing a slightly bigger decrease. In contrast, the testing curves start with a low score ($F1 < 0.30$), but gradually move upwards to ($F1 \leq 0.50$), where SE4RC has a slightly bigger score. The slight increase in the testing curve and decrease in training curve of SE4RC indicates that SE4RC is less prone to suffer from overfitting than the baseline method. The curves also show that adding more examples might increase the generalization for the methods and that the probability is slightly higher with SE4RC.

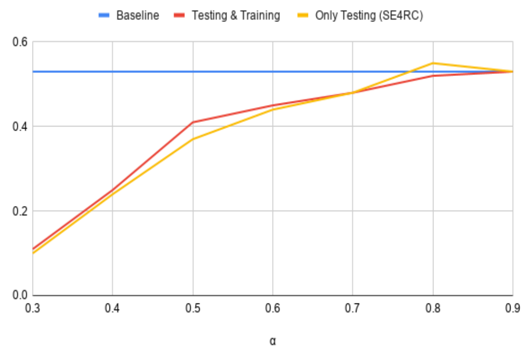


(a) Baseline (b) SE4RC

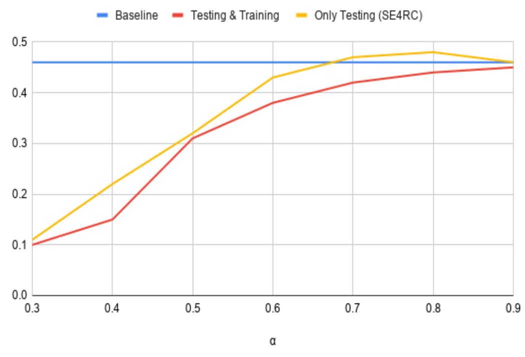
Figure 6.5: Learning Curves of the Baseline and SE4RC methods Based on 10-Fold Cross-Validation.

Threshold Setting Change

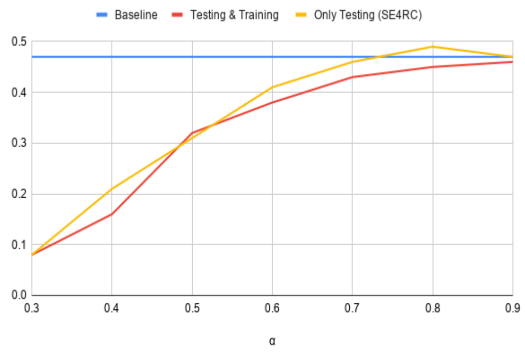
All the results of SE4RC reported before were based on using ($\alpha > 0.80$) in requirement expansion technique. The similarity threshold value was empirically set based on our preliminary experiments. These experiments include evaluating the performance of SE4RC with different values of threshold in the range of $0.30 < \alpha > 0.90$, where the value incrementally increases by 0.10 each time. These experiments also compare the performance of expanding training and testing requirements against expanding only testing requirements (i.e., SE4RC) in NFRs classification. Figure 6.6 shows the results of our preliminary experiments, indicating that expanding only testing results shows higher performance (especially in R & $F1$) than expanding both testing and training datasets, and $\alpha > 0.80$ has the best performance with only an expanding testing curve (SE4RC).



(a) Precision



(b) Recall



(c) F1-Score

Figure 6.6: The results of our preliminary experiments aim to set the similarity threshold (α) of SE4RE and comparing the performance of expanding the requirements in the training and testing dataset or only expanding the testing dataset (i.e., SE4RC).

Figure 6.6 also shows that the small value of α leads to low performance of SE4RC due to the expanding with dissimilar requirement. On the other hand, the high value of α (i.e., $\alpha > 90$) does not show a significant change over the baseline, indicating a lesser number of similar requirements and resulting in a few extended requirements.

6.5 Experimental Comparison of Related Methods

The previous section compared SE4RC with the baseline, whereas this section compares SE4RC with other classifiers. In particular, the key evaluation objective of this section was the following:

> To investigate the effectiveness and efficiency of our approach versus related methods applied to address the short-text classification problem through the feature extension method.

To fulfill this objective, we compare the performance of SE4RC with the following two methods:

1. The method by Lu and Liang [LL17]
2. The method by Man [Man14]

There are two main reasons for selecting these methods for comparison. First, the process description of these methods is relatively clear or can be inferred from the description; so, we can reconstruct these methods based on their description and objectively assess the performance of these methods against SE4RC. Second, these methods are closely related to ours as they all aim at expanding short text using internal sources (i.e., training data set) for supervised classification. Lu and Liange's methods expand user reviews to be classified according to NFR categories, and Man's method expands news titles to be classified according to the article categories (entertainment support, etc.). The summary of these methods are provided in Table 6.6.

6.5.1 Experiment Execution

To conduct a fair comparison, we used the same dataset used in evaluating SE4RC (i.e., PROMISE-exp described in section 5.2.1). As the source codes of these methods are not available, implementing these methods becomes the first necessary step in our comparison. In the section below, we describe how we implement Man's method, where Lu and Liang's methods are described in section 5.3.1. Then, we compare the performance of these two methods with SE4RC.

Implementing the Method by Man [Man14]

This method addresses the short text problem through expanding short texts with additional features that are extracted using the frequent terms set method (also known as frequent item set) [Bor12]. Man used the frequent terms set method to extract double terms set from the training dataset; two terms frequently appear together (having co-occurring relation) within an identical class (having class orientation relations). The double terms set is then used to build background knowledge as shown in Figure 6.7. During the expansion task, if the original text has a word showing in a pair(s) of the double terms set, the second word in this pair(s) will be added to the original features. In Man's method, both the training and testing datasets are expanded.

The procedure of Man's method includes four main steps: 1) pre-possessing (e.g., stemming and tokenization), 2) Information Gain for feature selection, 3) term frequency for feature weighting and representation, and 4) SVM for building a classifier.

Dataset	**ML Algorithm**	**Key Techniques**	**Classifier Training & Testing**
Lu and Liang [LL17]			
4000 user review sentences from iBooks and WhatsApp	DT with Bagging method	*Feature extension*: Word2Vec used to expands each user review with the most similar words; *Feature representation*: Bag of Words	A multi-class classifier is trained to classify a given user review into one of these four types of NFRs: Reliability, Usability, Portability, and Performance; 10-fold CV for classifier training and testing, with $P = 0.71$, $R = 0.72$, $F1 = 0.72$
Man [Man14]			
News title containing of 36 K documents labelled according to 9 categories	SVM with linear kernel	*Feature extension* Frequent term based method are used to build background knowledge which used to expand short text	A multi-class classifier is trained to classify news titles into one of the nine categories (e.g., Sports, Society and Entertainment). Holdout evaluation method, with $P = 0.81, R = 0.80, F1 = 0.81$

Table 6.6: Related Methods Used to Compare SE4RC

Man employed the hold-out validation method (70:30 training-test split) over the news title dataset to train and test the classifier. These titles are labeled according to nine classes (e.g., International, Sports, Society, and Entertainment), where each class contains 4000 documents. The background knowledge was built based on the content of the articles.

As this method does not provide implementation details, we made the following modification:

- Implementation tool: We used Python as the main implementation tool, NLTK for pre-processing, scikit-learn for feature representation, and ML model training and testing.

- Pre-processing: We applied stop-word removal in addition to the techniques mentioned in the original method. Although this technique is not reported in the study, it is important for our application. Without pre-processing the text, most of the terms set are stop words and, consequently, the classifier's performance is negatively affected.

- Feature selection: we implement information gain based on the equation provided by Yang and Pedersen [YP97] (Eq. 5.2), more in Section 5.4.2.

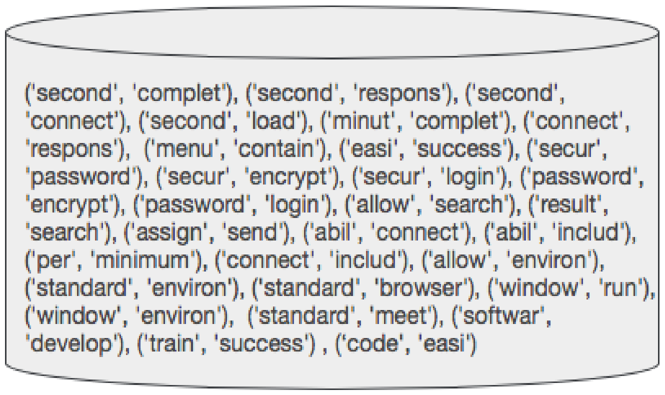

('second', 'complet'), ('second', 'respons'), ('second', 'connect'), ('second', 'load'), ('minut', 'complet'), ('connect', 'respons'), ('menu', 'contain'), ('easi', 'success'), ('secur', 'password'), ('secur', 'encrypt'), ('secur', 'login'), ('password', 'encrypt'), ('password', 'login'), ('allow', 'search'), ('result', 'search'), ('assign', 'send'), ('abil', 'connect'), ('abil', 'includ'), ('per', 'minimum'), ('connect', 'includ'), ('allow', 'environ'), ('standard', 'environ'), ('standard', 'browser'), ('window', 'run'), ('window', 'environ'), ('standard', 'meet'), ('softwar', 'develop'), ('train', 'success') , ('code', 'easi')

Figure 6.7: An example of double terms set obtained by applying Man's method on PROMISE-exp

α \ β	**0.01**	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**	**0.10**
0.01	835	667	474	285	170	132	113	85	67	39
0.02	44	44	39	28	14	11	8	4	3	2
0.03	4	4	4	3	3	4	3	3	2	1
0.04	0	0	0	0	0	0	0	0	0	0
0.05	0	0	0	0	0	0	0	0	0	0

Table 6.7: The number of extracted frequent term sets using different values of α and β by applying Man's method

- Classifier training: we build a multi-class classifier based on SVM with a linear kernel.

- Cross-validation: we used 10-fold cross-validation for training and testing, instead of hold-out.

- Threshold values: we set $\alpha = 0.01$, (co-occurring relation level) and $\beta = 0.05$ (class orientation relations). The β value is extremely less than the value used in Man's method ($\beta = 0.6$). However, the value used in Man's method does not extract any double terms as the dataset we used is smaller than what Man used (see Table 6.7). Thus, we chose the aforementioned values as they retrieve reasonable double terms (132) to build background knowledge and showed better performance than the values mentioned in Table 6.7.

The results of this method is represented in Table 6.8.

6.5.2 Experimental Results

Classification Performance. Table 6.8 shows that SE4RC has the best performance, followed by Man's method, and finally Lu and Liang's method. Lu and Liang's method

Category	**Lu & Liang**			**Man**			**SE4RC**		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
F	0.69* ±0.04	0.88 ±0.09	0.77 ±0.04	0.66* ±0.06	0.82 ±0.07	0.73 ±0.05	0.72 ±0.05	0.84 ±0.13	0.77 ±0.05
SE	0.63 ±0.15	0.56* ±0.15	0.57* ±0.10	0.66 ±0.21	0.51* ±0.17	0.55* ±0.15	0.70 ±0.18	0.65 ±0.15	0.65 ±0.13
US	0.45* ±0.20	0.40* ±0.20	0.41* ±0.18	0.45* ±0.28	0.42 ±0.30	0.42 ±0.28	0.60 ±0.19	0.53 ±0.19	0.55 ±0.17
O	0.34 ±0.15	0.30 ±0.16	0.31 ±0.15	0.36 ±0.18	0.32 ±0.14	0.33 ±0.14	0.42 ±0.13	0.46 ±0.26	0.43 ±0.17
PE	0.76 ±0.21	0.61 ±0.21	0.66 ±0.18	0.61* ±0.17	0.57* ±0.23	0.57* ±0.18	0.80 ±0.17	0.68 ±0.15	0.71 ±0.11
LF	0.36* ±0.39	0.14* ±0.13	0.18* ±0.17	0.49* ±0.03	0.29 ±0.18	0.32* ±0.15	0.80 ±0.29	0.41 ±0.13	0.51 ±0.17
A	0.49* ±0.34	0.44* ±0.27	0.34* ±0.26	0.25* ±0.27	0.29* ±0.35	0.26* ±0.29	0.69 ±0.38	0.63 ±0.35	0.65 ±0.35
MN	0.35 ±0.45	0.17 ±0.21	0.22 ±0.28	0.45 ±0.47	0.30 ±0.43	0.35 ±0.38	0.40 ±0.44	0.33 ±0.39	0.34 ±0.37
SC	0.45 ±0.47	0.22* ±0.22	0.29* ±0.30	0.42 ±0.44	0.35 ±0.40	0.35 ±0.37	0.55 ±0.47	0.45 ±0.42	0.49 ±0.43
FT	0.20 ±0.40	0.10 ±0.20	0.13 ±0.27	0.40 ±0.49	0.25 ±0.34	0.30 ±0.38	0.35 ±0.45	0.25 ±0.34	0.27 ±0.33
L	0.38 ±0.43	0.40 ±0.44	0.37 ±0.40	0.42 ±0.48	0.45 ±0.47	0.40 ±0.44	0.55 ±0.47	0.50 ±0.45	0.50 ±0.43
PO	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.10 ±0.03	0.10 ±0.03	0.10 ±0.03	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
Mic	**0.62** ±0.04	**0.62** ±0.04	**0.62** ±0.04	**0.59** ±0.05	**0.59** ±0.05	**0.59** ±0.05	**0.67** ±0.05	**0.67** ±0.05	**0.67** ±0.05
Mac	**0.43** ±0.20	**0.35** ±0.24	**0.35** ±0.21	**0.44** ±0.06	**0.39** ±0.05	**0.39** ±0.05	**0.55** ±0.04	**0.48** ±0.05	**0.49** ±0.04
Ex. Time	3.1 minutes			8.8 minutes			6.5 minutes		

* Indicates a significant statistical difference between the baseline and another classifier with $p<0.05$. Wilcoxon signed-rank test has been applied.

Table 6.8: 10-Fold Cross-Validated Test Results for the three Classifiers

No	Original Requirement	Expanded Requirement		
		Lu and Liang	**Man**	**SE4RC**
1	The product is expected to run on Windows CE and Palm operating systems.	The product is expected to run on Windows CE and Palm operating systems shall system system window product expect shall oper run	The product is expected to run on Windows CE and Palm operating system shall system week software level	The product is expected to run on Windows CE and Palm operating system The system shall run on Windows Server 2003.
2	The product shall have a consistent color scheme and fonts.	The product shall have a consistent color scheme and fonts color scheme system product shall consist	The product shall have a consistent color scheme and fonts product shall consist color scheme font system	The product shall have a consistent color scheme and fonts The product shall comply with corporate color scheme

Table 6.9: Examples of requirements before and after expanding them using different methods. The first requirement is labeled as Portability and the second as Look and Feel in PROMISE-exp.

is based on a different machine learning algorithm (DT) which might be the reason behind the poor performance of their method. The performance of Man and SE4RC seems to suggest that expanding requirements based on corpus-based methods is better than the occurrence-based method in NFRs classification. Table 6.9 shows examples of how requirements are expanded using each method, indicating that most of the words added by SE4RC are similar to those used in the original requirement. In contrast, the words added by Man's method could be unrelated to the original ones (e.g., week, level) in the first example. The words added by Lu and Liang are limited to the words represented in the original requirement. A comparison of the overall performance of these methods is depicted in Figure 6.8.

For the individual requirements category, SE4RC outperforms Lu and Liang's method in all classes except for PO, the most minority classes, which was not detected by both methods. Compared with Man's method, SE4RC shows higher performance in all the classes except Maintainability, Fault tolerance, and Portability. We noticed that these classes are minority classes and the biggest change was in precision while the recall was similar in Fault tolerance and Maintainability. Portability was not detected at all by SE4RC.

Statistically, the Wilcoxon signed-rank test shows that SE4RC has made a significant improvement over Lu and Liang's method in P, R and $F1$ of Usability, Look, and Feel and Availability, R and $F1$ of Security and Scalability, and only P of Functional. SE4RC also made a statistically significant improvement over Man's method in P of

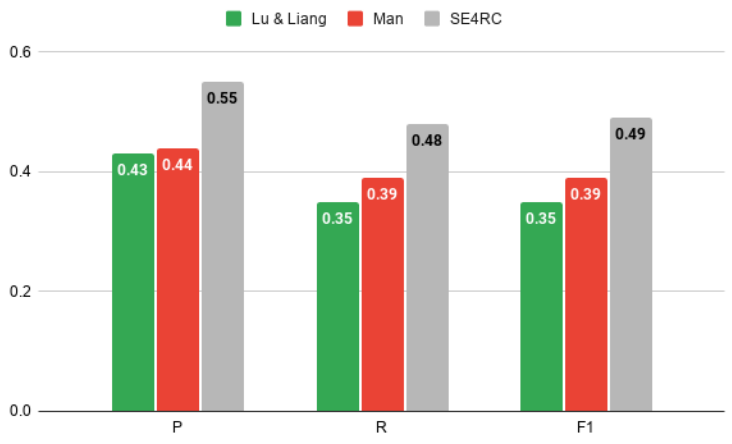
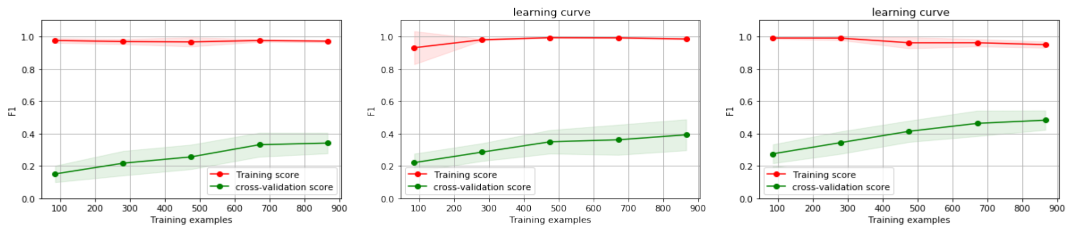


Figure 6.8: Overall performance comparison between three related methods



(a) Lu and Liange (b) Man (c) SE4RC

Figure 6.9: Learning Curves of Lu and Liange [LL17], Man [Man14] and SE4RC Based on 10-Fold Cross-Validation.

Functional and Usability, P and $F1$ of Look and Feel, R and $F1$ of Security, and P, R, and $F1$ of Performance and Availability.

Finally, in terms of the execution time, Lu and Liang's classifier is the fastest as it took 3.1 minutes to run, followed by the SE4RC classifier which took 6.5 minutes, and finally Man's classifier for 8.8 minutes.

Learning Curves. As shown in Figure, the training curves of all classifier start high (Lu $F1 > 0.95$, Man $F1 > 0.90$ and SE4RC $F1 \approx 1.0$). However, as the training progresses, the training curve of Lu stays mostly unchanged, Man increases to (F1 = 1.0), while SE4RC slightly declines to $F1 \approx 0.95$, indicating a low bias (low training errors) for all the classifiers. The testing curves of the three classifiers start low (Lu $F1 < 0.20$, Man $F1 \leq 20$, SE4RC $F1 \leq 0.30$) and move upwards gradually with more training examples. Among them, SE4RC moves upwards faster than the other classifiers, with ($F1 \approx 0.50$) for SE4RC at the end of the training and $F1 \leq 0.40$ for the other two methods. This indicates that SE4RC has fewer testing errors, suggesting that SE4RC suffers less from high variance (overfitting) than other classifiers.

6.6 Discussion

Considering the experimental results reported in Sections 6.3-6.5, this section presents some key findings from these results and discuss some research challenges we experienced throughout the development of the SE4RC method.

6.6.1 Measuring Requirements Similarity

Measuring similarity between requirements is a key part of SE4RC. The main finding of the first experiments, which determines the best similarity measure, is that corpus-based methods are better than knowledge-based methods in measuring similarity between requirements. Some of the methods belonging to the knowledge-based approach are limited to measuring the similarity between specific POS groups (e.g., nouns and verbs). This means that the similarity between adjectives and adverbs (e.g., " easy" and "simple") cannot be determined using these methods. Adjectives and adverbs have been considered the most probable POS groups used in NFRs [HKO08]. Ignoring such POS groups is a key reason behind the low quality of these methods in measuring the similarity between NFRs. Besides, the knowledge-based resource (i.e., WordNet) used in our experiment is not designed for software descriptions. For example, there is no relation between system and software but there is one between system and organization. Moreover, some technical words and their relations (e.g., programming languages such as Java and Python) are not defined in such resources (WordNet).

Although the corpus-based approach had better performance, the highest correction score achieved using this approach was 0.45 (moderate correction), indicating the difficulty of determining requirement similarities on an automated basis. The difficulty is also noticed in measuring the requirement similarity manually through the low inter-rater agreement score among the annotators (30), and the significant differences between our initial annotation for the 100 samples (50%-%50) and those provided by external annotators (reported in Table 6.2). In the following paragraphs, we discuss the possible reasons for this difficulty and how these difficulties impact the process of determining similarity manually (via annotators) or automatically (via implementation of similarity measures).

As previously stated, requirements are often written as short text which is challenging in its turn. In addition, besides containing explicit information (i.e., written text), requirements provide implicit information that cannot be understood in context [Mey93]; this increases analytical difficulties and thus creates confusion [oDRC^{+}01].

Examples of implicit information are perceptions and domain knowledge possessed by the requirement analyst [oDRC+01]. For example, the requirement: “The product shall prevent all personal and confidential data from being printed” and “the system shall optionally allow the user to print the invoice” can be considered semantically similar statements as they share the same concept (“print”). However, requirement analysts may have different perspectives as they consider the first requirement as security and the second as functional. Similarly, in our initial analysis of the 100 samples, we used our knowledge to measure the similarity while the external annotators did not. For example, we assumed that these two Scalability requirements are similar: "The product shall support 2,000 concurrent users." and "The product shall have the capacity for 5,000 roads." However, only one out of five annotators found that they are similar.

Besides the implicit information, different requirement categories have similar components (elements) that increase the difficulty of automatically measuring the similarity. For example, most NFR categories include measures (e.g., fit criteria) that can increase the similarity between dissimilar requirements. Consider, for example, the similarity scores of these two requirements provided in [RR12]: “The product shall produce the schedule within 3 seconds of the user’s request” and “The average music buyer shall be able to locate any piece of music within 6 seconds using no more than three actions”. Human annotators assigned to these requirements a score of 0, while that of the weighted average word2vec was 0.75. The criteria provided in the two requirements were quite different as the first one (3 seconds) is about how quickly the system provides the schedule (i.e., performance) while the second one (6 seconds) is about how quickly the music buyer can determine the location of a piece of music (i.e., usability). While the human annotators were able to distinguish these two requirements, the automated measure could not.

6.6.2 Semantic Expansion in NFRs Classification

The experiments’ results of this chapter confirm our finding of the previous chapter, demonstrating the effectiveness of expanding requirements in NFRs classification with SVM. However, this chapter’s results show that the effect of expanding short requirements with similar requirements (not words) is relatively slight in the overall classification performance. The difficulty of measuring similarity between NFRS is a key reason for this slight improvement. Besides, the size of the training dataset is small. We noticed a few similar requirements to be expanded in case of a high similarity threshold (i.e., 80%-90%). On the other hand, in the case with a low similarity threshold, most of the expanded requirements are less similar and belonging to different requirement classes, thereby increasing the classification errors. Large datasets can increase the possibility of having more similar requirements within the same class of requirements to be expanded. Nevertheless, the requirements expansion technique shows significant improvement in minority classes that are more distinct than others (i.e., less semantically similar to others), such as Legal and Fault Tolerance. The requirement-based similarity measures were able to find similar requirements belonging to these classes, leading to adding more features from the same classes, and, consequently, correctly predicting requirements.

The findings of the comparisons at Sections 6.5 and 6.4 can be concluded in three main points. First, expanding only the testing dataset works better, especially when the similarity measure is a corpus-based method. These methods retrieve different relations between words (e.g. opposite, part-of etc.) which can consequently bring the noise in NFRS classification. For example, we noticed that by applying Lu and Liang's similarity measure, some similar terms identified by the corpus-based method do not have a similar meaning (e.g., "schema" + "secure "). Moreover, our preliminary experiment (see Figure 6.6) shows that expanding only the testing dataset obtained a higher performance than expanding both training and testing datasets. A similar finding was made by Krzywicki et al. [KHB$^+$18] in expanding short text for supervised classification.

The second finding is that expanding methods based on word embedding work is better than co-occurrence measures due to the nature of overlapping in NFRs classification and short text of our dataset. By applying Man's method, we found that many of extracted frequent terms set are meaningless, i.e., not contributing to improving the classification accuracy, such as ('second', 'connect') and ('code', 'easi') in Figure 6.7.

The third finding is that expanding short text with additional requirements can help in improving the performance of minority classes (e.g., Legal, Fault Tolerance) as it increases the relation between the testing and training requirements, leading to correct classification of the requirements. Nevertheless, the improvement of classifier performance is linked with the quality of requirement similarity method.

6.7 Limitations and Threats to Validity

In this chapter, we use the same dataset and evaluation measures applied in the previous chapter. Thus, it exhibits similar limitations and threats to validity in the NFR classification. This section, however, reviews the threats related to measuring the semantic similarity among requirements.

Construct validity involves experimenter bias. This threat arises due to the subjectivity or inaccurate understanding resulting from manual construction. To measure the similarity, we randomly choose 100 pairs of requirements to determine the similarity between requirements. These pairs were annotated by five paid external annotators who did not have any information about our research goal (i.e., requirements classification). Thus, we believe there is little threat to construct validity.

Another potential threat to construct validity is the suitability and accuracy of our evaluation measures. In our experiments, we used rank correlation to select the optimal similarity method. Rank correlation is also widely used to evaluate the similarity between short texts [OSDCI11].

Internal validity threats of this study are quite low. In this study, to make any decision, we considered different attributes, including past applications and resources. For example, we choose nine similarity techniques to measure requirements similarity. These techniques are commonly used to measure the similarity of short text. For ML, as we mentioned in the section, a careful analysis was performed to choose the parameters used to build the ML model. This includes using K-fold cross-validation and the learning curves to measure the overfitting of each ML model.

External validity concerns the generalizability of our findings. To measure requirement similarity, we collected requirements from two different resources (dataset and a book). However, we applied the SE4RC method in a single dataset and classification task. This might threaten the generalizability of the SE4RC method. Thus, in the next chapter, we apply SE4RC to a different classification task and dataset.

6.8 Summary

In this chapter, we proposed an ML method (called SE4RC) for classifying NFRs in textual specifications. SE4RC aims to address short text classification by expanding each requirement with the most similar requirements extracted from the training dataset. We used weighted average word2vec to measure the similarity between requirements. We empirically evaluated SE4RC using PROMISE-exp against the baseline method (with no feature extension) and related methods that are proposed for short text classification. The results show that SE4RC outperforms the baseline method and related work.

One of the main conclusions of this chapter is that expanding the requirements with the most similar requirements can improve the performance of ML. However, there is still a need to improve the accuracy of measuring the similarity between requirements. Most of the misclassified instances in our approach were due to expanding a requirement with an unrelated requirement. Nevertheless, determining the similarity between requirements is extremely difficult due to the multiple forms of information provided in requirements (explicit and implicit), similar writing styles (i.e. most contain fit criteria), and the short length of the requirements.

In the next chapter, we re-evaluate this method and ML4RC on a different classification task. This task aims to classify usability requirements into sub-classes (i.e., usability goals) using supervised classification.

Chapter 7

An Evaluation of the Proposed ML Methods in Usability Requirements Classification - A Case Study

"Simply stated, if the customer can't find a product, then he or she will not buy it."

Jakob Nielsen

Usability is an essential quality of all interactive systems that has to be considered during software development [FB03, May99, FVGB04]. Like any NFRs, early consideration for usability (in RE phase) helps in avoiding maintenance costs. The maintenance costs of usability, in particular, are often significant due to the large number of usability-related changes that are commonly requested after developing systems [JBSSA04, FVGB04, JMSS07]. The late adoption of these changes are difficult due to the incorporation of usability into the early architectural design. This leads to delivering systems that are less usable than they could be, which directly affects users' experience and satisfaction [JBSSA04, AML06].

Usability requirements (URs) are categorized as NFRs [LY98] and written in NL documents. Thus, the early detection and identification of these requirements are challenging for many reasons. Besides those mentioned earlier (the ambiguity in NL and nature of NFRs—scattered across requirement documents), usability is still not an exact concept. A vast number of attempts have been made over the past 30 years or more to define, refine, or classify this concept. Each of these attempts has relied on relevant scientific knowledge and focused on a specific domain and objective, leading to a lack of agreement on the exact meaning and application of this concept over the years [Tra18].

This uncertainty is evident in URs, with a lack of consensus on what URs entail and how they can be measured [LY98, AKSS03]. This perhaps contributes to usability issues being the source of most unresolved problems within the development community [ZXY$^+$17]. Study of URs is particularly challenging — few studies provide real-world examples of usability requirements [LY98], and (as indicated in Chapter 3) URs are rarely the subject ML classification attempts (compared, for example to other NFRS, e.g. security).

Motivated by all of the above, in this chapter, we report a case study conducted in the context of automatically classifying NFRs using the ML classifier, focusing on the case of classifying URs into further categories. The main aim of this study is to evaluate the effectiveness and efficiency of the proposed ML models (ML4RC in Chapter 5 and SE4RC in Chapter 6) in classifying URs. The purpose of this case study is two-fold: 1) identifying the UR categories in real-life usability requirements and existing usability definitions, and 2) investigating the effectiveness of ML classifiers in classifying URs into further categories.

This chapter is organized as follows: Section 7.1 describes the background and previous works related to usability and URs. Section 7.2 reports the case study methodology. Section 7.3 presents the results. Section 7.4 describes the findings and presents a discussion about them. Section 7.5 presents threats to validity. Finally, Section 7.6 provides a summary of this chapter.

7.1 Background

Many usability classification models have been proposed to define and measure usability, and a few models have been built for URs. This section provides an overview of both usability classification models and URs.

7.1.1 Usability Concept and Classification Models

In 1998, the international standard ISO 9241-11 [Iso98] provided this most cited definition of usability [ARMRMB14, MRARMB09]:

> "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

According to this definition, usability consists of three distinct aspects [FHH00]:

- Effectiveness—the accuracy and completeness with which users achieve their goals.
- Efficiency—the relationship between the accuracy and completeness with which users achieve certain goals, and the resources expended in achieving goals.
- Satisfaction— users' comfort with, and positive attitudes towards, the use of the system.

Each usability aspect may have one or more indicators that serve as measures (e.g., efficiency may be measured by task completion time and learning time [FHH00]). Since the inception of ISO 9241-11, a series of definitions for usability have been provided [Bev01] either through international standards or by academic work. Following are some examples of these definitions:

Usability standards. Bevan [Bev01] divides usability standards into four broad categories (see Figure 7.1) taken from two stakeholder perspectives: the consumer perspective and the developer/producer perspective. The two categories that focus on the consumer perspective are Quality in Use (for supporting software specification, design, and evaluation) and Product Quality (recommendations, guidelines, and criteria to evaluate the interface). These are both directly concerned with user needs and requirements [Bev99]. By contrast, Process Quality (methods used for user-center design activities and how to evaluate them) and Organizational Capacity (guidance for an organization to apply user-centered design throughout the entire product development life cycle) are from the developer/producer perspective as they focus on the development process. Because our research focuses on user needs and requirements, we have only studied the usability standards related to the first two categories. An example of these standards is ISO/International Electro-technical Commission (IEC) 9126- 1 (2001) [II04] which defines usability as Product Quality, which has four aspects: understandability, learnability, operability, and attractiveness. ISO/IEC 25010 (2011) [fSEC$^+$11] defines usability as both Product Quality and Quality in Use. The usability concerning Product Quality has six aspects: appropriateness/recognisability,

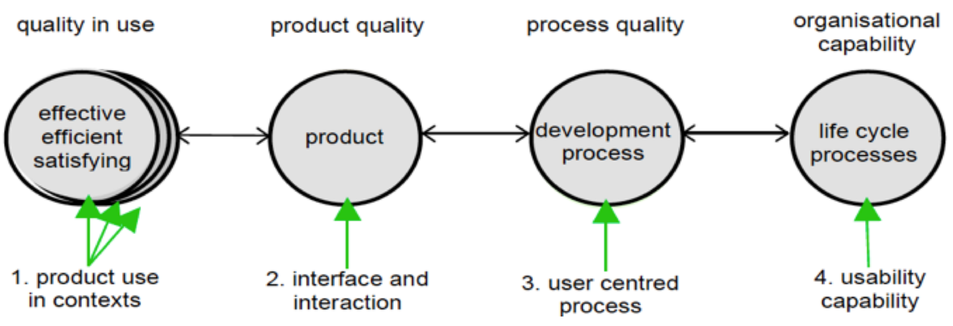


Figure 7.1: Four Categories of Usability Standards (from Bevan [Bev01]). Quality in Use and Product Quality are defined from the consumer perspective, whereas Process Quality and Organizational Capacity are from the developer/producer perspective.

learnability, operability, user error protection, user interface aesthetics, and accessibility. The usability related to Quality in Use has five aspects: satisfaction, effectiveness, efficiency, freedom from risk, and context coverage.

Academic work on usability. Many usability models and definitions contributing to the development of usable software systems have been proposed by the academic community. Examples of such models include that proposed by Nielsen [Nie94], who defined usability through five quality components: efficiency, satisfaction, learnability, memorability, and errors. Abran et al. [AKSS03] extended the ISO 9241-11 standard [Iso98] by adding learnability and security to the three original aspects, which are efficiency, effectiveness, and satisfaction. Seffah et al.[SDKP06] proposed a consolidated mode that contains 10 usability aspects. In addition to the three original aspects of ISO 9241-11 standard, their model defines seven additional aspects, which are learnability, productivity, safety, accessibility, universality, trustfulness, and usefulness. Shackel [Sha09] developed a usability framework consisting of four components: effectiveness, learnability, flexibility, and attitude. Alonso-Ríos et al. [ARVGMRMB09] proposed a hierarchical model consisting of six main aspects: efficiency, satisfaction, safety, knowability, operability, and robustness. Hasan and Al-Sarayreh [HAS15] proposed an integrated model of several existing models, which consists of 11 aspects and 35 sub-aspects. The sub-aspects are basically the measures or indicators of the usability aspects, which are effectiveness, efficiency, satisfaction, productivity, universality, learnability, appropriateness (or recognizability), accessibility, operability, user-interface aesthetics, and user-error tolerance. They divided their model into two main parts: usability during a system's development and usability while using the system.

7.1.2 Usability Requirements

URs, like other requirements, are defined in the early phases of project development and are used as an evaluation framework in later phases [TKV05]. URs specify how easy a system must be to use under development [LY98]. They, in other words, describe usability goals along with associated measures (method and acceptance criteria) [CWK05, TKV05], and illustrate how a system should be designed to meet the usability goals [TKV05]. Examples for usability goals are learnability or efficiency, examples for measurement methods are user test or questioners, and examples for acceptance criteria are time to complete task or number of errors. Achieving these requirements ensures that a system reaches the intended level of usability [LY98].

URs are commonly classified as NFR in Software Engineering and Human Computer Interaction disciplines. However, they could also be functional based on the fact that usability is an interdisciplinary field, combining interaction design and software architecture [SS20]. According to Juristo et al. [JMSS07], the requirements that describe usability goals (e.g., efficiency, effectiveness, and satisfaction) to be achieved are NFRs. An example of a non-functional UR is "A novice user should learn to use the system in less than 10 hours". While such requirements could use as a yardstick at the evaluation stage, they do not provide the developer with information about the kind of features to provide to satisfy such requirements. On the other hand, requirements that describe how usability goals could be achieved are functional usability requirements. These requirements are specific functional features that contribute to activate the usability level specified by non-functional usability requirements. For example, "The system should provide users with the ability to cancel actions".

URs differ from other requirements in that they are more strongly related to the user in specifying, testing, and accepting a system [CWK05]. These requirements are specified to support a user in performing particular tasks in the sense that the design of these tasks fulfill the criteria of usability [TKV05]. For example, URs determine the sequence of inputs and interaction steps that the user must take to perform a particular function. Therefore, in some cases, URs can interfere with functional requirements, such as the ability of users to perform a specific task (which can be considered as a functional requirement and as a usability requirement under the criterion of effectiveness) [TKV05]. Some usability goals can also interfered with non-functional goals such as safety and performance [WW97] that are considered a usability aspect in some usability classification models [ARVGMRMB09, WWD07, DGR12, HH93, Gou95].

Moreover, in some cases, it is difficult to coherently define and determine URs,

Category	Definition	Second Layer	Third Layer
Conceptual Requirements	Related to the type of service provided by a software	Learnability	Understandability, Completeness and Scalability
		Practicability	Accessibility and Calibrability
Functional Requirements	Operational features of the software	Operability	Functionality, Interoperability, Error tolerance and Simplicity
		Efficiency	Utility, Response time, Behavior, Sensibility and Stability
		Access Control	Security and Integrity
		Unambiguous	Consistency and Clarity
Non-functional Requirements	Non-operational features of the software	Helpfulness	Portability, Supportability and Availability
		Validity	Accuracy, Correctness and Availability
		Resilience	Manageability, Maintainability and Flexibility
		Testability	Adaptability
Business Requirements	Organizational needs and objective	Customizability	Cost& Schedule, Marketability, Organizational and Economy
		Affect	Reusability and Extensibility

Table 7.1: Usability requirements classification proposed by Kale et al.[KMMM10]

especially those related to the subjectivity of human experience. For example, the users need to perform a particular task within a specific amount of time while achieving a certain degree of satisfaction and a specific level of experience. Therefore, most URs require involving the users in the defining and testing phases [TKV05]. Moreover, in some cases (e.g., functional URs [JMSS07]), users might also not be a good source for completing usability requirements, which increases the need for human–computer interaction experts in defining these requirements, or alternatively, the use of such usability guidelines as suggested by Juristo et al. [JMSS07].

7.1.3 Usability Requirements Classification

A few attempts have been made to explicitly classify usability requirements. For example, Kale et al. [KMMM10] divided usability requirements into four different classes: conceptual, functional, non-functional, and business. Each class was further divided into sub-classes (see Table 7.1), but with poor subclass definition and substantial overlap. For example, availability is identified as a sub-class of helpfulness and validity.

Another attempt to classify URs has been made by Lauesen and Younessi [LY98],

URs Styles	Definition	Examples
Performance	This style is about defining tasks, user groups, and the performance objectives for user groups to perform these tasks.	Customers without previous ATM experience: In their first attempt, 90% of them must be able to withdraw a preset amount of cash within four minutes.
Defect	This style is similar to the performance style; however, it determines usability problems as well as the accepted level of these problems and their severity.	In their first attempt to carry out task A, users may not encounter more usability problems than task failures (at most, 0.2 per user).
Process	This style specifies the usability aspects of the design process, not the product.	In the design, a sequence of three prototypes must be made. Each prototype must be usability tested and the most important defects corrected.
Subjective	This style involves defining the criteria of satisfaction with the system.	80% of customers having tried the ATM at least once must find the system pleasant and helpful. 60% must recommend it to friends if asked.
Design	This is the traditional requirements style which specifies the screen interface and functions.	The system shall use the screen pictures shown in App. xx.
Guideline	This style defines the interface style guides and standards used to achieve high usability	The system shall follow the MS Windows style guide.

Table 7.2: The six styles of usability requirements (URs) provided by Lauesen and Younessi [LY98]

who identified six styles of usability requirements: performance, defect, process, subjective, design, and guideline. They also provided several examples as well as the pros and cons of each style. Table 7.2 summarizes these styles and examples. It is, however, not clear how these styles can be used to support requirements in the engineering process.

Apart from the few attempts that explicitly classified usability requirements, several studies implicitly classified usability requirements for different purposes and contexts [WW97, BM94, GSWG86, TKV05]. The classification, most often, was based on usability aspects (goals). For example, Terrenghi et al. [TKV05] used usability aspects as guidelines for specifying usability requirements in a mobile computing context. Becker and Berkemeyer [BB02] used requirements aspects for eliciting and testing usability requirements early using their proposed usability toolset. According to such studies, it can be concluded that the early classification of URs according to usability aspects can provide the following benefits:

- Making URs tangible and able to be tested and traced [ARVGMRMB09, WW97].
- Ensuring that the development team has a common understanding of usability

requirements and how to test them [ARVGMRMB09]

- Help in a systematic understanding of the usability problems by, for example, easily identifying the usability goals that might be missed or ignored
- Improving the communication between developers and usability experts by analyzing and addressing the usability of systems via these goals.

Motivated by the above, we will classify usability requirements automatically according to usability aspects in this case study. The following sections will show how these aspects are identified and used to automatically classifying usability requirements.

7.2 Case Study Methodology

The methodology of conducting the case study comprises four main steps: scoping, planning, data collection, and ML application.

7.2.1 Scoping: Establishing Aims and Objectives

The main goal of this case study is to investigate the effectiveness of ML models (i.e., ML4RC and SE4RC) for detecting and classifying URs. Thus, the objectives are defined as follows:

1. To define the usability aspects that can classify real-life URs and their interrelationships
2. To evaluate the effectiveness of the proposed ML models in classifying URs

In alignment with the research objectives, the research questions are defined as follows:

RQ1: What are the common categories of usability aspects that reflect and match real-life examples of URs?

RQ2: How well can the proposed methods classify NL requirements into the common set of usability aspects?

7.2.2 Case Study Planning

Figure 7.2 shows the overview of the methodology used in this case study consisting of two main phases: identifying the common usability aspects (to address RQ1), and

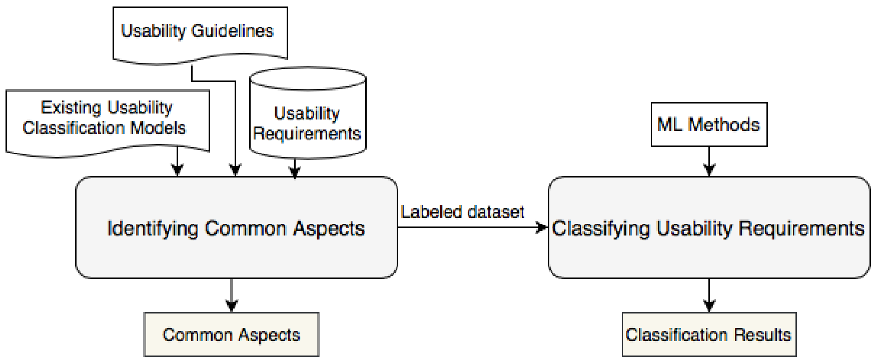


Figure 7.2: Phases in the Case Study Methodology

classifying URs using ML classifiers (to address RQ2). Brief descriptions of each are provided below, and more detailed descriptions are provided in Sections 7.2.3 and 7.2.4 respectively.

Phase 1: Identifying common aspects. This step aims at identifying common usability aspects that are used to characterize, define, and analyze software usability. The rationale behind the identification of common usability aspects is to explicitly represent a possible way of defining and classifying the URs due to a lack of widely accepted classification. A set of usability requirements is collected and used to identify the common aspects' interrelationships and refine the common aspects' definitions. The output of this step is the definition of common aspects which address the first research question in addition to pre-classified URs (i.e., labeled dataset) which will be used to build and evaluate ML models (phase 2).

Phase 2: Classifying URs using ML methods. This step aims at evaluating the effectiveness of ML models in classifying URs using the dataset developed in the previous phase. It includes applying ML methods that are described in Chapters 5 and 6 (i.e., ML4RC and SE4RC) separately to train SVM classifiers for predicting the common usability aspects identified in the previous step.

7.2.3 Data Collection, Analysis, and Validation

RQ1 requires the collection, analysis and validation of data to determine usability models and their sub-components (aspects), together with requirements that can be used to refine the definition of the aspects and build ML models. Figure 7.3 shows an overview of the method of collecting, analyzing, and validating each data point.

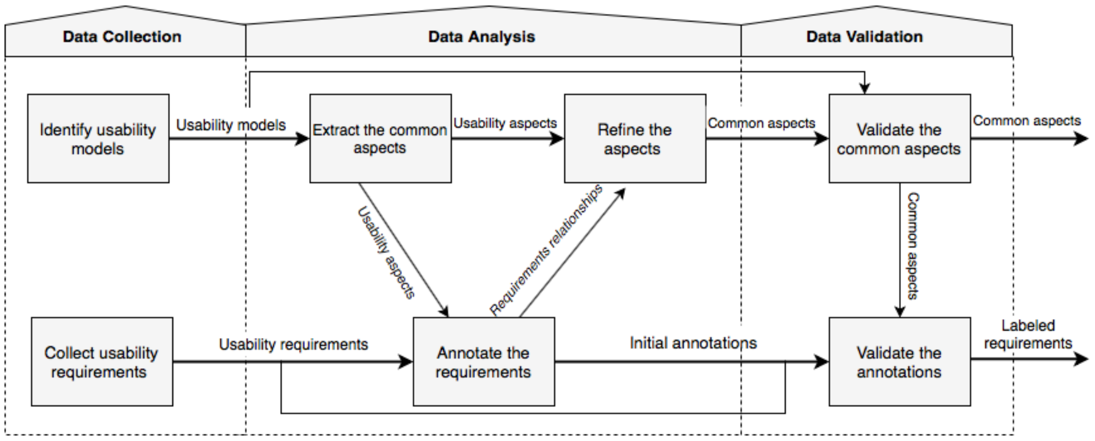


Figure 7.3: The process of identifying common aspects: collecting, analyzing, and validation data required to answer the first research question

Usability Models and Aspects

A set of usability classification models have been collected, analyzed, and mapped into a set of common aspects (also known as attributes, criteria, components, characteristics, or usability goals). Analyzing existing usability models is an important step to not only define the common usability aspects, but also recognize the relation between these aspects. We believe that the successful understanding of usability concepts (aspects) and their interrelationships will contribute to the proper identification and clear classification of usability requirements.

As shown in figure 7.3, the process of defining the common usability aspects consists of four steps: identify the models, extract the common aspect, refine these aspects, and validate them.

Step 1: Identifying Usability Models. We applied a snowball approach for performing the literature review to define usability classification models that provide a general definition of usability for a software system. We excluded the models proposed for specific user groups, application domain, context, or place. In order to restrict our research to more recent studies, we selected models published in the period between 1990 and 2020 (last 30 years when the review was conduced). Appendix C reports the procedures of the identification of these models in detail. The result of this review is 33 usability models summarized in Table C.5 in Appendix C.

Step 2: Extract the common aspects: We analyzed the identified models by listing all the usability aspects (around 226 aspects) and grouping the related ones to define the common aspects. This was not a straightforward step due to inconsistency in aspects' terms and definitions, with one name being assigned to different definitions and different names being used with the same definitions. Therefore, we grouped the aspects based on the definition (meaning), regardless of their names. We synthesized a comprehensive definition of each aspect according to the existing ones.

Step 3: Refine the aspects. The common usability aspects extracted from the previous step were then saved along with the synthesized definition derived from the existing models and a list of existing aspects that have the same meaning. The synthesized definition of each aspect was refined during the process of matching a set of URs to the common aspects. The refinement process includes expanding a definition by, for instance, adding more examples to clarify the boundary of each aspect. More details about URs (how they were collected and classified) are provided in the next section.

Step 4: Validate the common aspects. The mapping of existing aspects to the common ones was validated by an external annotator who has been working in NLP field for more than 15 years. The external annotator mapped all 226 aspects extracted from 33 models to the common set of usability aspects, with the option that the aspect cannot be mapped. Agreement with Step 3 (i.e. the researcher mapping) measured using Cohen's Kappa was 91%, which is considered a very good score [Alt90].

Usability Requirements

A set of URs are collected to refine the definition of the common usability aspects and to evaluate the ML classifiers. The procedure for collecting and validating these requirements consists of three main steps: collecting URs, annotating, and validating.

Step 1: Collecting the requirements. We gathered a set of requirements from different sources, including research papers, requirements specifications, student coursework, websites, and NFR datasets; other requirements were derived from existing guidelines such as content-accessibility guidelines [Con08] and web-usability guidelines [Tra16]. In total, 622 requirements are collected from different sources. Table 7.3 shows the distribution of the requirements according to the sources.

Step 2: Annotating the collected requirements. We manually analyzed and classified the collected requirements based on the common usability aspects identified in the previous section. All the collected requirements were classified manually with a primary focus on finding clear relationships between the common aspects and requirements. To achieve this, a set of requirements templates (i.e., sentence structures) were identified. A requirement template has been defined by Withall [Wit07] as "a fill-in-the-blanks definition for a requirement that is deemed to be typical of the type." For

Requirements Source	**Source Reference**	**No. of URs**
Available Dataset	PROMISE_exp	71
	Concordia	5
Research Paper	[AG17, LY98, CTL09, Que01, MCI$^+$08, BB02]	38
Lecture Resource (slides and course work)	[Bac11, Lis10, Chu, Bev00, Cre, est07, Mac13, Boc10]	38
	Collected from SWE students*	57
Books	[RR12, Som11, GDR04, Zie08]	12
Web Page	[(OP, Usa, Fir, Wah12, usa13, sta09, Kom20, Soe20, Bir19]	113
Available Requirements (Documents, Templates or User Manual)	[Tor09, GPM03, Rum12, Yev07, Com07, ITS, roc, CIC15]	76
Derived from Guidelines	[Con08, Tra16]	212
Total		622

* These requirements were collected from undergraduate and master student coursework in the department of software engineering in King Saud University and The university of Manchester, respectively.

Table 7.3: The sources and number of requirements collected for validating the model

example, the template for this UR: "90% of users will be able to make an airline reservation in less than five minutes" would be "A case includes <user_average> completed a task within <a given_time>." We used these templates to simplify the classification of the requirements, especially if the definition of aspect and the requirements' goal are not sufficient to distinguish the requirement class.

Step 3: Validate the annotations. After annotating all the requirements, we validated the annotations with a third-party annotator who has a master's degree in software engineering. The annotator was provided with the common aspects and their definitions and asked to classify the requirements based on the given aspects. The annotations provided by the external annotator were compared with ours; the inter-rater agreement, using Cohen's Kappa, was 64%, which is considered a good score ("Substantial agreement") [Alt90]. Disagreements (around 184 requirements) were reviewed again with the annotator until a consensus was reached (three requirements were removed due to lack of agreement).

7.2.4 Machine Learning Application

RQ2 requires development of ML models based on the output of RQ1. ML4RC (illustrated in Chapter 5) and SE4RC (illustrated in Chapter 6) are applied separately to classify URs. To evaluate these methods' effectiveness, we compare the classification

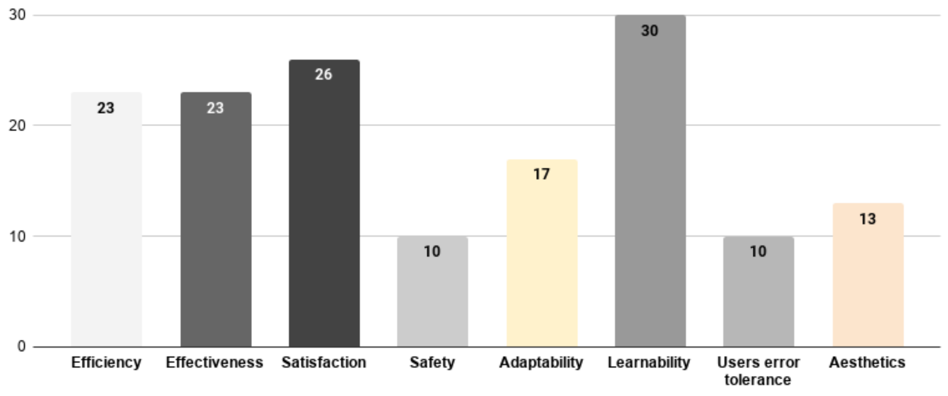


Figure 7.4: The frequency of each aspect in the 33 studied models

results obtained by applying these methods with a baseline classifier. We used SVM algorithm in these experiments as it showed the best performance in NFR classification (see Section 5.2).

The classifiers are evaluated using the same metrics (P, R, F1) with 10-fold cross-validation. We used nested 5-fold cross-validation to tune the parameters. Both macro and micro averages of each method are provided even though we focused on macro average as the base of comparison and discussion due to less bias towards majority classes. We also provide learning curves of $F1-score$ to show bias and variance of each method.

7.3 Results

This section reports and analyzes the results obtained from each objective/phase.

7.3.1 The Common Usability Aspects and their Relation with URs

We identified eight common usability aspects that appear with high frequency across the 33 studied usability models (see Figure 7.4 and Table C.6): *Efficiency*, *Effectiveness*, *Satisfaction*, *Safety*, *Learnability*, *Adaptability*, *Users' error tolerance*, and *Aesthetics*. The following paragraphs provide detailed definitions supported by an example.

1. Efficiency: This aspect defines what time and effort, resources, and financial costs required to perform specific tasks should be acceptable. This aspect is measured quantitatively and composed of three indicators: user efforts (efforts required to use a system such as training period), task time (the time taken to perform a task), and economic cost (incurred expenses, including the system itself, human resources, equipment, and consumables).

Example 1: Users must be able to generate a bank statement within three minutes.

2. Effectiveness: This aspect is concerned with system functionalities' ability to enable users to accomplish specific tasks and achieve desired results. This aspect can be either measured quantitatively by counting the number of tasks completed, amount of useful output, and errors rates, or qualitatively by questionnaires [HAS15]. It includes three indicators: completeness (completing users' tasks and achieve their goals), precision (performing tasks correctly with fewer errors), and usefulness (producing a desired result and useful output).

Example 2: Six out of eight (75%) users should be able to manage their accounts without making any errors.

3. Satisfaction: This describes the system's ability to provide a pleasant experience to users. This aspect is measured subjectively, either quantitatively or qualitatively, using a rating scale of satisfaction, questionnaires, number of positive and negative comments during use, and frequency of complaints. Its indicators include convenience (cognitive and physical satisfaction), likeability (emotional satisfaction), credibility (trust and confidence), and retention (continuous use).

Example 3: 70% of users must agree that they have a very pleasant experience while in the system.

4. Safety: This aspect is concerned with the system's ability to avoid damage and/or risks from using the system. There are three indicators for this aspect: user safety (physical safety, legal safeguarding and the safety of material assets), third-party safety (safety of individuals other than the user or equipment), and environmental safety (safety of the environment when the system is being used).

Example 4: The system should no longer operate in case of fire (e.g. an elevator)

5. Adaptability: The adoptability and adaptability by different users to their needs and preferences. Its indicators include personalization (anticipating users' preferences), customization (manageable content), universality (diverse backgrounds and cultures), portability (environment changes), and accessibility (availability to all users regardless of disabilities). Accessibility can be achieved through two criteria: alternatives and adapting to assistive technologies. Alternatives refer to providing equivalent options for the content (e.g., text and colors) and control elements (e.g., mouse and keyboard). Adapting to assistive technologies refers to making system content to be accessed and understood by users who use assistive technologies (e.g., data sequences).

Example 5: The system shall provide multi language support.

6. Learnability: A system's ability to enable users to understand and learn its functionalities in order to easily accomplish tasks the first time they encounter the software applications. Its indicators include clarity (perceivable by the mind and senses), familiarity (familiar design and functions), consistency, simplicity (straightforwardness of system features), helpfulness (user documentation and guidance), and navigability (easy to locate the system's features).

Example 6: Interface action and elements should be consistent.

7. Users error tolerance: The ability of a system to minimize the occurrence of users' errors (i.e., prevention of errors) and ensure that the user can deal with errors easily and correctly (i.e., users can recover from errors). The indicators of this concept are validating (the validity of the input data), restricting (reducing the possibility of inputting incorrect data or taking incorrect actions), reversibility (e.g., undo, redo and cancel), responsively (producing quick, meaningful warnings), and recoverability (handling an abnormal situation by, for example, providing appropriate instructions to correct the problems).

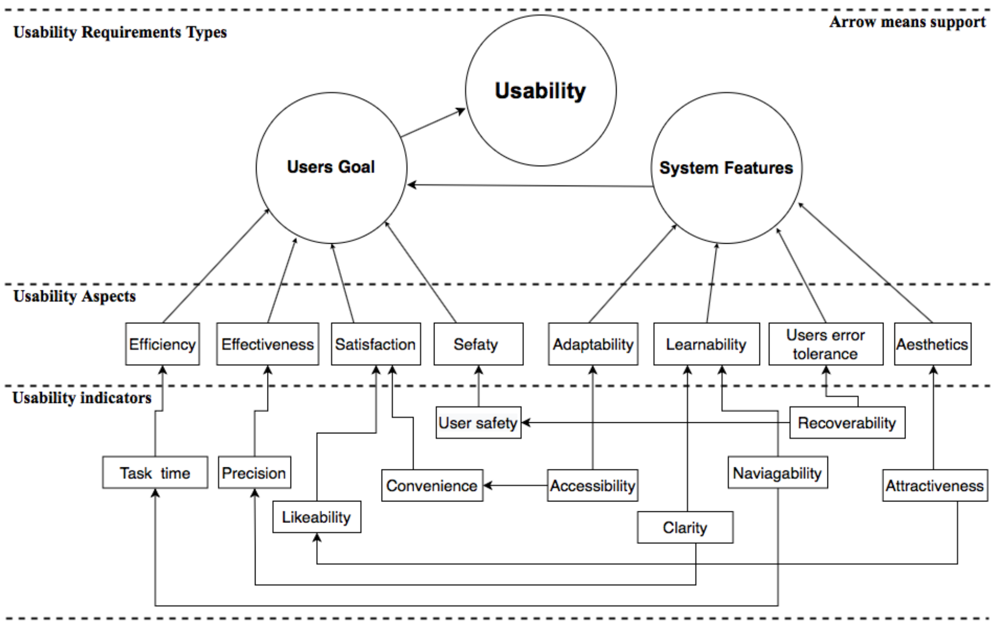


Figure 7.5: The relations between the usability aspects. This figure does not include all the indicators; however, it gives a general idea about how the relationships between the indicators and aspects, and among the indicators themselves, are created.

Example 7: The actions which cannot be undone should ask for confirmation.

8. Aesthetics: The ability of a system to be communicatively and aesthetically pleasing to users. The indicators of this aspect are attractiveness (e.g., the interface color), feedback (keeping a user informed through, for example, interaction feedback, progress feedback, and system status feedback), reachability (easy to locate the system itself through, for example, search engines), communicativeness (providing authoritative communication such as importing up-to-date and error-free information and offering active communication such as facilitating prompt and appropriate communication with users).

Example 8: The product shall comply with the corporate color scheme

Relationships between the common usability aspects. The most common usability aspects are grouped (see Figure 7.5) into two categories: 1) user goals (i.e., usability while using the system); and 2) system features (i.e., usability during system development). Each group is explained below and summarized in Table 7.4.

Users goal (Usability while using the system): These aspects focus on users (specifically on how users communicate with a system), as they include goals to be achieved by users. Examples of such goals include task time (“the ability to complete tasks within 5 minutes”), number of errors (“the ability to manage users’ accounts without making any errors”), satisfaction scale (“6 out of 10 users must rate the system as being either ‘very easy to use’ or ‘easy to use’”), risk impact and its probability (“avoiding eye strain caused by two hours of continuous use”), or specific contexts (such as user level [e.g., novice] and usage condition [e.g., no training] in the example, “Novice users must be able to perform tasks without training”). All the goals mentioned in these requirements belong to the first of the four aspects mentioned above (i.e., Efficiency, Effectiveness, Satisfaction, and Safety). We defined these requirements as usability goals, user-focused requirements, and non-functional URs (as they describe *how* a user communicates with system). These requirements often begin with users and measure the communication between users and a system in term of satisfaction and performance. Thus, these requirements must be tested with the users through, for example, usability testing. The common templates used for this type of requirement are shown in Table 7.5

System features (Usability during system development): These aspects describe how a system should be designed (e.g., interface characteristics) to achieve user goals. The requirements belonging to this group are more about a system’s physical properties (system features), including specifying what system elements should include and how they should appear, behave, and interact. Examples of these requirements include “Error messages should explain how to recover from the error,” “Main system functions shall be available from the home page,” and “It is possible to accomplish any given task with just the keyboard, without the need to use the mouse.” These characteristics belong to the last four aspects discussed above: Adaptability, Learnability, Users error tolerance, and Aesthetic. They can be identified and tested by a development team through prototypes. The common templates used for these requirements are shown in Table 7.5

7.3.2 The Effectiveness of Detecting and Classifying URs

We used URs dataset described in Section 7.2.3 to build and test ML models. Table 7.6 provides a summary of the URs dataset, exhibiting the problems of multiclass imbalances, short text, and high dimensionality. The following section will show the results of applying ML4RC and SE4RC. After that, we will show the impact of applying each technique of ML4RC in classifying URs.

	User Goals	System Features
Meaning	Performance and Satisfaction	Easy-to-use interface
Evaluating	By usability testing	By testing a paper or machine prototype
Identifying	Users and the development team must all participate in specifying these requirements	The development team defines these requirements for a system being developed, and users can be involved if they are available.
Requirements Examples	The payment process must be completed within three minutes. The number of mistakes noted by the students shall be decreased by 50% in the first year.	The interface actions and elements should be consistent. Error messages should explain how to recover from the error.

Table 7.4: User goals versus system features

Evaluation of the Proposed Methods (ML4RC and SE4RC)

Classification Performance. Table 7.7 shows the results of applying the baseline, ML4RC and SE4RC methods in classifying URs. It is clearly seen from the table that SE4RC has the best performance, followed by the baseline classifier, and finally ML4RC. These results are similar to those obtained by applying these method in NFRs classification, confirming the effectiveness of feature extension in requirements classification. To find which technique(s) of ML4RC negatively affected the accuracy of classifying URs, we evaluate each technique separately in the next section.

For individual requirement categories, SE4RC outperforms all the other methods in the three majority classes (i.e., Learnability, Adaptability, and Efficiency) and one minor class (User error tolerance). SE4RC expands short requirements with the most similar requirement. The highly similar requirements within one class are more likely to be retrieved from large classes (those with a high number of instances), and the classes contain distinct features (e.g., User error tolerance). Therefore, these classes are improved more with SE4RC; similar observations have been noticed in Chapter 6. On the other hand, ML4RC outperforms the other methods in three out of five minor classes: Satisfaction, Effectiveness, and Safety (the most minor class), indicating the effectiveness of ML4RC in classifying minor classes. ML4RC applied both feature selection and dataset decomposition, which help improve the performance of minority classes, especially those with distinctive features (e.g. Safety). This observation has also been noticed in Chapter 5. Statistically, the Wilcoxon signed-rank test shows that the baseline has an improvement over the ML4RC method on Efficiency (P and F1), Learnability (R), Adaptability (R and F), and User Errors Tolerance (P). In contrast, there is no significant difference between SE4RC and the baseline method.

Aspect type	**Common Templates**	**Requirements Examples**
Goals (User-related requirements)	A case includes completing a task with a given_time	Callers and supervisors must be able to accomplish any system task within 2 minutes.
	A case includes skill_level user completed a task with a given_time	Novice users shall perform tasks Q and R in 15 minutes.
	A case includes that user average agree with a specific-premise	80% of a statistically valid sample of users shall continue to use it after an initial 4-week trail usage period.
	A case includes user_average completed a task with spesfic_num_errors	Six out of eight users should be able to manage their accounts without making any errors.
User interface (System-related requirements)	A system_part must contain specific_constraint	The error message must be displayed on the next line of the error resources to allow users to find and fix the errors immediately.
	A system_part must follow specific_criteria	All buttons in the system will adhere to established button convention (link to established button convention regarding size, naming, position, etc.)
	A system_part must contain specific_content	The system must be able to support users by providing help messages and guidance.
	A system_part must provide a specific_service	The application must be able to check inputted data before submission.

Table 7.5: Common Usability Requirements Templates

Category	**#Reqs.**	**Percent**	**Avg Len.**	**#Feat.**
Learnability	152	24.4	17.27	845
Adaptability	110	17.7	18.61	698
Efficiency	105	16.9	20.74	531
Satisfaction	72	11.6	30.65	437
Effectiveness	59	9.5	28.86	435
Users error tolerance	56	9.0	22.19	465
Aesthetics	51	8.2	15.00	360
Safety	17	2.7	20.52	159
Total	**622**	**100.0**		

Table 7.6: Summary of the URs dataset

Category	Baseline Classifier			ML4RC Classifier			SE4RC Classifier		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
Learnability	0.63 ±0.11	0.82 ±0.13	0.71 ±0.10	0.68 ±0.21	0.49* ±0.24	0.54 ±0.20	0.65 ±0.11	0.83 ±0.14	0.73 ±0.11
Adaptability	0.65 ±0.16	0.73 ±0.11	0.68 ±0.12	0.50 ±0.26	0.44* ±0.21	0.46* ±0.21	0.68 ±0.16	0.74 ±0.09	0.70 ±0.12
Efficiency	0.83 ±0.12	0.77 ±0.17	0.79 ±0.13	0.58* ±0.19	0.56 ±0.16	0.55* ±0.14	0.85 ±0.13	0.78 ±0.16	0.80 ±0.12
Satisfaction	0.81 ±0.14	0.74 ±0.18	0.76 ±0.14	0.83 ±0.11	0.78 ±0.17	0.79 ±0.11	0.80 ±0.13	0.74 ±0.18	0.75 ±0.13
Effectiveness	0.74 ±0.27	0.53 ±0.26	0.57 ±0.23	0.57 ±0.13	0.72 ±0.16	0.63 ±0.14	0.69 ±0.25	0.55 ±0.28	0.56 ±0.21
Users error tolerance	0.82 ±0.22	0.51 ±0.24	0.59 ±0.22	0.62* ±0.22	0.44 ±0.21	0.46 ±0.18	0.84 ±0.22	0.53 ±0.22	0.62 ±0.21
Aesthetics	0.72 ±0.26	0.46 ±0.16	0.54 ±0.16	0.50 ±0.45	0.40 ±0.37	0.43 ±0.39	0.72 ±0.26	0.46 ±0.16	0.54 ±0.16
Safety	0.50 ±0.50	0.40 ±0.44	0.43 ±0.43	0.83 ±0.16	0.72 ±0.24	0.73 ±0.19	0.50 ±0.50	0.40 ±0.44	0.43 ±0.45
Mic.	**0.69** ±0.01	**0.69** ±0.01	**0.69** ±0.01	**0.63** ±0.09	**0.63** ±0.01	**0.63** ±0.01	**0.70** ±0.01	**0.70** ±0.01	**0.70** ±0.01
Mac.	**0.71** ±0.01	**0.62** ±0.01	**0.63** ±0.01	**0.64** ±0.08	**0.57** ±0.10	**0.58** ±0.10	**0.72** ±0.01	**0.63** ±0.01	**0.64** ±0.01
Ex. Time	71 seconds			70 seconds			3 minutes		

* Indicates a significant statistical difference between the baseline and another classifier with p<0.05. Wilcoxon signed-rank test has been applied.

Table 7.7: 10-Fold Cross-Validated Test Results for the baseline classifier, ML4RC and SE4RC in classifying usability requirements

In terms of execution time, ML4RC is the fastest method, taking 48 seconds to run, followed by the baseline with 70, and finally the SE4RC method which took 3 minutes. The execution time is based on the time spent on 10-fold CV without considering the time used for tuning the parameter, as some classifiers have more parameters than others.

Learning Curves. Figure 7.6 shows that the training curves of the baseline and SE4RC methods are similar, both starting with F1 in the range F > 0.95 and moving slightly upward to F = 1.00 (i.e., high bias). ML4RC also started with a high F1-score value (F1 = 0.93) and moved slightly up and down until it stopped at F = 0.95. On the other hand, all the testing curves of all the methods start with a low value ($F \leq 0.20$) and move upwards where the curve of SE4RC and baseline are relatively similar; however, SE4RC stopped is a slightly higher value. ML4RC testing curve shows the worst improvement by adding more examples when compared with the curve of SE4RC and the baseline.

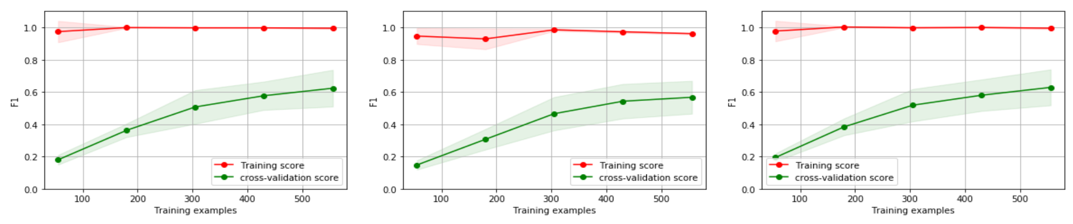
Evaluation of ML4RC Techniques

Classification Performance. Table 7.8 shows that the average performance of the FE classifier is the highest among the other classifiers, followed by the DD and baseline, and finally the FS classifier. The DD classifier achieves higher recall but lower precision than the baseline, and, consequently, the same F1-score value. These results are similar to those mentioned in Chapter 5 (see Table 5.9). However, in NFRs classification, FS shows comparable performance to the baseline, while in URs classification, it shows lower performance than baseline. This confirms that the features selected using the semantic role-based feature selection technique are not enough to distinguish requirements classes.

For individual requirement categories, apart from the adaptability and satisfaction, the FE classifier achieves the highest classification results for all the classes. In contrast, FS shows the lowest performance among all classifiers, apart from Aesthetic. DD outperform FS and FE classifiers on Satisfaction, and the baseline on Efficiency, User error tolerance, Safety, and Satisfaction. The overlapping among the class in the *mix* set (i.e., Learnability, Adaptability, and Efficiency) and *min* set (e.g., Effectiveness and Aesthetic) are a key reason for low performance achieved in these classes. Table 7.9 shows the most frequent feature for each class. The table clearly shows the overlapping between Efficiency and Effectiveness as well as Learnability with Adaptability and Aesthetic. As Aesthetic has a lower number of examples, it shows the worst performance among all classes using DD. Statistically, the Wilcoxon signed-rank test only shows that the DD method shows a significant decrease than the baseline in the P and F1 of Aesthetics.

In terms of execution time, FS is the fastest method, taking 59 seconds to run in terms of execution time, followed by the DD classifier (61 seconds), and baseline (71 seconds). Finally, the FE classifier took 1.41 minutes to run. The execution time is based on the time spent on the 10-fold CV without considering the time used for tuning the parameter as some classifiers have more parameters than others.

Learning Curves. Figure 7.7 shows that the learning curves of the baseline, DD, and FS classifier start with F1 in the range $1.0 < F > 0.90$ and move slightly upward to $F = 1.0$, where DD is slightly lower than the others (lower bias). However, the learning curve of FE starts with $F = 100$ and stay largely unchanged (high bias). On the other hand, the testing curve starts with a low value in all the curves ($F < 0.20$) and moves upwards, where the curve of FE is more prominently going upwards (lower variance).



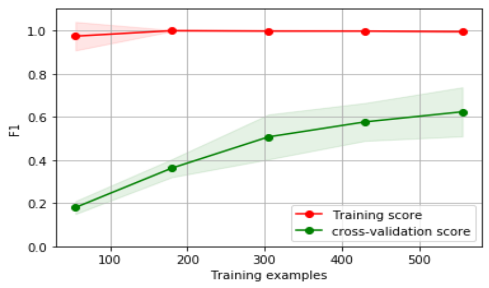
(a) Baseline (b) ML4RC Classifier (c) SE4RC

Figure 7.6: Learning curves of the baseline, ML4RC and SE4RC methods in usability requirements classification based on 10-fold cross-validation.

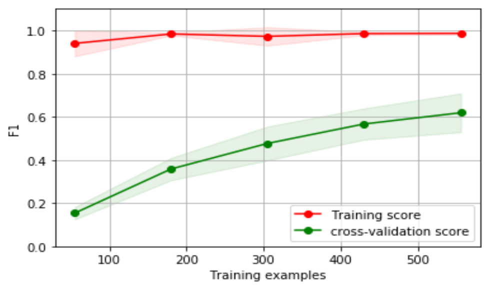
Category	Baseline Classifier			DD Classifier			FS Classifier			FE Classifier		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
Learnability	0.63 ±0.11	0.82 ±0.13	0.71 ±0.10	0.62 ±0.13	0.73 ±0.15	0.67 ±0.13	0.58 ±0.11	0.77 ±0.17	0.66 ±0.12	0.67 ±0.12	0.78 ±0.14	0.71 ±0.12
Adaptability	0.65 ±0.16	0.73 ±0.11	0.68 ±0.12	0.65 ±0.11	0.66 ±0.11	0.65 ±0.09	0.61 ±0.10	0.66 ±0.20	0.61 ±0.11	0.64 ±0.16	0.72 ±0.10	0.67 ±0.11
Efficiency	0.83 ±0.12	0.77 ±0.17	0.79 ±0.13	0.85 ±0.12	0.84 ±0.09	0.83 ±0.05	0.82 ±0.12	0.77 ±0.18	0.78 ±0.13	0.83 ±0.14	0.78 ±0.15	0.80 ±0.13
Satisfaction	0.81 ±0.14	0.74 ±0.18	0.76 ±0.14	0.86 ±0.16	0.76 ±0.18	0.79 ±0.16	0.86 ±0.15	0.68 ±0.21	0.73 ±0.15	0.86* ±0.14	0.75 ±0.19	0.78 ±0.13
Effectiveness	0.74 ±0.27	0.53 ±0.26	0.57 ±0.23	0.58 ±0.28	0.54 ±0.26	0.55 ±0.24	0.70 ±0.25	0.51 ±0.26	0.54 ±0.21	0.74 ±0.23	0.58 ±0.24	0.59 ±0.17
Users error tolerance	0.82 ±0.22	0.51 ±0.24	0.59 ±0.22	0.72 ±0.17	0.63 ±0.24	0.64 ±0.18	0.68 ±0.28	0.56 ±0.27	0.59 ±0.24	0.81 ±0.15	0.67 ±0.21	0.70 ±0.16
Aesthetics	0.72 ±0.26	0.46 ±0.16	0.54 ±0.16	0.39* ±0.33	0.34 ±0.27	0.35* ±0.27	0.69 ±0.23	0.46 ±0.19	0.53 ±0.18	0.71 ±0.33	0.50 ±0.24	0.57 ±0.26
Safety	0.50 ±0.50	0.40 ±0.44	0.43 ±0.43	0.60 ±0.49	0.50 ±0.45	0.53 ±0.45	0.45 ±0.47	0.35 ±0.39	0.38 ±0.41	0.70 ±0.46	0.55 ±0.42	0.60 ±0.42
Mic.	**0.69** ±0.01	**0.69** ±0.01	**0.69** ±0.01	**0.68** ±0.09	**0.68** ±0.01	**0.68** ±0.01	**0.66** ±0.09	**0.66** ±0.09	**0.66** ±0.09	**0.71** ±0.01	**0.71** ±0.01	**0.71** ±0.01
Mac.	**0.71** ±0.01	**0.62** ±0.01	**0.63** ±0.01	**0.66** ±0.11	**0.63** ±0.09	**0.63** ±0.10	**0.67** ±0.08	**0.59** ±0.09	**0.60** ±0.09	**0.75** ±0.09	**0.67** ±0.10	**0.68** ±0.10
Ex. Time	71 seconds			61 seconds			59 seconds			1.41 minutes		

* Indicates a significant statistical difference between the baseline and another classifier with p<0.05. Wilcoxon signed-rank test has been applied.

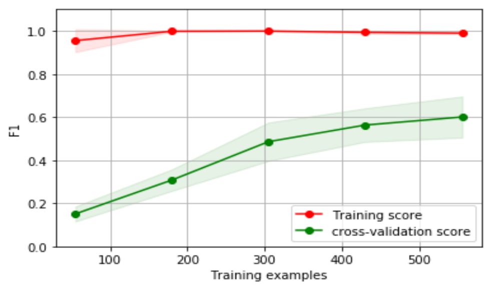
Table 7.8: 10-Fold Cross-Validated Test Results for the baseline classifier and the classifiers applying the techniques of ML4RC separately (Dataset Decomposition, Feature Selection, and Feature extension)



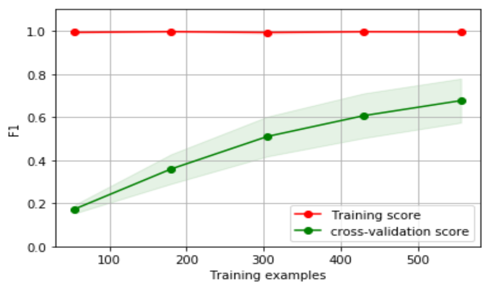
(a) Baseline



(b) DD Classifier



(c) FS Classifier



(d) FE Classifier

Figure 7.7: Learning Curves of the baseline, DD, FS, and FE classifiers Based on 10-Fold Cross-Validation

Learnability	**Adaptability**	**Efficiency**	**Satisfaction**	**Effectiveness**	**Users error tolerance**	**Aesthetics**	**Safety**
page	able	able	Application	perform	error	website	automated
provide	accessible	minutes	easy	error	message	color	per
easy	screen	training	sample	tasks	data	message	average
help	text	tasks	statistically	complete	enter	interface	protect
interface	language	successfully	valid	sample	actions	appear	injure
button	data	perform	specifically	failures	validate	attractive	case
functions	form	complete	scale	enable	require	content	open
navigation	reader	average	rate	effective	input	used	lost
screen	support	seconds	neutral	easy	confirmation	data	moving
consistent	interface	learn	following	increase	list	information	history

Table 7.9: Top 10 most frequent features for each usability class in UR_databaset

7.4 Lessons Learned

In this section, we discuss some lessons learned from this study and link them with the literature. We organize this section based on the research questions as follows: the common set of usability aspects (RQ1) and using ML to classify URs (RQ2).

7.4.1 Usability Aspects According to Usability Requirements

Many models have been proposed to classify the usability concept into a set of aspects (also known as goals and attributes). Very few of these models considered the application of these aspects in requirement analysis and classification. Among the 33 models we studied, only one model, proposed by Hasan and Al-Sarayreh [HAS15], uses the Softgoal Interdependency Graph (SIG) which is used to visualize non-functional goals to represent usability aspects. Their SIG model divides usability aspects based on usability evaluation time into two main categories: usability during system development (e.g., accessibility and learnability) and usability while using the system (e.g., effectiveness, efficiency and satisfaction). However, their model provides neither sufficient justification for this classification, nor differences between requirements under each category. This increases the difficulty in understanding how this model could work correctly in the requirements analysis processes.

In the URs literature, many studies address usability requirements during the system development process with a clear focus on requirement elicitation and evaluation [OPCF$^+$13, GYK04, MT09, BCMA02]. However, very few studies focused on analyzing and classifying URs (see Section 7.1.3). An example of these studies is the one conducted by Juristo et al. [JMSS07] who suggested classifying usability requirements into FR and NFR. Juristo et al. defined non-functional URs as the requirements that specify the user level of effectiveness, efficiency, or satisfaction that a system should achieve. These requirements are said to be used as a criterion at the evaluation stage. On the other hand, they defined functional URs as usability features that have an impact on software system functionality (e.g., undo and cancel). Such requirements cannot be well defined by developers or users, as argued, and this increases the need for human-computer interaction (HCI) knowledge (i.e., HCI experts). Therefore, they proposed an alternative solution for HCI experts through the use of HCI guidelines. They extract a set of functional usability features among HCI sources and classify them as the following: feedback, undo/cancel, user input error prevention/correction, wizard, user profile, help, and common aggregation. They evaluated the benefits of their approach in developing a software system, finding that their approach reduces ambiguous usability details early and helps developers easily apply usability features, thereby enhancing the usability of the final system.

Our classification of usability aspects is similar to the aforementioned models. For example, we group the aspect into user goals and system features, similar to Hasan and Al-Sarayreh's [HAS15] model. Each group represents different requirements; user goals are for users while user interface attributes are for developers, similar to Juristo et al. [JMSS07]. However, compared to Juristo et al. [JMSS07], we add a further step: grouping usability requirements based on a set of usability aspects (goals) and linking the aspects with each other to show how they can be achieved and measured in practice. Moreover, we used these aspects in requirement classification, showing the relation between aspects and how they can impact each other (achieved by others). In this way, a requirement analyst could clearly and quickly analyze and validate URs by, for example, identifying what usability aspect is satisfied or missed, and how a particular requirement can support system usability (i.e., through any usability aspect). In comparison with Hasan and Al-Sarayreh's [HAS15], we provide more details about how usability aspects can be applied in requirements specification and evaluation. We also derive the aspects and their interrelationships from collected data (usability models and URs) and use different aspects (quantity and definitions) than those used in their model.

The relation between usability aspects derived from the collected requirements is similar to the layered models proposed by Welie et al. [VWVDVE99]. Welie et al.'s model breaks down the usability concept into usage indicators and means to achieve good usability in practice. As shown in Figure 7.8, their model consists of three main levels: the first contains an abstract definition of the usability aspects of efficiency, effectiveness, and satisfaction. The second consists of the usage indicators "indicator can be observed in practice when users are at work", such as performance speed. The third contains the means "cannot be observed in user test and they are not goals by themselves whereas indicators are observable goals", such as consistency. However, their model is not complete; some elements are duplicated (e.g., satisfaction) and others lack clear explanations as to how they can be achieved or evaluated (e.g., learnability). In addition, they haven't considered the requirements design and evaluation in developing or defining this model.

Our application in this chapter, on the other hand, focuses on URs and built based on a deep understanding and analysis of usability URs. Although we paid attention

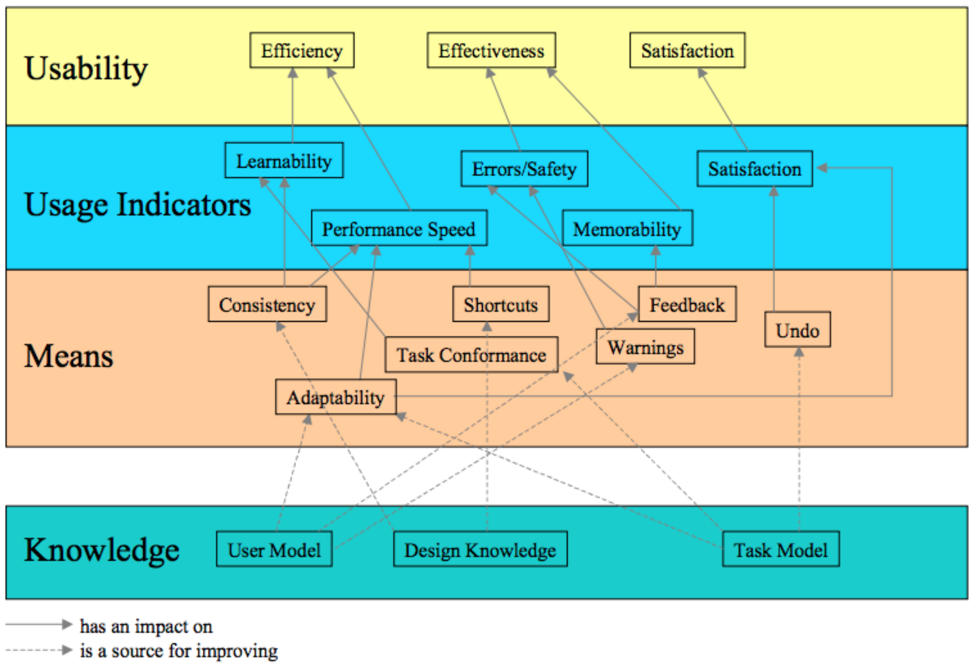


Figure 7.8: The layered model proposed by Welie et al [VWVDVE99]

to usability aspects that commonly appear in the related models to enhance the generalizability, these aspects might not be enough to be used in some specific application domains. In addition, to define the relationships between requirements, we classify real-life examples of URs by taking into account the requirements templates, besides the semantic meaning of requirements. Although this gives us a clear distinction between requirement types (i.e., user goals or system features), it poses a difficulty in automatically classifying URs, especially when the semantic meaning and requirement template are inconsistent for predicting the requirements classes. For example, this requirement: "The average buyer shall be able to learn to find relevant items within 3 minutes" is classified as efficiency, even though it seems to be semantically related to learnability. This requirement belongs to the temple "A case includes completing a task with a given_time". Therefore, it will be specified and tested with the end-user. Unfortunately, this is a common issue in automatically classifying NFRs as we discussed in Chapter 6 (see Section 6.6.1), thus, such issues are difficult to overcome.

Overlapping is also a common problem in NFR. We notice overlapping among usability concepts and requirement classes which increases the challenges in extracting the common aspects and classifying requirements. For example, in the usability models we studied, we noticed that there is overlapping between the aspects themselves and the aspect with other NFRs such as error tolerance and security. Moreover, there is overlapping between the URs themself and other non-functional requirements such as performance, error tolerance, and operability. For example, both “time taken by a user to perform a task” and “time taken by a system to respond" are classified as UR. However, they are different. The first is related to usability as it is about communication between users and a system while the second is related to system performance. In addition, we notice that some aspects that frequently appeared in existing models rarely appear within URs such as safety. This clearly indicates that addressing usability in RE (i.e., URs) is still an immature area and much more research is needed.

7.4.2 Using ML to Classify URs

According to the review reported in Chapter 3, ML algorithms are not applied in analyzing and classifying usability requirements, indicating the low effort spent by the software engineering community to handle usability early in the RE phase. However, a few methods are proposed in the HCI community [HS13, BG17]. These attempts applied SVM to classify user reviews according to various usability or user experience dimensions provided in existing usability models (e.g., BEVAN [Bev08]). The results of applying these methods showed the effectiveness of ML in classifying user reviews into major usability classes. However, minor classes have very low results (P= 0.00, R =0.00, F=0.00). It is argued that the low performance of these classes is due to the low number of samples. However, the lack of diversity among the classes could also be a key reason for the low performance of such classes. For example, Satisfaction, Enjoyment, Fun and Frustration, and Motivation are separate classes. The main difference between our method and these methods is, besides the differences in classes, that they used user reviews which contain more distinct features than requirements. For example, "favorite", "disappoint", "fun", "nice", "cool" have been identified as common terms used for the satisfaction class. These features are distinct and intuitive; however, they appear less frequently in requirements. The other difference between our model and these models is that they classify the reviews without considering the system development process. We classified requirements based on the usability aspect that was organized based on a system development process.

In this chapter, the result of applying SVM to classify URs according to common usability aspects is promising ($F = 0.63$). However, more effort is required to handle the similarity among the aspects (e.g, Effectiveness with Efficiency, Aesthetics with Learnability and Adaptability). To illustrate, Effectiveness and Efficiency have very common cases. For example, “A case including user_average completed task with a given_time" could be categorized under efficiency (i.e., focusing on the time taken to complete the task) or effectiveness (i.e., the ability to complete a task of user_average). However, as Efficacy is a major class, it reaches a higher classification performance than effectiveness. Therefore, handling overlapping among URs can be through either adding more examples to the minor classes or considering multi-label classification where each requirement can be classified into more than one category. We will leave this to future work.

The application of the proposed method ML4RC and SE4RC in URs classification made changes over the baseline classifier. Although ML4RC could not outperform the baseline, its techniques (Dataset Decomposition and Feature Extension) have made an improvement over the baseline. For example, as shown in section 7.3.2, the DD classifier improves the recall of SVM but decreases precision. Moreover, it is more efficient than the baseline. For the FS classifier, the efficiency of the SVM classifier is enhanced (less execution time). However, it has not improved the classification results. Finally, extending requirements with synonyms extracted from WordNet significantly improves the classification results for all classes, except those with very few samples; not many similar features belong to such classes. On the other hand, expanding requirement with a similar requirement (i.e., SE4RC) shows a slight improvement over the baseline. This technique's big improvement goes to the majority classes (more likely to find similar requirement within the same class), and minor with district features (more like to retrieve similar requirements with the same class). However, ML4RC is more costly in execution time than the baseline.

Most of the findings reported on applying ML4RC and SE4RC in URs classification are similar to those mentioned in NFR classification. This increases the generalizability of these findings on other requirements classification tasks. However, the overall average of URs is better than those achieved by NFRs ($F1$ of baseline in NFR = 0.47, and in URs = 0.63). The simplicity of the classification task (less number of classes) is one of the main reasons. Moreover, the deep analyzing and understanding of the classification problem (i.e., dataset) is another important factor. In URs classification, we analyse the usability concept and the interrelationships between requirements deeply to reduce class ambiguity and understand the classification task. This understanding of the problem can improve the quality of a dataset, which, in its turn, enhances classification performance [KBLK10].

7.5 Threats to Validity

In this section, we discuss the threats to the validity of this chapter, as well as our mitigation strategies.

Construct validity. A potential threat to construct validity is related to incorrectly understanding usability. To mitigate this threat, we conducted a systematic review of the definition of usability to identify common usability characteristics. We validated our understanding with an external annotator by mapping a set of existing characteristics to the common ones. Another threat is personal bias in the manual classification of URs. To minimize this threat, two annotators classified the requirements separately and then discussed and resolved any disagreements on the classification results. The third threat is insufficient URs to draw reasonable conclusions. To reduce this threat, we extracted URs from different sources and domains.

Internal validity. There are minimal threats to internal validity as a careful analysis was performed while choosing the independent variables used by ML, and in identifying and grouping usability attributes. For example, each task for identifying and grouping usability attributes was validated with an external annotator, before moving to the next task. To mitigate the overfitting threat in ML models, we applied K-fold cross-validation and showed learning curves.

External validity. A potential threat to external validity is the lack of generalizability of our findings. This chapter is designed to mitigate this threat since it demonstrates the application of the proposed methods to a different classification task. This task has a different dataset, which is collected from different sources and labeled with different classes. Moreover, we used the K-fold cross-validation procedure to train and validate all the classifiers.

7.6 Summary

This chapter reports a case study to evaluate the proposed ML methods (ML4RC and SE4RC) in classifying UR into other classes. The case study consists of two main phases: 1) identifying the usability aspect used to guide ML models, and 2) evaluating the effectiveness of using ML to automatically classify URs. To identify usability aspects, we conducted a snowball-based review to extract the common aspects from 33 existing usability models. These aspects are refined through the process of classifying a set of usability requirements collected from various sources. To evaluate ML models in UR classification, we use the pre-classified requirements (622 requirements) to build and test ML models. We apply the proposed methods (SE4RC and ML4RC) and compare their performance with a baseline method. In this way, we compare the effectiveness of addressing the three problems (i.e., high dimensionality, class imbalance, and short text) in classing URs.

The results shows that there are eight common usability aspects: *Efficiency*, *Effectiveness*, *Satisfaction*, *Safety*, *Learnability*, *Adaptability*, *Users error tolerance*, and *Aesthetics*. These aspects are grouped into two categories: user goal and system features; requirements belonging to each group have different specification and evaluation methods. Moreover, applying ML to classify URs into these aspects shows encouraging results. However, more efforts are needed to improve classification accuracy. These efforts could comprise adding more instances to minor classes or applying multi-label classification. We will leave that to future work.

Applying ML4RC and SE4RC shows similar classification performance to those obtained with NFRs classification (in Chapter 5 and 6). This confirms the previous chapter's findings, including the effectiveness of feature extension and data decomposition in requirement classification. In the next chapter, we conclude the thesis and discuss future directions.

Chapter 8

Conclusion and Future Direction

"There is no end to learning, but
there are many beginnings."

Tim Johnson

The work presented in this thesis focuses on using ML to automatically classify requirements early in the RE phase, where most of the requirements are written in NL documents. Specifically, it investigates the impact of three classification problems (high dimensionality, class imbalanced, and short text classification) in detecting and classifying textual NFRs using standard ML algorithms. This includes proposing two solutions (ML4RC in Chapter 5, SE4RC in Chapter 6), which were evaluated in two different classification tasks (identifying and classifying NFRs in Chapters 5 and 6, and classifying URs in Chapter 7). This chapter concludes the research by reiterating contributions (Section 8.1), summarising the findings (Section 8.2), and highlighting future works 8.3.

8.1 Contributions

The key contributions of this research are summarized as follows:

- A systematic review provided *a comprehensive understanding of ML algorithms' use in NFR classification* (what and how). This review can be used by other researchers as a roadmap for developing this area by, for example, exploring new solutions or improving existing ones.
- *A new ML classifier (ML4RC) to handle the three classification problems.* This

method consists of three new key techniques, data decomposition, semantic role-based feature selection, and features extension. We conducted evaluation studies showing the ML4RC is an improvement over related methods, and its key techniques outperform existing solutions and a baseline method.

- *Expanding short requirements with similar requirements (not words) to handle short-text classification in the SE4RC method.* This expansion performs competitively well with respect to related work and a baseline method in different classification tasks.
- *An empirical comparison of several similar semantic methods for measuring the similarity of NFR requirements*, illustrating the difficulties of measuring NFR similarity and the effectiveness of corpus-based measures in measuring NFR similarity.
- *Identifying common usability aspects* based on conducting a systematic review of usability definitions and classification models.
- *Application of ML in UR classification* to investigate the effectiveness of proposed solutions (ML4RC and SE4RC) in a different classification task. The results confirmed previous findings, suggesting the effectiveness of the proposed solutions in various text classification tasks.

8.2 Main Findings

This research contains different empirical studies, critical assessment, and systematic reviews, making many findings and providing new insights. This section reviews the most salient findings, organized by the key thesis chapters: chapter 3,4,5, 6 and 7.

Chapter 3. This chapter, which presents a systematic review on using ML in NFRs classification, indicates many findings, including the following:

- Using ML in NFRs classification has received increasing attention in the last years.
- Supervised learning works better than unsupervised in NFRs classification, where SVM is the most frequently used algorithm.
- ML is effective in NFRs classification, with an accuracy of more than 70%.

- The most addressed non-functional requirements are security requirements.
- Cross-validation technique is more frequently used to evaluate NFRs classifier, and precision, recall, F1-score are the most used measures.
- Lack of sufficient reporting of ML methods and findings.
- Lack of clarification and justifications on the use of specific evaluation methods.
- A few shared training datasets—most of them are relatively small.
- Lack of standard definition, classification, and representation of NFRs
- The difficulty of selecting relevant features and classifying NFRs, due to many shared contexts and high similarity among different non-functional classes.

Chapter 4, which provides an overview of the three problems and analyzes the existing solutions in the context of NFRs classification, includes the following findings:

- High dimensionality, class imbalance, and short text classification problems are critical for supervised ML classifiers' effectiveness and generalizability.
- Insufficient solutions to handle these problems in NFRs classification.
- High dimensionality is the most addressed problem, followed by class imbalance, while short text classification problem is clearly handled by one study.
- Most feature selection methods applied to handle high dimensionality in NFRs are statistical-based methods.
- Most of the solutions applied for an imbalance class problem in NFRs classification are data-level solutions.
- Feature extension is the common solution to address short text classification, and it is applied once in NFRs classification.

Chapter 5, which presents the first solution (ML4RC), includes the following findings:

- Data decomposition technique can improve the performance of NFRs classifiers, outperforming oversampling and undersampling techniques.

- Feature selection methods cannot improve the overall performance of NFRs classifiers; however, they enhance the generalizability of the classifiers.
- Linguistics feature selection methods outperform statistical-based methods with relatively small datasets in requirements classification.
- Feature extension is very promising in requirements classification, outperforming baseline methods; however, it less efficient (i.e., it took more time).
- By applying each technique separately, feature extension shows the highest performance, while feature selection is the lowest.
- Addressing all the problems by applying ML4RC is less effective than separately performing dataset decomposition or feature extension techniques. Feature selection is the reason for the lower performance of ML4RC.

Chapter 6, which presents the second method (SE4RC), includes the following findings:

- Corpus-based methods are better than knowledge-based methods in determining the similarity between two requirements.
- It is difficult to determine requirement similarity on an automated basie due to the short length of requirements, shared context in NFRs (e.g., fit-criteria), and containing both implicit and explicit information.
- Expanding requirements with additional requirements shows an improvement in NFRs classification. However, it is less effective than word expansion due to the difficulty in measuring requirements similarity.

Chapter 7: which presents the application of the proposed methods in URs classification, includes the following findings:

- Lack of attempts in classifying URs, while great variant in defining usability concept.
- URs can be functional (system attributes) or non-functional (user goals).
- There are 8 common usability aspect, namely *Efficiency*, *Effectiveness*, *Satisfaction*, *Safety*, *Learnability*, *Adaptability*, *Users error tolerance*, and *Aesthetics*.

- Applying the proposed methods with URs shows the same performance as NFRs.
- The overall performance of URs classification is higher than NFRs classification. The reasons behind this observation are likely to be the simplicity (8 class against 12 classes), and a comprehensive understanding of the classification task before applying ML (e.g., analyzing requirements templates and usability concepts).

8.3 Future Work

There are many possible research directions for the work presented in the thesis. These directions can be broadly grouped into two categories: 1) empirical experiment and analysis, which aims to improve the present work performance, and 2) extrinsic evaluation, aiming to use or evaluate the proposed method in supporting RE tasks.

8.3.1 More Empirical Experiments and Analysis

Considering Class Content when Decomposing Data. The data decomposition method shows encouraging results in handling class imbalance problems. However, this method divides the dataset only based on class distribution without considering the logical relation between classes or features. Consequently, the performance of the classes that have less distinct features is low in the first layer and all subsequent layers. Thus, in our future work, we investigate how to improve the performance of the first layer by either 1) considering both class content and class distribution in decomposing a training dataset or 2) defining different classification setting to improve the process of distinguishing classes of the first level (e.g., apply a diffident ML algorithm or create a rule-based technique).

Beyond textual features. Feature selection methods, either statistical-based or linguistic-based methods, cannot improve the classifier performance. In addition to the short length of these requirements, words often associated with a particular NFR tend to be scattered over the other NFRs (e.g., security and performance e [CHSZS07], performance, and scalability [CHSZS07]. The literature shows that using keywords is not enough (based on the experiment of Cleland-Huang et al. [CHSZS07]). ML4RC shows that detailed structure analysis is prone to error due to removing relative features. Multi-words feature (e.g., 2-gram, 3-gram, and multi-word expressions) have also been shown to be ineffective in NFR classification [ZYWS11]. Therefore, in our future work, we plan to investigate the use of different types of features in addition to textual features. Potential directions for this line of research include: 1) using several learners where each one is trained using a specific type of features, 2) combining different types of features with training a single classifier.

More Types and methods in Feature Extension. Feature extension methods show promising results in NFRs classification. Thus, further investigation on the use of feature extension is needed. The investigation will include using different sources and methods of extended features. The sources could be those describing non-functional classes, such as standards and glossary. The measures could include using different similarity methods and/or background knowledge.

Different Classification Techniques. This research applied the widely-used machine learning algorithms in NFRs classification (i.e., SVM, NB, DT, and KNN). Another potential area for future work could be investigating the use of different classification methods, including advanced learning models such as transfer learning.

Generalization to Non-Requirements Context. Although two different datasets have been used in this research, both are requirements datasets. Therefore, another potential area for future work could be to explore whether methods for classifying NFRs can be transferred to other domains such as tweets, user reviews, and comments classification. This may include the case of analyzing the usability concerns of a product/application through classifying user reviews based on the eight identified classes and measuring their satisfaction for each class by applying sentiment analysis.

8.3.2 Extrinsic Evaluation of NFRs Classifiers

Supporting manual analysis of requirements. An optional direction of future work is to demonstrate the practical usefulness of NFRs classifiers in the analysis of requirements at the early stage of RE. This includes comparing requirements analysis workload (e.g., analysis time and defects in requirements documents) with and without using such classifiers. Winkler et al. [WGV19] conducted a similar empirical study, but they evaluated the usefulness of ML in classifying elements of requirements documents into requirements and non-requirements. However, our future work will focus on NFRs.

Supporting other RE tasks. Another direction of future work could be to utilize the classified NFRs to support different RE tasks. For example, identifying requirement type can use in generating traceability links, rewriting requirements, analyzing the quality of requirements by detecting the missing parts, and visualizing the system goals and others.

8.4 Conclusion Remarks

This thesis has contributed towards understanding the impact of three classification problems in requirements classification (i.e. high dimensionality, class imbalance, and short text classification problems). In addition to the deep understanding of the problems and existing solutions, this research proposed two new solutions to address these problems (i.e. ML4RC and SE4RC). The application of these solutions in two classification tasks (NFRs and UR classification) shows their effectiveness in classifying requirements using standard ML algorithms.

The experimental studies in this research have also revealed other issues, making NFR classification a very challenging task. These problems include shared context, overlap, bi-meaning (implicit and explicit), and a lack of consensus in definitions and representations of NFRs. Tackling such issues will improve not only the accuracy of proposed solutions but also the understanding of these requirements. Therefore, based on an analysis of the results, this research has suggested further investigations and directions for future work.

The novelty of this research is not limited to the proposed solutions and experimental studies; it also provides a systematic understanding of the use of ML in the classification of NFRs, analyses related work, identifies and fills gaps, and highlights challenges and opportunities. Therefore, this research has established a better foundation for future investigations in this field.

Bibliography

[15990] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[AAS$^+$19] Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, Lionel C Briand, and Eduardo Vaz. A machine learning-based approach for demarcating requirements in textual specifications. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 51–62. IEEE, 2019.

[ABNZ19] Waad Alhoshan, Riza Batista-Navarro, and Liping Zhao. Using frame embeddings to identify semantically related software requirements. 2019.

[AC06] JJ García Adeva and Rafael Calvo. Mining text with pimiento. *IEEE internet computing*, 10(4):27–35, 2006.

[AC$^+$10] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[AFK$^+$20] Arshad Ahmad, Chong Feng, Muzammil Khan, Asif Khan, Ayaz Ullah, Shah Nazir, and Adnan Tahir. A systematic literature review on using machine learning algorithms for software requirements identification on stack overflow. *Security and Communication Networks*, 2020, 2020.

[AG17] TP Anjos and Leila Amaral Gontijo. Usability tool to support the development process of e-commerce website. In *International Conference on Human-Computer Interaction*, pages 11–18. Springer, 2017.

[AG19] Issa Alsmadi and Keng Hoon Gan. Review of short-text classification. *International Journal of Web Information Systems*, 2019.

[AKG$^+$17] Zahra Shakeri Hossein Abad, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe, and Kurt Schneider. What works better? a study of classifying requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 496–501. IEEE, 2017.

[Ako18] Haldun Akoglu. User's guide to correlation coefficients. *Turkish journal of emergency medicine*, 18(3):91–93, 2018.

[AKSS03] Alain Abran, Adel Khelifi, Witold Suryn, and Ahmed Seffah. Usability meanings and interpretations in iso standards. *Software quality journal*, 11(4):325–338, 2003.

[alm06] *Semi-supervised learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 2006.

[Alt90] Douglas G Altman. *Practical statistics for medical research*. CRC press, 1990.

[AM81] Russell J Abbott and DK Moorhead. Software requirements and specifications: A survey of needs and languages. *Journal of Systems and Software*, 2(4):297–316, 1981.

[AM06] Sisira Adikari and Craig McDonald. User and usability modeling for hci/hmi: a research design. In *2006 International Conference on Information and Automation*, pages 151–154. IEEE, 2006.

[AML06] Sisira Adikari, Craig McDonald, and Neil Lynch. Usability in requirements engineering. 2006.

[AMPG20] Aman Agarwal, Mamta Mittal, Akshat Pathak, and Lalit Mohan Goyal. Fake news detection using a blend of neural networks: An application of deep learning. *SN Computer Science*, 1:1–9, 2020.

[ARMRMB14] David Alonso-Ríos, Eduardo Mosqueira-Rey, and Vicente Moret-Bonillo. A taxonomy-based usability study of an intelligent speed adaptation device. *International Journal of Human-Computer Interaction*, 30(7):585–603, 2014.

[ARVGMRMB09] David Alonso-Ríos, Ana Vázquez-García, Eduardo Mosqueira-Rey, and Vicente Moret-Bonillo. Usability: a critical analysis and a taxonomy. *International Journal of Human-Computer Interaction*, 26(1):53–74, 2009.

[AS02] Ian F Alexander and Richard Stevens. *Writing better requirements*. Pearson Education, 2002.

[ASR$^+$15] Aida Ali, Siti Mariyam Shamsuddin, Anca L Ralescu, et al. Classification with class imbalance problem: a review. *Int. J. Advance Soft Compu. Appl*, 7(3):176–204, 2015.

[AW15] Abdiansah Abdiansah and Retantyo Wardoyo. Time complexity analysis of support vector machines (svm) in libsvm. *International journal computer and application*, 128(3):28–34, 2015.

[AZ12] Charu C Aggarwal and ChengXiang Zhai. A survey of text classification algorithms. In *Mining text data*, pages 163–222. Springer, 2012.

[Bac11] Karen Bachmann. Collaborative techniques for developing usability requirements, 2011.

[BAMK18] Danushka Bollegala, Vincent Atanasov, Takanori Maehara, and Ken-ichi Kawarabayashi. Classinet–predicting missing features for short-text classification. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(5):1–29, 2018.

[BB02] Shirley A. Becker and Anthony Berkemeyer. Rapid application design and testing of web usability. *IEEE MultiMedia*, 9(4):38–46, 2002.

[BBC$^+$04] Pere Botella, X Burgués, JP Carvallo, X Franch, G Grau, J Marco, and C Quer. Iso/iec 9126 in practice: what do we need to know. In *Software Measurement European Forum*, volume 2004. Citeseer, 2004.

[BCMA02] Nigel Bevan, Nigel Claridge, Martin Maguire, and Maria Athousaki. Specifying and evaluating usability requirements using the common industry format. In *IFIP World Computer Congress, TC 13*, pages 149–159. Springer, 2002.

[BDCD19] Cody Baker, Lin Deng, Suranjan Chakraborty, and Josh Dehlinger. Automatic multi-class non-functional software requirements classification using neural networks. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 610–615. IEEE, 2019.

[BDLO$^+$13] Gabriele Bavota, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, Fabio Ricci, and Genoveffa Tortora. The role of artefact corpus in lsi-based traceability recovery. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 83–89. IEEE, 2013.

[Bev99] Nigel Bevan. Quality in use: Meeting user needs for quality. *Journal of systems and software*, 49(1):89–96, 1999.

[Bev00] Nigel Bevan. Iso and industry standards for user centred design, 2000. Available at `https://rauterberg.employee.id.tue.nl/lecturenotes/0H420/Usability_standards.pdf`, last accessed October 2020.

[Bev01] Nigel Bevan. International standards for hci and usability. *International journal of human-computer studies*, 55(4):533–552, 2001.

[Bev08] Nigel Bevan. Classifying and selecting ux and usability measures. In *International Workshop on Meaningful Measures: Valid Useful User Experience Measurement*, volume 11, pages 13–18, 2008.

[BFOS84] Leo Breiman, Jerome Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees chapman & hall. *New York*, 1984.

[BG04] Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep):1089–1105, 2004.

[BG17] Elsa Bakiu and Emitza Guzman. Which feature is unusable? detecting usability and user experience issues from user reviews. In

2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), pages 182–187. IEEE, 2017.

[BGOQ03] Robert Bryll, Ricardo Gutierrez-Osuna, and Francis Quek. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern recognition*, 36(6):1291–1302, 2003.

[BGST12] Daniel Berry, Ricardo Gacitua, Pete Sawyer, and Sri Fatimah Tjong. The case for dumb requirements engineering tools. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 211–217. Springer, 2012.

[Bir19] Alex Birkett. 8 ways to measure satisfaction (and improve ux), 2019.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[BKM91] Nigel Bevana, Jurek Kirakowskib, and Jonathan Maissela. What is usability. In *Proceedings of the 4th International Conference on HCI*. Citeseer, 1991.

[BM94] Nigel Bevan and Miles Macleod. Usability measurement in context. *Behaviour & information technology*, 13(1-2):132–145, 1994.

[BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[Boc10] Gregor Bochmann. Non-functional requirements, 2010. Available at `http://www.eiti.uottawa.ca/~bochmann/SEG3101/Notes/SEG3101-ch3-4%20-%20Non-Functional%20Requirements%20-%20Qualities.pdf`, last accessed October 2020.

[Boe00] Barry Boehm. Requirements that handle ikiwisi, cots, and rapid change. *Computer*, 33(7):99–102, 2000.

[Bor12] Christian Borgelt. Frequent item set mining. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 2(6):437–456, 2012.

[BPM04] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[Bro14] Jason Brownlee. Machine learning mastery. `https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features\protect\@normalcr\relax-and-how-to-get-good-at-it/`, 2014. Online; accessed March 2020.

[Bro15] Manfred Broy. Rethinking nonfunctional software requirements. *Computer*, (5):96–99, 2015.

[Bro16] Jason Brownlee. Deep learning and time series forecasting: Machine learning mastery. `https://machinelearningmastery.com/supervised\protect\@normalcr\relax-and-unsupervised-machine-learning-algorithms/`, 2016. Online; accessed August 2020.

[BSA10] JONATHAN Bertman, NEIL Skolnik, and JANE Anderson. Ehrs get a failing grade on usability. *Internal Medicine News*, 43(11):50, 2010.

[BZ19] Manal Binkhonain and Liping Zhao. A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications*, 2019.

[C$^+$98] Software & Systems Engineering Standards Committee et al. Ieee std 1061-1998—ieee standard for a software quality metrics methodology. *IEEE Computer Society, Tech. Rep*, 1998.

[CBAB12] Abhijit Chakraborty, Mrinal Kanti Baowaly, Ashraful Arefin, and Ali Newaz Bahar. The role of requirement engineering in software development life cycle. *Journal of emerging trends in computing and information sciences*, 3(5):723–729, 2012.

[CdPL09] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*, pages 363–379. Springer, 2009.

[CF14] Gregory W Corder and Dale I Foreman. *Nonparametric statistics: A step-by-step approach.* John Wiley & Sons, 2014.

[CGC10] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Information and Software Technology*, 52(4):436–445, 2010.

[Cha09] Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 875–886. Springer, 2009.

[CHMLP07] Jane Cleland-Huang, Sepideh Mazrouee, Huang Liguo, and Dan Port. nfr, March 2007. Available at `https://doi.org/10.5281/zenodo.268542`, last accessed March 2021.

[CHSZS07] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. Automated classification of non-functional requirements. *Requirements Engineering*, 12(2):103–120, 2007.

[CHTQ09] Jingnian Chen, Houkuan Huang, Shengfeng Tian, and Youli Qu. Feature selection for text classification with naïve bayes. *Expert Systems with Applications*, 36(3):5432–5435, 2009.

[Chu] Lawrence Chung. Non-functional requirements (slides). Available at `https://personal.utdallas.edu/~chung/SYSM6309/NFR-18-4-on-1.pdf`, last accessed November 2017.

[CIC15] Cic building information modelling standards (phase one), 2015. Available at `http://www.cic.hk/files/page/51/CIC%20BIM%20Standards_FINAL_ENG_v1.pdf`, last accessed November 2020.

[CJS11] Mengen Chen, Xiaoming Jin, and Dou Shen. Short text classification improved by learning multi-granularity topics. In *Twenty-Second International Joint Conference on Artificial Intelligence*. Citeseer, 2011.

[CL99] Larry L Constantine and Lucy AD Lockwood. *Software for use: a practical guide to the models and methods of usage-centered design*. Pearson Education, 1999.

[CL11] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[Cla14] Tom Clancy. The standish group report. *Chaos report*, 2014.

[CM05] Courtney D Corley and Rada Mihalcea. Measuring the semantic similarity of texts. In *Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment*, pages 13–18, 2005.

[CNYM12] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.

[Coh60] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[Com98] IEEE Computer Society. Software Engineering Technical Committee. *Ieee guide for developing system requirements specifications*. IEEE, 1998.

[Com07] Adriano Comai. Requirements-by-example, 2007. Available at `https://fdocuments.in/document/requirements-by-example.html`, last accessed October 2020.

[Con08] W. W. W. Consortium. Web content accessibility guidelines (wcag) 2.0, 2008.

[Cre] Armin Cremers. Chapter 9, non-functional requirements. Available at `https://www.yumpu.com/en/document/view/7895175/chapter-9-non-functional-requirements`, last accessed October 2020.

[CT$^+$94] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, volume 161175. Citeseer, 1994.

[CT10] Gavin C Cawley and Nicola LC Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, 11:2079–2107, 2010.

[CTL09] Chaomei Chen, Kaushal Toprani, and Natasha Lobo. Human factors in the development of trend detection and tracking techniques. In *Human Computer Interaction: Concepts, Methodologies, Tools, and Applications*, pages 1678–1686. IGI Global, 2009.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[CW08] Xue-wen Chen and Michael Wasikowski. Fast: a roc-based feature selection metric for small samples and imbalanced data classification problems. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 124–132, 2008.

[CW12] Lifei Chen and Shengrui Wang. Automated feature weighting in naive bayes for high-dimensional data classification. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1243–1252, 2012.

[CWK05] Luiz Marcio Cysneiros, Vera Maria Werneck, and Andre Kushniruk. Reusable knowledge for satisficing usability requirements. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 463–464. IEEE, 2005.

[CY04] Luiz Marcio Cysneiros and Eric Yu. Non-functional requirements elicitation. In *Perspectives on software requirements*, pages 115–138. Springer, 2004.

[DBC$^+$19] Emmanuel Gbenga Dada, Joseph Stephen Bassi, Haruna Chiroma, Adebayo Olusola Adetunmbi, Opeyemi Emmanuel Ajibuwa, et al. Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon*, 5(6):e01802, 2019.

[DCCM20] Edna Dias Canedo and Bruno Cordeiro Mendes. Software requirements classification using machine learning algorithms. *Entropy*, 22(9):1057, 2020.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DDAÇ19] Fabiano Dalpiaz, Davide Dell'Anna, Fatma Basak Aydemir, and Sercan Çevikol. Requirements classification with interpretable machine learning and dependency parsing. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 142–152. IEEE, 2019.

[DDF$^+$03] Alan Dix, Alan John Dix, Janet Finlay, Gregory D Abowd, and Russell Beale. *Human-computer interaction*. Pearson Education, 2003.

[Dem06] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[DFFP18] Fabiano Dalpiaz, Alessio Ferrari, Xavier Franch, and Cristina Palomares. Natural language processing for requirements engineering: The best is yet to come. *IEEE software*, 35(5):115–119, 2018.

[DGR12] Sanjay Kumar Dubey, Anubha Gulati, and Ajay Rana. Integrated model for software usability. *International Journal on Computer Science and Engineering*, 4(3):429, 2012.

[DHR17] Roger Deocadez, Rachel Harrison, and Daniel Rodriguez. Automatically classifying requirements from app stores: A preliminary study. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 367–371. IEEE, 2017.

[Die95] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.

[DKC14] Barnan Das, Narayanan C Krishnan, and Diane J Cook. Racog and wracog: Two probabilistic oversampling techniques. *IEEE transactions on knowledge and data engineering*, 27(1):222–234, 2014.

[DKJ05] Tore Dyba, Barbara A Kitchenham, and Magne Jorgensen. Evidence-based software engineering for practitioners. *IEEE software*, 22(1):58–65, 2005.

[DL97] Manoranjan Dash and Huan Liu. Feature selection for classification. *Intelligent data analysis*, 1(3):131–156, 1997.

[DLDPO10] Andrea De Lucia, Massimiliano Di Penta, and Rocco Oliveto. Improving source code lexicon via traceability and information retrieval. *IEEE Transactions on Software Engineering*, 37(2):205–227, 2010.

[DLP03] Kushal Dave, Steve Lawrence, and David M Pennock. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, pages 519–528, 2003.

[DLWZ19] Xuelian Deng, Yuqing Li, Jian Weng, and Jilian Zhang. Feature selection for text classification: A review. *Multimedia Tools and Applications*, 78(3):3797–3816, 2019.

[DMM08] Marie-Catherine De Marneffe and Christopher D Manning. The stanford typed dependencies representation. In *Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation*, pages 1–8, 2008.

[DPRHB10] Jean-Baptist Du Prel, Bernd Röhrig, Gerhard Hommel, and Maria Blettner. Choosing statistical tests: part 12 of a series on evaluation of scientific publications. *Deutsches Ärzteblatt International*, 107(19):343, 2010.

[DRS10] Sanjay Kumar Dubey, Ajay Rana, and Arun Sharma. Usability and development environment for software applications: An integrated model. *international journal of Advanced Research in Computer Science*, 1(2), 2010.

[DRW07] Leticia Duboc, David Rosenblum, and Tony Wicks. A framework for characterization and analysis of software system scalability. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 375–384. ACM, 2007.

[DWAJ$^+$05] Mary Dixon-Woods, Shona Agarwal, David Jones, Bridget Young, and Alex Sutton. Synthesising qualitative and quantitative evidence: a review of possible methods. *Journal of health services research & policy*, 10(1):45–53, 2005.

[EBdS20] Khadija El Bouchefry and Rafael S de Souza. Learning in big data: Introduction to machine learning. In *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, pages 225–249. Elsevier, 2020.

[ECS18] Vasiliki Efstathiou, Christos Chatzilenas, and Diomidis Spinellis. Word embeddings for the software engineering domain. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 38–41, 2018.

[est07] E-store project software requirements specification, 2007. Aviable at `https://www.studocu.com/in/document/lovely-professional-university/software-engineering/other/estore-software-requirement-specification-srs/3071299/view`, last accessed September 2020.

[EVF16] Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández. Are" non-functional" requirements really non-functional? an investigation of non-functional requirements in practice. In *Proceedings of the 38th International Conference on Software Engineering*, pages 832–842, 2016.

[FB03] Eelke Folmer and Jan Bosch. Usability patterns in software architecture. In *Proc. 10th Int. Conf. on Human-Computer Interaction (HCII2003) Volume I*, pages 93–97, 2003.

[FD96] Anthony Finkelstein and John Dowell. A comedy of errors: the london ambulance service case study. In *Proceedings of the 8th International Workshop on Software Specification and Design*, pages 2–4. IEEE, 1996.

[FDE$^+$17] Alessio Ferrari, Felice Dell'Orletta, Andrea Esuli, Vincenzo Gervasi, and Stefania Gnesi. Natural language requirements processing: a 4d vision. *IEEE Software*, 34(6):28–35, 2017.

[FDSSVA19] Yair Fogel-Dror, Shaul R Shenhav, Tamir Sheafer, and Wouter Van Atteveldt. Role-based association of verbs, actions, and sentiments with entities in political discourse. *Communication Methods and Measures*, 13(2):69–82, 2019.

[FH10] Xinghua Fan and Hongge Hu. Construction of high-quality feature extension mode library for chinese short-text classification. In *2010 WASE International Conference on Information Engineering*, volume 2, pages 87–90. Ieee, 2010.

[FHH00] Erik Frøkjær, Morten Hertzum, and Kasper Hornbæk. Measuring usability: are effectiveness, efficiency, and satisfaction really correlated? In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 345–352, 2000.

[FHH$^+$09] Eibe Frank, Mark Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, Ian H Witten, and Len Trigg. Weka-a machine learning workbench for data mining. In *Data mining and knowledge discovery handbook*, pages 1269–1277. Springer, 2009.

[Fir] Usability First. Requirements specification. Avilable at `http://www.usabilitynet.org/trump/methods/recommended/requirements.htm`, last accessed September 2017.

[Fir04] Donald Firesmith. Specifying reusable security requirements. *J. Object Technol.*, 3(1):61–75, 2004.

[FJWC01] Xavier Ferré, Natalia Juristo, Helmut Windl, and Larry Constantine. Usability basics for software developers. *IEEE software*, 18(1):22–29, 2001.

[Fle71] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[FML05] Dimitris Fragoudis, Dimitris Meretakis, and Spiridon Likothanassis. Best terms: an efficient feature-selection algorithm for text categorization. *Knowledge and Information Systems*, 8(1):16–33, 2005.

[For03] George Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of machine learning research*, 3(Mar):1289–1305, 2003.

[FS10] George Forman and Martin Scholz. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *Acm Sigkdd Explorations Newsletter*, 12(1):49–57, 2010.

[fSEC$^+$11] International Organization for Standardization/International Electrotechnical Commission et al. Iso/iec 25010; systems and software engineering-systems and software quality requirements and evaluation (square)-system and software quality models". 2011.

[FVGB04] Eelke Folmer, Jilles Van Gurp, and Jan Bosch. Software architecture analysis of usability. In *IFIP International Conference on Engineering for Human-Computer Interaction*, pages 38–58. Springer, 2004.

[GAS14] Deepak Gupta, Anil Ahlawat, and Kalpna Sagar. A critical analysis of a hierarchy based usability model. In *2014 international conference on contemporary computing and informatics (IC3I)*, pages 255–260. IEEE, 2014.

[GBC16] I Goodfelow, Y Bengio, and A Courville. Deep learning (adaptive computation and machine learning series). *DOI*, 10:1762–1766, 2016.

[GCCH10] Marek Gibiec, Adam Czauderna, and Jane Cleland-Huang. Towards mining replacement queries for hard-to-retrieve traces. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 245–254, 2010.

[GCCH17] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. Semantically enhanced software traceability using deep learning techniques. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 3–14. IEEE, 2017.

[GDR04] Thomas Geis, Wolfgang Dzida, and Wolfgang Redtendbacher. *Specifying usability requirements and test criteria for interactive systems*. Citeseer, 2004.

[Gér19] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[GF94] Orlena CZ Gotel and CW Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101. IEEE, 1994.

[GFB$^+$11] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2011.

[GIS10] Diman Ghazi, Diana Inkpen, and Stan Szpakowicz. Hierarchical versus flat classification of emotions in text. In *Proceedings of the NAACL HLT 2010 workshop on computational approaches to analysis and generation of emotion in text*, pages 140–146. Association for Computational Linguistics, 2010.

[GJ02] Daniel Gildea and Daniel Jurafsky. Automatic labeling of semantic roles. *Computational linguistics*, 28(3):245–288, 2002.

[Gli05] Martin Glinz. Rethinking the notion of non-functional requirements. In *Proc. Third World Congress for Software Quality*, volume 2, pages 55–64, 2005.

[Gli07] Martin Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26. IEEE, 2007.

[GMFG05] Miha Grčar, Dunja Mladenič, Blaž Fortuna, and Marko Grobelnik. Data sparsity issues in the collaborative filtering framework. In *International Workshop on Knowledge Discovery on the Web*, pages 58–76. Springer, 2005.

[GMPCH14] Jin Guo, Natawut Monaikul, Cody Plepel, and Jane Cleland-Huang. Towards an intelligent domain-specific traceability solution. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 755–766, 2014.

[Gol03] Nahid Golafshani. Understanding reliability and validity in qualitative research. *The qualitative report*, 8(4):597–607, 2003.

[Gou95] John D Gould. How to design usable systems (excerpt). In *Readings in Human–Computer Interaction*, pages 93–121. Elsevier, 1995.

[GPM03] Global personal marketplace, 2003. Available at `https://www.it.uu.se/edu/course/homepage/pvt/SRS.pdf`, last accessed September 2020.

[Gra92] Robert B Grady. *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc., 1992.

[Gro99] Marko Grobelnik. Feature selection for unbalanced class distribution and naive bayes. In *International conference on machine learning*. Citeseer, 1999.

[GS17] Barney G Glaser and Anselm L Strauss. *Discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017.

[GSM07] Vicente García, Jose Sánchez, and Ramon Mollineda. An empirical study of the behavior of classifiers on imbalanced and overlapped data sets. In *Iberoamerican Congress on Pattern Recognition*, pages 397–406. Springer, 2007.

[GSWG86] Michael Good, Thomas M Spine, John Whiteside, and Peter George. User-derived impact analysis as a tool for usability engineering. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pages 241–246, 1986.

[GWSH09] David Grosse-Wentrup, Alexandra Stier, and Uvo Hoelscher. Supporting tool for usability specifications. In *World Congress on Medical Physics and Biomedical Engineering, September 7-12, 2009, Munich, Germany*, pages 845–847. Springer, 2009.

[GYK04] Karin Garmer, Jessica Ylvén, and IC MariAnne Karlsson. User participation in requirements elicitation comparing focus group interviews and usability tests for eliciting usability requirements for medical equipment: a case study. *International Journal of Industrial Ergonomics*, 33(2):85–98, 2004.

[Har18] Jamie Harding. *Qualitative data analysis: From start to finish*. SAGE Publications Limited, 2018.

[HAS15] Lina A Hasan and Khalid T Al-Sarayreh. An integrated measurement model for evaluating usability attributes. In *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication*, pages 1–6, 2015.

[HDS06] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering*, 32(1):4, 2006.

[HG09] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[HH93] Deborah Hix and H Rex Hartson. *Developing user interfaces: Ensuring usability through product and process*. Wiley, 1993.

[HJD11] Elizabeth Hull, Ken Jackson, and Jeremy Dick. *Introduction*, pages 1–23. Springer London, London, 2011.

[HK11] Jan Hauke and Tomasz Kossowski. Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae*, 30(2):87–93, 2011.

[HKKT20a] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F Tichy. Norbert: Transfer learning for requirements classification. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 169–179. IEEE, 2020.

[HKKT20b] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F. Tichy. Supplementary Material of "NoRBERT: Transfer Learning for Requirements Classification", May 2020. Online; accessed 3 August 2020.

[HKO08] Ishrar Hussain, Leila Kosseim, and Olga Ormandjieva. Using linguistic knowledge to classify non-functional requirements in srs documents. In *International Conference on Application of Natural Language to Information Systems*, pages 287–298. Springer, 2008.

[HKP12] Jiawei Han, Micheline Kamber, and Jian Pei. 8 - classification: Basic concepts. In Jiawei Han, Micheline Kamber, and Jian Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 327–391. Morgan Kaufmann, Boston, third edition edition, 2012.

[Hoo94] Ivy Hooks. Writing good requirements. In *INCOSE International Symposium*, volume 4, pages 1247–1253. Wiley Online Library, 1994.

[Hos11] C Hoskinson. Army's faulty computer system hurts operations. *Politico*, 2011.

[HS13] Steffen Hedegaard and Jakob Grue Simonsen. Extracting usability and user experience information from online user reviews. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2089–2098, 2013.

[HSZC09] Xia Hu, Nan Sun, Chao Zhang, and Tat-Seng Chua. Exploiting internal and external semantics for the clustering of short texts using world knowledge. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 919–928, 2009.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Overview of supervised learning. In *The elements of statistical learning*, pages 9–41. Springer, 2009.

[IEL18] Tahira Iqbal, Parisa Elahidoost, and Levi Lucio. A bird's eye view on requirements engineering and machine learning. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 11–20. IEEE, 2018.

[II04] ISO ISO and TR IEC. Iso/iec 9126, software engineering - product quality, parts 1-4. *International Organization for Standardization, Geneva, Switzerland. Switzerland*, 1999-2004.

[II09] Aminul Islam and Diana Inkpen. Semantic similarity of short texts. *Recent Advances in Natural Language Processing V*, 309:227–236, 2009.

[IK09] Muhammad Ilyas and Josef Kung. A similarity measurement framework for requirements engineering. In *2009 Fourth International Multi-Conference on Computing in the Global Information Technology*, pages 31–34. IEEE, 2009.

[IKT05] M Ikonomakis, Sotiris Kotsiantis, and V Tampakas. Text classification using machine learning techniques. *WSEAS transactions on computers*, 4(8):966–974, 2005.

[Iso98] W Iso. 9241-11. ergonomic requirements for office work with visual display terminals (vdts). *The international organization for standardization*, 45(9), 1998.

[ISO18] W ISO. 9241–11: 2018 ergonomics of human-system interaction–part 11. *Usability: Definitions and Concepts*, 2018.

[ITK06] Tasadduq Imam, Kai Ming Ting, and Joarder Kamruzzaman. z-svm: An svm for improved classification of imbalanced data. In *Australasian Joint Conference on Artificial Intelligence*, pages 264–273. Springer, 2006.

[ITS] Information Technology Service Management(ITSM) Tool Implementation Services. Available at `https://buyandsell.gc.ca/cds/public/2019/01/24/9af871f99ba35e239bc6bf564e733d02/itsm_annex_a_sow_en.pdf`, last accessed November 2020.

[JBB14] A. Jovic, K. Brkic, and N. Bogunovic. An overview of free software tools for general data mining. In *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1112–1117, May 2014.

[JBSSA04] Bonnie E John, Len Bass, Maria-Isabel Sanchez-Segura, and Rob J Adams. Bringing usability concerns to the design of software architecture. In *IFIP International Conference on Engineering for Human-Computer Interaction*, pages 1–19. Springer, 2004.

[JC97] Jay J Jiang and David W Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*, 1997.

[JGDE08] Andreas Janecek, Wilfried Gansterer, Michael Demel, and Gerhard Ecker. On the relationship between feature selection and classification accuracy. In *New challenges for feature selection in data mining and knowledge discovery*, pages 90–105, 2008.

[JHS09] Wei Jin, Hung Hay Ho, and Rohini K Srihari. Opinionminer: a novel machine learning system for web opinion mining and extraction. In *Proceedings of the 15th ACM SIGKDD international* *conference on Knowledge discovery and data mining*, pages 1195–1204, 2009.

[JJ04] Taeho Jo and Nathalie Japkowicz. Class imbalances versus small disjuncts. *ACM Sigkdd Explorations Newsletter*, 6(1):40–49, 2004.

[JM00] Daniel Jurasky and James H Martin. Speech and language processing: An introduction to natural language processing. *Computational Linguistics and Speech Recognition. Prentice Hall, New Jersey*, 2000.

[JM20] Daniel Jurafsky and H. James Martin. *Speech and language processing*. Pearson Education India, third edition, 2020.

[JMJ16] Rajni Jindal, Ruchika Malhotra, and Abha Jain. Automated classification of security requirements. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2027–2033. IEEE, 2016.

[JMSS07] Natalia Juristo, Ana Moreno, and Maria-Isabel Sanchez-Segura. Guidelines for eliciting usability functionalities. *IEEE Transactions on Software Engineering*, 33(11):744–758, 2007.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[Joa99] Thorsten Joachims. Svmlight: Support vector machine. *SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund*, 19(4), 1999.

[Jon72] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

[JYZ$^+$11] Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu, and Tiejun Zhao. Target-dependent twitter sentiment classification. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 151–160, 2011.

[KBdR16] Tom Kenter, Alexey Borisov, and Maarten de Rijke. Siamese cbow: Optimizing word embeddings for sentence representations, 2016.

[KBLK10] Aurangzeb Khan, Baharum Baharudin, Lam Hong Lee, and Khairullah Khan. A review of machine learning algorithms for text-documents classification. *Journal of advances in information technology*, 1(1):4–20, 2010.

[KC07] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. 2007.

[KF68] Henry Kučera and W. Nelson Francis. *Computational analysis of present-day American English*. Providence, Brown University Press., 1968.

[KG18] Daniel E Kim and Mikhail Gofman. Comparison of shallow and deep neural networks for network intrusion detection. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 204–208. IEEE, 2018.

[KHB$^+$18] Alfred Krzywicki, Bradford Heap, Michael Bain, Wayne Wobcke, and Susanne Schmeidl. Using word embeddings with linear models for short text classification. In *Australasian Joint Conference on Artificial Intelligence*, pages 819–827. Springer, 2018.

[KJMH$^+$19] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[KKN14] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. In *2014 Science and Information Conference*, pages 372–378. IEEE, 2014.

[KM17] Zijad Kurtanović and Walid Maalej. Automatically classifying functional and non-functional requirements using supervised machine learning. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 490–495. IEEE, 2017.

[KMMM10] K.V. Kale, S.C. Mehrotra, R.R. Manza, and R.R. Manza. *Computer Vision and Information Technology: Advances and Applications*. I.K. International Publishing House Pvt. Limited, 2010.

[KMNF06] Svetlana Kiritchenko, Stan Matwin, Richard Nock, and A Fazel Famili. Learning and evaluation in the presence of class hierarchies: Application to text categorization. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 395–406. Springer, 2006.

[KNL14] Mohamad Kassab, Colin Neill, and Phillip Laplante. State of practice in requirements engineering: contemporary data. *Innovations in Systems and Software Engineering*, 10(4):235–241, 2014.

[KO14] Eric Knauss and Daniel Ott. (semi-) automatic categorization of natural language requirements. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 39–54. Springer, 2014.

[Kom20] Andreas Komninos. An introduction to usability, 2020. Available at `https://www.interaction-design.org/literature/article/an-introduction-to-usability`, last accessed October 2020.

[KPP$^+$02] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering*, 28(8):721–734, 2002.

[Kri11] Klaus Krippendorff. Computing krippendorff's alpha-reliability. 2011.

[KS96] Gerald Kotonya and Ian Sommerville. Requirements engineering with viewpoints. *Software Engineering Journal*, 11(1):5–18, 1996.

[KWM11] Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. Twitter sentiment analysis: The good the bad and the omg! In *Fifth International AAAI conference on weblogs and social media*. Citeseer, 2011.

[KZP06] Sotiris B Kotsiantis, Ioannis D Zaharakis, and Panayiotis E Pintelas. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, 26(3):159–190, 2006.

[KZP07] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[L^+98] Dekang Lin et al. An information-theoretic definition of similarity. In *Icml*, volume 98, pages 296–304, 1998.

[Lap17] Phillip A Laplante. *Requirements engineering for software and systems*. CRC Press, 2017.

[LC98] C Leacock and M Chodorow. Combining local context and wordnet sense similarity for word sense identification. wordnet, an electronic lexical database. *The MIT Press*, 1998.

[LCH13] Yonghua Li and Jane Cleland-Huang. Ontology-based trace retrieval. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 30–36. IEEE, 2013.

[LD13] Rushi Longadge and Snehalata Dongre. Class imbalance problem in data mining review. *arXiv preprint arXiv:1305.1707*, 2013.

[LDBT15] Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304, 2015.

[Les04] Stefan Lessmann. Solving imbalanced classification problems with support vector machines. In *IC-AI*, volume 4, pages 214–220, 2004.

[LFL98] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[LHG$^+$18] Chuanyi Li, Liguo Huang, Jidong Ge, Bin Luo, and Vincent Ng. Automatically classifying user requests in crowdsourcing requirements engineering. *Journal of Systems and Software*, 138:108–123, 2018.

[LHM$^+$14] Feng-Lin Li, Jennifer Horkoff, John Mylopoulos, Renata SS Guizzardi, Giancarlo Guizzardi, Alexander Borgida, and Lin Liu. Non-functional requirements as qualities, with a spice of ontology. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 293–302. IEEE, 2014.

[LHW$^+$17] Peipei Li, Lu He, Haiyan Wang, Xuegang Hu, Yuhong Zhang, Lei Li, and Xindong Wu. Learning from short text streams with topic drifts. *IEEE transactions on cybernetics*, 48(9):2697–2711, 2017.

[Lis10] Pen Lister. Usability evaluation formative techniques, 2010. Available at `https://www.slideshare.net/PenLister/uid-formative-evaluation`, last accessed October 2017.

[LJ98] Yong H Li and Anil K Jain. Classification of text documents. *The Computer Journal*, 41(8):537–546, 1998.

[LL17] Mengmeng Lu and Peng Liang. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 344–353, 2017.

[LLHZ16] Siwei Lai, Kang Liu, Shizhu He, and Jun Zhao. How to generate a good word embedding. *IEEE Intelligent Systems*, 31(6):5–14, 2016.

[LLS09] Ying Liu, Han Tong Loh, and Aixin Sun. Imbalanced text classification: A term weighting approach. *Expert systems with Applications*, 36(1):690–701, 2009.

[LMP04] Mich Luisa, Franch Mariangela, and Novi Inverardi Pierluigi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56, 2004.

[Löw93] Jonas Löwgren. *Human-computer interaction: What every system developer should know*. Studentlitteratur, 1993.

[LP98] Andreas Lecerof and Fabio Paternò. Automatic support for usability evaluation. *IEEE transactions on software engineering*, 24(10):863–888, 1998.

[LSZS20] Qianmu Li, Yanjun Song, Jing Zhang, and Victor S Sheng. Multiclass imbalanced learning with one-versus-one decomposition and spectral clustering. *Expert Systems with Applications*, 147:113152, 2020.

[LT16] Weiwei Liu and Ivor W Tsang. Sparse perceptron decision tree for millions of dimensions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1881–1887, 2016.

[LVC$^+$19] Márcia Lima, Victor Valle, Estevão Costa, Fylype Lira, and Bruno Gadelha. Software engineering repositories: Expanding the promise database. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 427–436, 2019.

[LWDD10] Hong Bo Li, Wei Wang, Hong Wei Ding, and Jin Dong. Trees weighting random forest method for classifying high-dimensional noisy data. In *2010 IEEE 7th International Conference on E-Business Engineering*, pages 160–163. IEEE, 2010.

[LWZ08] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2008.

[LWZ$^+$15] Peipei Li, Haixun Wang, Kenny Q Zhu, Zhongyuan Wang, Xuegang Hu, and Xindong Wu. A large probabilistic semantic network based approach to compute term similarity. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2604–2617, 2015.

[LY98] Søren Lauesen and Houman Younessi. Six styles for usability requirements. In *REFSQ*, volume 98, pages 155–166, 1998.

[LY05] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on knowledge and data engineering*, 17(4):491–502, 2005.

[LZ05] Yi Liu and Yuan F Zheng. One-against-all multi-class svm classification using reliability measures. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 849–854. IEEE, 2005.

[LZ11] Yuxuan Li and Xiuzhen Zhang. Improving k nearest neighbor with exemplar generalization for imbalanced classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 321–332. Springer, 2011.

[Mac13] Lucas Machado. Usability requirements and their elicitation, 2013. Available at `https://www.slideshare.net/sirlucasmachado/usability-requirements-and-their-elicitation`, last accessed October 2020.

[Man14] Yuan Man. Feature extension for short text categorization using frequent term sets. *Procedia Computer Science*, 31:663–670, 2014.

[May99] Deborah J Mayhew. The usability engineering lifecycle. In *CHI'99 Extended Abstracts on Human Factors in Computing Systems*, pages 147–148, 1999.

[MB02] Ernst Mayr and Walter J Bock. Classifications and other ordering systems. *Journal of Zoological Systematics and Evolutionary Research*, 40(4):169–194, 2002.

[MBF$^+$90] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to wordnet: An on-line lexical database. *International journal of lexicography*, 3(4):235–244, 1990.

[MBM13] Hendrik Meth, Manuel Brhel, and Alexander Maedche. The state of the art in automated requirements elicitation. *Information and Software Technology*, 55(10):1695–1709, 2013.

[MBW$^+$19] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 810:1–124, 2019.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[MCI$^+$08] Jennifer Marlow, Paul Clough, Neil Ireson, J Cigarrán Recuero, Javier Artiles, and Franca Debole. The multimatch project: Multilingual/multimedia access to cultural heritage on the web. In *Museums on the Web Conference (MW2008): Proceedings, J. Trant and D. Bearman (eds). Toronto: Archives & Museum Informatics*, 2008.

[MCN92] John Mylopoulos, Lawrence Chung, and Brian Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on software engineering*, 18(6):483–497, 1992.

[MCS$^+$06] Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. Corpus-based and knowledge-based measures of text semantic similarity. In *Aaai*, volume 6, pages 775–780, 2006.

[Mey85] B. Meyer. On formalism in specifications. *IEEE Softw.*, 2(1):6–26, January 1985.

[Mey93] Bertrand Meyer. On formalism in specifications. In *Program Verification*, pages 155–189. Springer, 1993.

[MHG13] Lingling Meng, Runqing Huang, and Junzhong Gu. A review of semantic similarity measures in wordnet. *International Journal of Hybrid Information Technology*, 6(1):1–12, 2013.

[MJ51] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[MKIZ14] Hamid Mousavi, Deirdre Kerr, Markus Iseli, and Carlo Zaniolo. Mining semantic structures from syntactic structures in free text documents. In *2014 IEEE International Conference on Semantic Computing*, pages 84–91. IEEE, 2014.

[MKMG97] Robert T Monroe, Andrew Kompanek, Ralph Melton, and David Garlan. Architectural styles, design patterns, and objects. *IEEE software*, 14(1):43–52, 1997.

[ML11] Jin Matsuoka and Yves Lepage. Ambiguity spotting using wordnet semantic similarity in support to recommended practice for software requirements specifications. In *2011 7th International Conference on Natural Language Processing and Knowledge Engineering*, pages 479–484. IEEE, 2011.

[MLTB93] George A Miller, Claudia Leacock, Randee Tengi, and Ross T Bunker. A semantic concordance. In *HUMAN LANGUAGE TECHNOLOGY: Proceedings of a Workshop Held at Plainsboro, New Jersey, March 21-24, 1993*, 1993.

[MM03] Andrian Marcus and Jonathan I Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 125–135. IEEE, 2003.

[MMS13] Megha Mishra, Vishnu Kumar Mishra, and HR Sharma. Question classification using semantic, syntactic and lexical features. *International Journal of Web & Semantic Technology*, 4(3):39, 2013.

[MN15] Anas Mahmoud and Nan Niu. On the role of semantics in automated requirements tracing. *Requirements Engineering*, 20(3):281–300, 2015.

[Mon02] Andrew F Monk. Fun, communication and dependability: extending the concept of usability. In *People and Computers XVI-Memorable Yet Invisible*, pages 3–14. Springer, 2002.

[MR05] Oded Maimon and Lior Rokach. Decomposition methodology for knowledge discovery and data mining. In *Data mining and knowledge discovery handbook*, pages 981–1003. Springer, 2005.

[MRARMB09] Eduardo Mosqueira-Rey, David Alonso-Ríos, and Vicente Moret-Bonillo. Usability taxonomy and context-of-use taxonomy for usability analysis. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 812–817. IEEE, 2009.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[MT09] Fernando Molina and Ambrosio Toval. Integrating usability requirements that can be evaluated in design time into model driven engineering of web information systems. *Advances in Engineering Software*, 40(12):1306–1317, 2009.

[MW16] Anas Mahmoud and Grant Williams. Detecting, classifying, and tracing non-functional software requirements. *Requirements Engineering*, 21(3):357–381, 2016.

[MY01] Larry M Manevitz and Malik Yousef. One-class svms for document classification. *Journal of machine Learning research*, 2(Dec):139–154, 2001.

[MY07] Larry Manevitz and Malik Yousef. One-class document classification via neural networks. *Neurocomputing*, 70(7-9):1466–1481, 2007.

[MYZ13] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013.

[MZN10] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 311–317, 2010.

[Nas12] Victor Nassar. Common criteria for usability review. *Work*, 41(Supplement 1):1053–1057, 2012.

[NE00] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46, 2000.

[Nie94] Jakob Nielsen. *Usability engineering*. Morgan Kaufmann, 1994.

[NL03] Colin J Neill and Phillip A Laplante. Requirements engineering: the state of the practice. *IEEE software*, 20(6):40–45, 2003.

[NM19] Roberto Navigli and Federico Martelli. An overview of word and sense similarity. *Natural Language Engineering*, 25(6):693–714, 2019.

[NMM06] Kamal Nigam, Andrew McCallum, and Tom M Mitchell. Semi-supervised text classification using em., 2006.

[NPK$^+$16] Harikrishna Narasimhan, Weiwei Pan, Purushottam Kar, Pavlos Protopapas, and Harish G Ramaswamy. Optimizing the multiclass f-measure via biconcave programming. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1101–1106. IEEE, 2016.

[NS11] Mohd Hairul Nizam Nasir and Shamsul Sahibuddin. Critical success factors for software projects: A comparative study. *Scientific research and essays*, 6(10):2174–2186, 2011.

[oDRC$^+$01] J Natt och Dag, Björn Regnell, Pär Carlshamre, Michael Andersson, and Joachim Karlsson. Evaluating automated support for requirements similarity analysis in market-driven development. In *Proc. 7th Int. Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'01)*. Citeseer, 2001.

[(OP] OPEN Process Framework Repository Organization (OPFRO). Usability requirements. Available at `http://www.opfro.org/index.html?Components/WorkProducts/RequirementsSet/Requirements/UsabilityRequirements.html~Contents`, last accessed September 2020.

[OPCF$^+$13] Yeshica Isela Ormeño, Jose Ignacio Panach, Nelly Condori-Fern, Óscar Pastor, et al. Towards a proposal to capture usability requirements through guidelines. In *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12. IEEE, 2013.

[OSDCI11] Jesús Oliva, José Ignacio Serrano, María Dolores Del Castillo, and Ángel Iglesias. Symss: A syntax-based measure for short-text semantic similarity. *Data & Knowledge Engineering*, 70(4):390–405, 2011.

[PA12] David MW Powers and Adham Atyabi. The problem of cross-validation: averaging and bias, repetition and significance. In *2012 Spring Congress on Engineering and Technology*, pages 1–5. IEEE, 2012.

[PARS13] Anuja Priyam, GR Abhijeeta, Anju Rathee, and Saurabh Srivastava. Comparative analysis of decision tree classification algorithms. *International Journal of current engineering and technology*, 3(2):334–337, 2013.

[PBM04] Ronaldo C Prati, Gustavo EAPA Batista, and Maria Carolina Monard. Class imbalances versus class overlapping: an analysis of a learning system behavior. In *Mexican international conference on artificial intelligence*, pages 312–321. Springer, 2004.

[PBP03] Siddharth Patwardhan, Satanjeev Banerjee, and Ted Pedersen. Using measures of semantic relatedness for word sense disambiguation. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 241–257. Springer, 2003.

[PBU93] Jenny Preece, David Benyon, and Open University. *A guide to usability: Human factors in computing*. Addison-Wesley Longman Publishing Co., Inc., 1993.

[PGK05] Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106, 2005.

[Phy09] Thair Nu Phyu. Survey of classification techniques in data mining. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 18–20, 2009.

[PMB09] Francisco Pereira, Tom Mitchell, and Matthew Botvinick. Machine learning classifiers and fmri: a tutorial overview. *Neuroimage*, 45(1):S199–S209, 2009.

[PNH08] Xuan-Hieu Phan, Le-Minh Nguyen, and Susumu Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceedings of the 17th international conference on World Wide Web*, pages 91–100, 2008.

[PP14] Vijay Pappu and Panos M Pardalos. High-dimensional data classification. In *Clusters, Orders, and Trees: Methods and Applications*, pages 119–150. Springer, 2014.

[PRS$^+$94] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey. *Human-computer interaction*. Addison-Wesley Longman Ltd., 1994.

[PS03] Fuchun Peng and Dale Schuurmans. Combining naive bayes and n-gram language models for text classification. In *European Conference on Information Retrieval*, pages 335–350. Springer, 2003.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PVSGOH20] J Manuel Pérez-Verdejo, Angel J Sánchez-García, and Jorge Octavio Ocharán-Hernández. A systematic literature review on machine learning for automated requirements classification. In *2020*

8th International Conference in Software Engineering Research and Innovation (CONISOFT), pages 21–28. IEEE, 2020.

[PZZ12] Piyaphol Phoungphol, Yanqing Zhang, and Yichuan Zhao. Robust multiclass classification for learning from imbalanced biomedical data. *Tsinghua Science and technology*, 17(6):619–628, 2012.

[QLL$^+$10] Xiaojun Quan, Gang Liu, Zhi Lu, Xingliang Ni, and Liu Wenyin. Short text similarity based on probabilistic topics. *Knowledge and information systems*, 25(3):473–491, 2010.

[Que01] Whitney Quesenbery. What does usability mean: Looking beyondease of use'. In *Annual conference-society for technical communication*, volume 48, pages 432–436. Citeseer, 2001.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[Qui93] JR Quinlan. C4. 5: Programs for machine learning. morgan kaufmann, san francisco. *C4. 5: Programs for machine learning. Morgan Kaufmann, San Francisco.*, 1993.

[Ran05] Justus J Randolph. Free-marginal multirater kappa (multirater k [free]): An alternative to fleiss' fixed-marginal multirater kappa. *Online submission*, 2005.

[RDPM19] Alejandro Rago, J Andres Diaz-Pace, and Claudia Marcos. Do concern mining tools really help requirements analysts? an empirical study of the vetting process. *Journal of Systems and Software*, 156:181–203, 2019.

[Res95] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*, 1995.

[RHS05] Chuck Rosenberg, Martial Hebert, and Henry Schneiderman. Semi-supervised self-training of object detection models. *Carnegie Mellon University*, 2005.

[roc] FactoryTalk View Site Edition User's Guide. Available at `https://literature.rockwellautomation.com/idc/groups/`

literature/documents/um/viewse-um006_-en-e.pdf, last accessed November 2020.

[ROW13] Abderahman Rashwan, Olga Ormandjieva, and René Witte. Ontology-based classification of non-functional requirements in software specifications: a new corpus and svm-based classifier. In *2013 IEEE 37th Annual Computer Software and Applications Conference*, pages 381–386. IEEE, 2013.

[RP92] Colette Rolland and Christophe Proix. A natural language approach for requirements engineering. In *International Conference on Advanced Information Systems Engineering*, pages 257–277. Springer, 1992.

[RR12] Suzanne Robertson and James Robertson. *Mastering the requirements process: Getting requirements right*. Addison-wesley, 2012.

[RRSK10] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[Rum12] Nikolaus Rumm. Intelligent tools for policy design, 2012. Available at https://cordis.europa.eu/docs/projects/cnect/9/287119/080/deliverables/001-D31SoftwareRequirementsSpecificationandUseCases.pdf, last accessed October 2020.

[Rya93] Kevin Ryan. The role of natural language in requirements engineering. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 240–242. IEEE, 1993.

[Sak16] Tetsuya Sakai. Two sample t-tests for ir evaluation: Student or welch? In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1045–1048, 2016.

[SCM+19] Juan Salas, Alberto Chang, Lourdes Montalvo, Almendra Núñez, Max Vilcapoma, Arturo Moquillaza, Braulio Murillo, and Freddy Paz. Guidelines to evaluate the usability and user experience of learning support platforms: A systematic review. In *Iberoamerican Workshop on Human-Computer Interaction*, pages 238–254. Springer, 2019.

[SDKP06] Ahmed Seffah, Mohammad Donyaee, Rex B Kline, and Harkirat K Padda. Usability measurement and metrics: A consolidated model. *Software Quality Journal*, 14(2):159–178, 2006.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[SFHA14] Hassan Saif, Miriam Fernández, Yulan He, and Harith Alani. On stopwords, filtering and data sparsity for sentiment analysis of twitter. In *LREC 2014, Ninth International Conference on Language Resources and Evaluation. Proceedings.*, pages 810–817, 2014.

[SG14] Sonia Singh and Priyanka Gupta. Comparative study id3, cart and c4. 5 decision tree algorithm: a survey. *International Journal of Advanced Information Science and Technology (IJAIST)*, 27(27):97–103, 2014.

[SGR09] Richard Berntsson Svensson, Tony Gorschek, and Björn Regnell. Quality requirements in practice: An interview study in requirements engineering for embedded systems. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 218–232. Springer, 2009.

[SGSG19] Jenni AM Sidey-Gibbons and Chris J Sidey-Gibbons. Machine learning in medicine: a practical introduction. *BMC medical research methodology*, 19(1):64, 2019.

[SH06] Mehran Sahami and Timothy D Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th international conference on World Wide Web*, pages 377–386, 2006.

[Sha09] Brian Shackel. Usability–context, framework, definition, design and evaluation. *Interacting with computers*, 21(5-6):339–346, 2009.

[SHA12] Hassan Saif, Yulan He, and Harith Alani. Alleviating data sparsity for twitter sentiment analysis. CEUR Workshop Proceedings (CEUR-WS.org), 2012.

[SIL07] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[SIL15] Guzman Santafe, Iñaki Inza, and Jose A Lozano. Dealing with the evaluation of supervised classification algorithms. *Artificial Intelligence Review*, 44(4):467–508, 2015.

[SK74] Andrew Jhon Scott and M Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, pages 507–512, 1974.

[SKVHN09] Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Rusboost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 40(1):185–197, 2009.

[SL09] Jeff Sauro and James R Lewis. Correlations among prototypical usability metrics: evidence for the construct of usability. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1609–1618, 2009.

[SLRR05] Américo Sampaio, Neil Loughran, Awais Rashid, and Paul Rayson. Mining aspects in requirements. In *Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*, 2005.

[SM98] Alistair Sutcliffe and Neil Maiden. The domain theory for requirements engineering. *IEEE Transactions on Software Engineering*, 24(3):174–196, 1998.

[Soe20] Mads Soegaard. Usability: A part of the user experience, 2020. Available at `https://www.interaction-design.org/literature/article/usability-a-part-of-the-user-experience`, last accessed October 2020.

[Som05] Ian Sommerville. Integrated requirements engineering: A tutorial. *IEEE software*, 22(1):16–23, 2005.

[Som11] Ian Sommerville. Software engineering 9th edition. *ISBN-10*, 137035152:18, 2011.

[SP04] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition)*. Pearson Addison Wesley, 2004.

[SRS14] Vibhu Saujanya Sharma, Roshni R Ramnani, and Shubhashis Sengupta. A framework for identifying and analyzing non-functional requirements from text. In *Proceedings of the 4th international workshop on twin peaks of requirements and architecture*, pages 1–8, 2014.

[SS17] Kalpna Sagar and Anju Saha. A systematic review of software usability studies. *International Journal of Information Technology*, pages 1–24, 2017.

[SS20] Alberto Rodrigues da Silva and Carolina Lisboa Sequeira. Towards a library of usability requirements. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 1371–1378, 2020.

[SSS07] Forrest Shull, Janice Singer, and Dag IK Sjøberg. *Guide to advanced empirical software engineering*. Springer, 2007.

[sta09] What is the best way of formally expressing usability requirements?, 2009. Available at `https://stackoverflow.com/questions/513230/what-is-the-best-way-of-formally-expressing-usability-\protect\@normalcr\relaxrequirements`, last accessed October 2020.

[Stą17] Katarzyna Stąpor. Evaluating and comparing classifiers: Review, some recommendations and limitations. In *International Conference on Computer Recognition Systems*, pages 12–21. Springer, 2017.

[Ste16] Jerzy Stefanowski. Dealing with data difficulty factors while learning from imbalanced data. In *Challenges in computational statistics and data mining*, pages 333–363. Springer, 2016.

[SW13] John Slankas and Laurie Williams. Automated extraction of non-functional requirements in available documentation. In *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, pages 9–16. IEEE, 2013.

[Swa19] Manohar Swamynathan. *Mastering machine learning with python in six steps: A practical implementation guide to predictive data analytics using python*. Apress, 2019.

[SWK09] Yanmin Sun, Andrew KC Wong, and Mohamed S Kamel. Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(04):687–719, 2009.

[SY19] Mucahid Mustafa Saritas and Ali Yasar. Performance analysis of ann and naive bayes classification algorithm for data classification. *International Journal of Intelligent Systems and Applications in Engineering*, 7(2):88–91, 2019.

[SYD$^+$14] Ge Song, Yunming Ye, Xiaolin Du, Xiaohui Huang, and Shifu Bie. Short text classification: A survey. *Journal of Multimedia*, 9:635–643, 2014.

[SZK$^+$17] Si Si, Huan Zhang, S Sathiya Keerthi, Dhruv Mahajan, Inderjit S Dhillon, and Cho-Jui Hsieh. Gradient boosted decision trees for high dimensional sparse output. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3182–3190. JMLR. org, 2017.

[SZLM08] Jiang Su, Harry Zhang, Charles X Ling, and Stan Matwin. Discriminative parameter learning for bayesian networks. In *Proceedings of the 25th international conference on Machine learning*, pages 1016–1023, 2008.

[TAL14] Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. *Data classification: Algorithms and applications*, page 37, 2014.

[TB13] Sri Fatimah Tjong and Daniel M Berry. The design of sree—a prototype potential ambiguity finder for requirements specifications and lessons learned. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 80–95. Springer, 2013.

[TKV05] Lucia Terrenghi, Marcus Kronen, and Carla Valle. Usability requirements for mobile service scenarios. *Human Computer Interaction*, pages 1–10, 2005.

[TM11] Nenad Tomasev and Dunja Mladenic. Nearest neighbor voting in high-dimensional data: Learning from past occurrences. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 1215–1218. IEEE, 2011.

[TM18] H Taud and JF Mas. Multilayer perceptron (mlp). In *Geomatic Approaches for Modeling Land Change Scenarios*, pages 451–455. Springer, 2018.

[TMHM16] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1):1–18, 2016.

[Tor09] Gessica Tortolano. Usability requirements template, 2009. Aviable at `https://sites.google.com/site/superuserfriendly/templates/usability-requirements-template`, last accessed September 2020.

[Tra16] David Travis. 247 web usability guideline, 2016. Available at `https://www.userfocus.co.uk/resources/guidelines.html`, last accessed October 2017.

[Tra18] Noam Tractinsky. The usability construct: a dead end? *Human–Computer Interaction*, 33(2):131–177, 2018.

[Tur01] Peter D Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *European conference on machine learning*, pages 491–502. Springer, 2001.

[UG14] Alper Kursat Uysal and Serkan Gunal. The impact of preprocessing on text classification. *Information Processing & Management*, 50(1):104–112, 2014.

[Usa] UsabilityNet. Recommended methods: 5. usability requirements. Avilable at `http://www.usabilitynet.org/trump/methods/recommended/requirements.htm`, last accessed September 2017.

[usa13] With measurable usability goals – we all score, 2013. Avilable at `https://www.usability.gov/get-involved/blog/2013/09/measurable-usability-goals.html`, last accessed October 2020.

[VG$^+$05] Anthony J Viera, Joanne M Garrett, et al. Understanding interobserver agreement: the kappa statistic. *Fam med*, 37(5):360–363, 2005.

[VR11] Radu Vlas and William N Robinson. A rule-based natural language technique for requirements discovery and classification in open-source software development projects. In *2011 44th Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2011.

[VS06] Sudhir Varma and Richard Simon. Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, 7(1):91, 2006.

[VWVDVE99] Martijn Van Welie, Gerrit C Van Der Veer, and Anton Eliëns. Breaking down usability. In *Interact*, pages 613–620, 1999.

[Wah12] Kathy Wahlbin. How to write user stories for web accessibilit, 2012.

[WB13] Karl Wiegers and Joy Beatty. *Software requirements*. Pearson Education, 2013.

[WC09] Mike Wasikowski and Xue-wen Chen. Combating the small sample class imbalance problem using feature selection. *IEEE Transactions on knowledge and data engineering*, 22(10):1388–1400, 2009.

[Wei04] Gary M Weiss. Mining with rarity: a unifying framework. *ACM Sigkdd Explorations Newsletter*, 6(1):7–19, 2004.

[Wei18] Paweł Weichbroth. Usability attributes revisited: a time-framed knowledge map. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1005–1008. IEEE, 2018.

[Wei20] Paweł Weichbroth. Usability of mobile applications: a systematic literature study. *IEEE Access*, 8:55563–55577, 2020.

[WG68] Martin B Wilk and Ram Gnanadesikan. Probability plotting methods for the analysis for the analysis of data. *Biometrika*, 55(1):1–17, 1968.

[WGV19] Jonas Paul Winkler, Jannis Grönberg, and Andreas Vogelsang. Optimizing for recall in automatic requirements classification: An empirical study. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 40–50. IEEE, 2019.

[WHYL12] Bing-kun Wang, Yong-feng Huang, Wan-xia Yang, and Xing Li. Short text classification based on strong feature thesaurus. *Journal of Zhejiang University SCIENCE C*, 13(9):649–659, 2012.

[Wil92] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[Wil99] William M Wilson. Writing effective natural language requirements specifications. *Naval Research Laboratory*, 1999.

[Wit07] Stephen Withall. *Software requirement patterns*. Pearson Education, 2007.

[WLL18] Tianlu Wang, Peng Liang, and Mengmeng Lu. What aspects do non-functional requirements in app user reviews describe? an exploratory and comparative study. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 494–503. IEEE, 2018.

[WLZ08] Jiao Wang, Si-wei Luo, and Xian-hua Zeng. A random subspace method for co-training. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 195–200. IEEE, 2008.

[Woh14] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10, 2014.

[WP94] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

[WRH97] William M Wilson, Linda H Rosenberg, and Lawrence E Hyatt. Automated analysis of requirement specifications. In *Proceedings of the 19th international conference on Software engineering*, pages 161–171, 1997.

[WW97] Dennis Wixon and Chauncey Wilson. The usability engineering framework for product design and evaluation. In *Handbook of human-computer interaction*, pages 653–688. Elsevier, 1997.

[WWD07] Sebastian Winter, Stefan Wagner, and Florian Deissenboeck. A comprehensive model of usability. In *IFIP International Conference on Engineering for Human-Computer Interaction*, pages 106–122. Springer, 2007.

[WWZY17] Jin Wang, Zhongyuan Wang, Dawei Zhang, and Jun Yan. Combining knowledge with deep convolutional neural networks for short text classification. In *IJCAI*, volume 350, 2017.

[WY12] Shuo Wang and Xin Yao. Multiclass imbalance problems: Analysis and potential solutions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(4):1119–1130, 2012.

[WY13] Shuo Wang and Xin Yao. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, 62(2):434–443, 2013.

[Xu18] Shuo Xu. Bayesian naïve bayes classifiers to text classification. *Journal of Information Science*, 44(1):48–59, 2018.

[Xue08] Nianwen Xue. Labeling chinese predicates with semantic roles. *Computational linguistics*, 34(2):225–255, 2008.

[YDRG$^+$11] Hui Yang, Anne De Roeck, Vincenzo Gervasi, Alistair Willis, and Bashar Nuseibeh. Analysing anaphoric ambiguity in natural language requirements. *Requirements engineering*, 16(3):163, 2011.

[Yev07] Ilyin Yevgeniy. Software requirements specification for project management system project, 2007.

[YGLC13] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. A biterm topic model for short texts. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1445–1456, 2013.

[YGX$^+$13] Liuzhi Yin, Yong Ge, Keli Xiao, Xuehua Wang, and Xiaojun Quan. Feature selection for high-dimensional imbalanced data. *Neurocomputing*, 105:3–11, 2013.

[YJGS20] Muhammad Younas, Dayang NA Jawawi, Imran Ghani, and Muhammad Arif Shah. Extraction of non-functional requirement using semantic similarity distance. *Neural Computing and Applications*, 32(11):7383–7397, 2020.

[YL99] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49, 1999.

[YL03] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 856–863, 2003.

[YP97] Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *Icml*, volume 97, page 35. Nashville, TN, USA, 1997.

[Yu97] Eric SK Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of ISRE'97: 3rd IEEE International Symposium on Requirements Engineering*, pages 226–235. IEEE, 1997.

[ZAF$^+$21] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J Letsholo, Muideen A Ajagbe, Erol-Valeriu Chioasca, and Riza T Batista-Navarro. Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing Surveys (CSUR)*, 54(3):1–41, 2021.

[ZBFK08] Arthur Zimek, Fabian Buchwald, Eibe Frank, and Stefan Kramer. A study of hierarchical and flat classification of proteins. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(3):563–571, 2008.

[Zho12] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[Zhu05] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. *University of Wisconsin-Madison Department of Computer Sciences*, 2005.

[Zie08] Peter Zielczynski. *Requirements Management Using IBM Rational RequisitePro*. IBM Press/Pearson plc, 2008.

[ZLLG18] Min-Ling Zhang, Yu-Kun Li, Xu-Ying Liu, and Xin Geng. Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, 12(2):191–202, 2018.

[ZLS$^+$18] Jichuan Zeng, Jing Li, Yan Song, Cuiyun Gao, Michael R Lyu, and Irwin King. Topic memory networks for short text classification. *arXiv preprint arXiv:1809.03664*, 2018.

[ZSM15] Shu Zhang, Samira Sadaoui, and Malek Mouhoub. An empirical analysis of imbalanced data classification. *Computer and Information Science*, 8(1):151, 2015.

[ZW15] Xinwei Zhang and Bin Wu. Short text classification based on feature extension using the n-gram model. In *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 710–716. IEEE, 2015.

[ZWL18] Lei Zhang, Shuai Wang, and Bing Liu. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1253, 2018.

[ZWS04] Zhaohui Zheng, Xiaoyun Wu, and Rohini Srihari. Feature selection for text categorization on imbalanced data. *ACM Sigkdd Explorations Newsletter*, 6(1):80–89, 2004.

[ZXY$^+$17] Jie Zou, Ling Xu, Mengning Yang, Xiaohong Zhang, and Dan Yang. Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis. *Information and Software Technology*, 84:19–32, 2017.

[ZYWS11] Wen Zhang, Ye Yang, Qing Wang, and Fengdi Shu. An empirical study on classification of non-functional requirements. In *The twenty-third international conference on software engineering and knowledge engineering (SEKE 2011)*, pages 190–195, 2011.

[ZZ20] Weijie Zheng and Hong Zhao. Cost-sensitive hierarchical classification for imbalance classes. *Applied Intelligence*, pages 1–11, 2020.

[ZZYZ21] Shengli Zhang, Fu Zhu, Qianhao Yu, and Xiaoyue Zhu. Identifying dna-binding proteins based on multi-features and lasso feature selection. *Biopolymers*, page e23419, 2021.

Appendix A

A Supplement for Chapter 3

A.1 Overview

This Appendix provides a supplement to Chapter 3: A systematic review of ML methods for identification and classification NFRs. It contains Table A.1, which lists the 51 studies, and Table A.2, which provides a comprehensive overview of the studies identified in the review. Besides, it contains tables (A.3, A.4, A.5, A.6) that used to generate figures (3.7, 3.8, 3.9, 3.11, and 3.10) to show which studies were involved clearly.

A.2 The 51 Selected Studies

Study ID	Authors Name	Year	Paper Title	Paper type	Publisher Venue	Publisher Name	DOI
S1	Cleland-Huang, Settimi, Zou & Solc	2007	Automated classification of non-functional requirements	Journ.	RE	Springer	https://doi.org/10.1007/s00766-007-0045-1
S2	Hussain, Kosseim & Ormandjieva	2008	Using linguistic knowledge to classify non-functional requirements in SRS documents	Conf.	International Conference on Application of Natural Language to Information Systems	Springer	https://doi.org/10.1007/978-3-540-69858-6_28
S3	Gokyer, Cetin, Sener & T. Yondem	2008	Non-functional requirements to architectural concerns: ML and NLP at crossroads	Conf.	International Conference on Software Engineering Advances	IEEE	10.1109/ICSEA.2008.28
S4	Casamayor, Godoy & Campo	2010	Identification of non-functional requirements in textual specifications: A semi-supervised learning approach	Journ.	Information and Software Technology	Science-Direct	https://doi.org/10.1016/j.infsof.2009.10.010
S5	Zhang, Yang, Wang & Shu	2011	An empirical study on classification of non-functional requirements	Conf.	International Conference on Software Engineering & Knowledge Engineering	Knowledge Systems Institute Graduate School [1]	N/A

[1] we accessed this paper via Semantic Scholar

Study ID	Authors Name	Year	Paper Title	Paper type	Publisher Venue	Publisher Name	DOI
S6	Knauss, Houmb, Schneider, Islam & Jürjens	2011	Supporting Requirements Engineers in Recognising Security Issues	Conf.	Requirements Engineering: Foundation for Software Quality	Springer	`https://doi.org/10.1007/978-3-642-19858-8_2`
S7	Hindle, Ernst, Godfrey. & Mylopoulos	2012	Automated topic naming	Journ.	Empirical Software Engineering	Springer	`https://doi.org/10.1007/s10664-012-9209-9`
S8	Rashwan, Ormandjieva, & Witte	2013	Ontology-based classification of non-functional requirements in software specifications: a new corpus and svm-based classifier	Conf.	Computer Software and Applications Conference	IEEE	`10.1109/COMPSAC.2013.64`
S9	Slankas and Williams	2013	Automated extraction of non-functional requirements in available documentation	Conf.-WS	Natural Language Analysis in Software Engineering	IEEE	`10.1109/NAturaLiSE.2013.6611715`
S10	Ott	2013	Automatic requirement categorization of large natural language specifications at Mercedes-Benz for review improvements.	Conf.	Requirements Engineering: Foundation for Software Quality	Springer	`https://doi.org/10.1007/978-3-642-37422-7_4`
S11	Riaz, King, Slankas & Williams	2014	Hidden in plain sight: Automatically identifying security requirements from natural language artifacts	Conf.	RE	IEEE	`10.1109/RE.2014.6912260`

Study ID	Authors Name	Year	Paper Title	Paper type	Publisher Venue	Publisher Name	DOI
S12	Knauss and Ott	2014	(Semi-) automatic Categorization of Natural Language Requirements	Conf.	Requirements Engineering: Foundation for Software Quality	Springer	`https://doi.org/10.1007/978-3-319-05843-6_4`
S13	Nguyen, Grundy, Almorsy	2015	Rule-based extraction of goal-use case models from text	Conf.	European Software Engineering Conference and Symposium on the Foundations of Software Engineering	ACM	`https://doi.org/10.1145/2786805.2786876`
S14	Maiti, Mitropoulos	2015	Capturing, eliciting, predicting and prioritizing (CEPP) non-functional requirements metadata during the early stages of agile software developmen	Conf.	Southeastcon	IEEE	`10.1109/SECON.2015.7133007`
S15	Mahmoud and Williams	2016	Detecting, classifying, and tracing non-functional software requirements	Journ.	RE	Springer	`https://doi.org/10.1007/s00766-016-0252-8`
S16	Jindal and Malhotra	2016	Automated classification of security requirements	Conf.	International Conference on Advances in Computing, Communications and Informatics	IEEE	`10.1109/ICACCI.2016.7732349`

Study ID	Authors Name	Year	Paper Title	Paper type	Publisher Venue	Publisher Name	DOI
S17	Malhotra, Chug, Hayrapetian & Raje	2016	Analyzing and evaluating security features in software requirements	Conf.	Innovation and Challenges in Cyber Security	IEEE	`10.1109/ICICCS.2016.7542334`
S18	Singh, Singh, Sharma	2016	Rule-Based System for Automated Classification of Non-Functional Requirements from Requirement Specifications	Conf.	International Conference on Advances in Computing, Communications and Informatics	IEEE	`10.1109/ICACCI.2016.7732115`
S19	Lu and Liang	2017	Automatic Classification of Non-Functional Requirements from Augmented App User Reviews	Conf.	The Evaluation and Assessment in Software Engineering Conference	ACM	`https://doi.org/10.1145/3084226.3084241`
S20	Deocadez, Harrison, Rodriguez	2017	Automatically Classifying Requirements from App Stores: A Preliminary Study	Conf.-WS	RE Workshops	IEEE	`10.1109/REW.2017.58`
S21	Li, Huang, Ge, Luo, Ng	2017	Automatically classifying user requests in crowdsourcing requirements engineering	Journ.	Journal of Systems and Software	Science-Direct	`https://doi.org/10.1016/j.jss.2017.12.028`
S22	Kurtanović and Maalej	2017	Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning	Conf.	RE	IEEE	`10.1109/RE.2017.82`

Study ID	Authors Name	Year	Paper Title	Paper type	Publisher Venue	Publisher Name	DOI
S23	Abad, Karras, Ghazi,Glinz, Ruhe, Schneider	2017	What works better? a study of classifying requirements	Conf.	RE	IEEE	`10.1109/RE.2017.36`
S24	Zou,Xu, Yang, Zhang, Yang	2017	Towards comprehending the non-functional requirements through Developers' eyes: An exploration of Stack Overflow using topic analysis	Journ.	Information and Software Technology	Science-Direct	`https://doi.org/10.1016/j.infsof.2016.12.003`
S25	Navarro-Almanza, Juarez-Ramirez, Licea	2017	Towards supporting software engineering using deep learning: A case of software requirements classification	Conf.	International Conference in Software Engineering Research and Innovation	IEEE	`10.1109/CONISOFT.2017.00021`
S26	Dekhtyar and Fong	2017	Re data challenge: Requirements identification with word2vec and tensorflow	Conf.	RE	IEEE	`10.1109/RE.2017.26`
S27	Munaiah, Meneely, Murukannaiah	2017	A Domain-Independent Model for Identifying Security Requirements	Conf.	RE 20	IEEE	`10.1109/RE.2017.79`
S28	Tóth, Vidács	2018	Study of Various Classifiers for Identification and Classification of Non-functional Requirements	Conf.	International Conference on Computational Science and Its Applications	Springer	`https://doi.org/10.1007/978-3-319-95174-4_39`

Study ID	Authors Name	Year	Paper Title	Paper type	Publisher Venue	Publisher Name	DOI
S29	Marinho, Arruda, Wanderley, Lins	2018	A Systematic Approach of Dataset Definition for a Supervised Machine Learning Using NFR Framework	Conf.	International Conference on the Quality of Information and Communications Technology	IEEE	10.1109/QUATIC.2018.00024
S30	Hakim and Rochimah	2018	Oversampling Imbalance Data: Case Study on Functional and Non Functional Requirement	Conf.	Electrical Power, Electronics, Communications, Controls and Informatics Seminar	IEEE	10.1109/EECCIS.2018.8692986
S31	Amasaki	2018	The Effects of Vectorization Methods on Non-Functional Requirements Classification	Conf.	Euromicro Conference on Software Engineering and Advanced Applications	IEEE	10.1109/SEAA.2018.00036
S32	Alhindawi	2018	Information Retrieval - Based Solution for Software Requirements Classification and Mapping	Conf.	International Conference on Mathematics and Computers in Sciences and Industry	IEEE	10.1109/MCSI.2018.00042
S33	Wang, Zhang, Liang, Daneva, Sinderen	2018	Can App Changelogs Improve Requirements Classification from App Reviews? An Exploratory Study	Conf.	International Symposium on Empirical Software Engineering and Measurement	ACM	https://doi.org/10.1145/3239235.3267428

Study ID	Authors Name	Year	Paper Title	Paper type	Publisher Venue	Publisher Name	DOI
S34	Dalpiaz, Dell'Anna, Aydemir	2019	Requirements Classification with Interpretable Machine Learning and Dependency Parsing	Conf.	RE	IEEE	`10.1109/RE.2019.00025`
S35	Abad, Gervasi, Zowghi, Far	2019	Supporting Analysts by Dynamic Extraction and Classification of Requirements-Related Knowledge	Conf.	International Conference on Software Engineering	IEEE	`10.1109/ICSE.2019.00057`
S36	Raharja and Siahaan	2019	Classification of Non-Functional Requirements Using Fuzzy Similarity KNN Based on ISO / IEC 25010	Conf.	International Conference on Information & Communication Technology and System	IEEE	`10.1109/ICTS.2019.8850944`
S37	Younas, Jawawi,Ghani, Shah	2019	Extraction of non-functional requirement using semantic similarity distance	Journ.	Neural Computing and Applications	Springer	`https://doi.org/10.1007/s00521-019-04226-5`
S38	Baker, Deng, Chakraborty, Dehlinger	2019	Automatic multi-class non-functional software requirements classification using neural networks	Conf.	Computer Software and Applications Conference	IEEE	`10.1109/COMPSAC.2019.10275`
S39	Haque, Rahman, Siddik	2019	Non-Functional Requirements Classification with Feature Extraction and Machine Learning: An Empirical Study	Conf.	International Conference on Advances in Science, Engineering and Robotics Technology	IEEE	`10.1109/ICASERT.2019.8934499`
S40	Rahman, Haque, Tawhid, Siddik	2019	Classifying non-functional requirements using RNN variants for quality software development	Conf.-WS	Machine Learning Techniques for Software Quality Evaluation	ACM	`https://doi.org/10.1145/3340482.3342745`

Study ID	**Authors Name**	**Year**	**Paper Title**	**Paper type**	**Publisher Venue**	**Publisher Name**	**DOI**
S41	Taj, Arain, Memon, Zubedi	2019	To apply Data Mining for Classification of Crowd sourced Software Requirements	Conf.	International Conference on Software and Information Engineering	ACM	`https://doi.org/10.1145/3328833.3328837`
S42	Gilson, Galster, Georis	2019	Extracting Quality Attributes from User Stories for Early Architecture Decision Making	Conf.	International Conference on Software Architecture Companion	IEEE	`10.1109/ICSA-C.2019.00031`
S43	Bhowmik and Do	2019	Refinement and resolution of just-in-time requirements in open source software and a closer look into non-functional requirements	Jour.	Journal of Industrial Information Integration	Science-Direct	`https://doi.org/10.1016/j.jii.2018.03.001`
S44	Jha and Mahmoud	2019	Mining non-functional requirements from app store reviews	Journ.	Empirical Software Engineering volume	Springer	`https://doi.org/10.1007/s10664-019-09716-7`
S45	Wang, Mahakala, Gupta, Hussein, Wang	2019	A linear classifier based approach for identifying security requirements in open source software development	Jour.	Journal of Industrial Information Integration	Science-Direct	`https://doi.org/10.1016/j.jii.2018.11.001`
S46	Palacio, McCrystal, Moran	2019	Learning to Identify Security-Related Issues Using Convolutional Neural Networks	Conf.	IEEE International Conference on Software Maintenance and Evolution	IEEE	`10.1109/ICSME.2019.00024`
S47	Hey, Keim, Koziolek, Tichy	2020	NoRBERT: Transfer learning for requirements classification	Conf.	RE	IEEE	`10.1109/RE48521.2020.00028`

Study ID	Authors Name	Year	Paper Title	Paper type	Publisher Venue	Publisher Name	DOI
S48	Li and Chen	2020	An ontology-based learning approach for automatically classifying security requirements	Journ.	Journal of Systems and Software	Science-Direct	`https://doi.org/10.1016/j.jss.2020.110566`
S49	Kobilica, Ayub, Hassine	2020	Automated Identification of Security Requirements: A Machine Learning Approach	Conf.	Evaluation and Assessment in Software Engineering	ACM	`https://doi.org/10.1145/3383219.3383288`
S50	Canedo and Mendes	2020	Software Requirements Classification Using Machine Learning Algorithms	Jour.	Entropy	MDPI	`https://doi.org/10.3390/e22091057`
S51	Shreda and Hanani	2021	Identifying Non-functional Requirements from Unconstrained Documents using Natural Language Processing and Machine Learning Approaches	Journ.	IEEE Access	IEEE	`0.1109/ACCESS.2021.3052921`

Table A.1: The 51 selected studies

A.3 A Comprehensive Overview of the 51 Studies

ID.	Main Contribution	Technique	Dataset	Evaluation Method
S1	Used a probability-based method (i.e., supervised learning) in NFR classification.	Extracted "indicator terms" of each NFR category from a training dataset. These terms are then used to compute the likelihood of an input requirement belonging to a specific NFR type. The likelihood score is compared against a threshold; if it above, the input requirement will be classified into a specific NFR type; otherwise, it will be classified as FR.	Two different datasets: academic (small and large) and industrial dataset. The small academic dataset contains 15 student term projects (684 requirements), Known as PROMISE. The large dataset has 30 student project. The industrial dataset contains 2,064 sentences.	Used 5-fold cross-validation to evaluate the classification precision and recall using different numbers of features, data sizes, and domains (industrial vs academic).
S2	Used linguistic analysis (POS) to select features.	Used DT (C4.5) to classify requirements into F and NFR. The features were selected based on the probability of the occurrence of their POS groups in NFR sentences.	PROMISE	Used 10-fold cross-validation and train-test-split (holdout) to evaluate the method through TP, FP, precision and recall. Additionally, compared the method with the Cleland-Huang et al. method.

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S3	Used the extracted NFRs to construct the Utility Concern Spaces	Used SVM to classify NFRs, which then used to construct the Utility Concern Spaces (a matrix for the correlation of architectural aspects [AA] and quality attributes [QA])	N/A	Compared the accuracy of matrix produced by the method (cells) to the one generated by an expert system architect.
S4	Used a semi-supervised classifier in NFR classification.	Used a semi-supervised classifier (MNB and Expectation Maximization strategy) for classifying NFRs.	PROMISE	Used 10-fold cross-validation to evaluate the classification accuracy, recall, precision, and F-measure. Additionally, compared the proposed method with supervised algorithms (KNN, and Rocchio algorithm).
S5	Investigated different index terms to find the most appropriate one for NFR classification	Compared to the performance of different index terms (N-grams, individual words, and multi-word expressions) with information gain and SVM.	PROMISE	Used a 10-fold cross-validation technique to measure precision and recall of each NFR type. Additionally, compared their results with Cleland-Huang et al. work.

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S6	Investigated the feasibility of automatically identifying security-relevant requirements using a NB classifier.	Used NB classifiers to identify security-relevant requirements.	SecReq is an available security requirements dataset consisting of 510 requirements (187 are security) from three industrial requirements documents.	Used 10-fold cross-validation to mesure precision, recall, and f-mesure of the method. Additionally, compared the performance in different domains and with the baseline, which is classifying all requirements as security requirements.
S7	Proposed a cross-project topic extraction method to extract and classify topics (themes) according to NFR type.	Extracted topics from commit-log comments using LDA. Additionally, used supervised learning (NB) and semi-unsupervised learning (through three lists of domain-independent keywords) to classifying the extracted topic according to NFR labels.	Commit comments of three open-source database systems (MySQL, MaxDB, PostgreSQL)	Used 10-fold cross-validation to measure F-measure and ROC for semi-unsupervised and (single-label, multi-label) supervised classifiers.
S8	Build an annotated gold standard NFRs dataset based on a requirements ontology (known as Concordia dataset).	Developed a new dataset based on a requirements ontology. Used SVM to to build and test a NFRs classifier.	Concordia dataset (1021 Requirements from 6 different sources) and PROMISE	Used 6-fold cross-validation with Precision, Recall, F-Measure, and confusion matrices. Additionally, compared their classifier (SVM) with previous work (Cleland-Huang et al.).

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S9	Modified the distance metric (levenshtein distance) of KNN in NFRs classification.	Modified KNN to extract NFRs from different NL documents (i.e., data use agreements, install manuals, regulations, request for proposals, requirements specifications, and user manuals).	11 documents related to electronic health records and PROMISE for providing comparisons to prior research.	Used 10-fold cross-validation for measuring the *P*, *R*, and F1-score. Additionally, compared different ML algorithms' performance to find the best (SVM, NB, modified KNN). Compared the performance of different pre-processing techniques (original, lema, stop word removal, and Casamayor et al.'s technique) with different ML algorithms (NB and SVM).
S10	Extracted and classified requirements from large industrial documents	Extracted and classified requirements form large industrial documents (3,000 pages) using SVM and NB. Used "topic generalization" and "recall+" for post-processing data to correctly assign requirement to the topic.	Two German automotive specifications of Mercedes-Benz (one contains 1087 and the other has 2385 requirements).	Used 10-fold cross-validation and computed the precision and recall. Additionally, compared the performance of the whole proposed method against applying each pre-processing and post-processing technique separately using different ML algorithms (SVM and NB).

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S11	Extracted security-relevant sentences and classify them according to security objectives. Translated the extracted sentences into functional security requirements using requirement templates.	Used KNN (Levenshtein distance) to detect and classify security requirements according to security objectives (such as confidentiality, integrity, availability). Additionally, the extracted requirements are rewritten using requirements templates.	10,963 sentences extracted from six natural language artefacts from the electronic healthcare domain.	Assessed the effectiveness of the classifier using precision, recall, F1-score with 10-fold cross-validation. Additionally, compared the performance of multiple ML algorithms (SVM, KNN, MNB, and combined method—Which applies the three algorithms with a majority voting technique).
S12	Investigated the involvement of the user in classifying NFRs to overcome the lack of sufficient training data.	Built a requirements classifier with three modus: manual, semi-automatic, and fully-automatic modus. These models are categorized based on user involvement in classification decisions. SVM was used for the automatic and semi-automatic modus.	2,000 requirements from Mercedes-Benz specification	Used recall, precision, and F1-score to compare all modes' performance against ground truth (i.e., manual expert classification). Additionally, measured the trust in automated classification through questioners, while the effort and reliability of semi-automatic classification are measured based on user choices and questionnaires.

ID.	Main Contribution	Technique	Dataset	Evaluation Method
S13	Developed a rule-based method that automatically extracts and classifies goal-use case models from NL documents.	Developed rule-based method to extract use-case goals from NL requirement documents and ML algorithms to classify the goals as (business, functional or non-functional goals).	PROMISE	Used precision and recall with 10-fold cross-validation to assess the classifier performance and compare it with other works (e.g. Casamayor et al. work and original version of Mallet).
S14	Captured and classified NFRs from multiple sources (images and informal documents) during the early stages of software development.	Used KNN to classify requirments captured from multiple sources (images and informal documents).	N/A	N/A
S15	Used unsupervised learning in classifying NFR requirements. Additionally, used semantic similarity between words to link source code with the extracted NFRs.	Used unsupervised learning (cluster techniques and semantic similarity method) in classifying NFR requirements.	568 requirements extracted from three different requirements documents in different application domains (Smartrip, Safe Drink, BlueWallet).	Used precision and recall to measure the performance of the classifier in each system's specifications.
S16	Classified security requirements into four further classes.	Used DT and information gain to classify security requirements into their respective security types.	58 security requirements from PROMISE	Used Sensitivity and ROC with 10-fold cross-validation to assess the classifier performance.

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S17	Used machine learning to classify security requirements into four security features. Additionally, checked the completeness and consistency of the classified requirements.	Classified security requirements according to security features (e.g., authentication, encryption, etc.). Additionally, generated a concept graph for each feature to check various properties of requirements (e.g., consistency, completeness, etc.).	PROMISE	N/A
S18	Used thematic roles and fit criteria to build a rule-based classifier and prioritized the extracted requirement.	Extracted thematic roles and fit criteria from requirements to classify them into NFR sub-classes with a rule-based classifier. Additionally, prioritized the extracted requirement based on their occurrence within a document.	PROMISE and Concordia	Used precision, recall and F1-score to assess the classifier which is trained and test using PROMISE dataset and verified using Concordia dataset.

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S19	Compared a new feature extension technique using word-embedding and BOW with other exciting techniques.	Augmented users review with most related words and used bagging to classify the reviews based on NFR categories	4000 user review sentences collected form tow Apps (iBooks and WhatsApp)	Used Recall, Precision, F1-score to assess the performance of the classifier. Compared the classifier performance with different features representation techniques (BOW, TF-IDF, Chi-Squared) and ML algorithms (NB, DT, and Bagging).
S20	Used different semi-supervised methods to classify NFRS	Applied semi-supervised learning algorithms (Self-Training, RASCO, Rel-RASCO) to classify user reviews as functional or non-functional.	300 reviews collected from 40 apps in the app store	Used accuracy metric (inductive and transductive accuracy) to evaluate the performances of the three semi-supervised learning algorithms which are separately executed with four ML algorithms (NB, DT, SVM, and KNN) in different ratios of labelled data.

ID.	Main Contribution	Technique	Dataset	Evaluation Method
S21	Classified user request according to requirements categories.	Applied semi-supervised learning (using active learning) to classify user requests into different NFR categories. Two lists of keywords (project-specific and non-project-specific keywords) are used by the classifier.	3000 user requests from three projects in sourceforge.net (KeePass, Mumble, WinMerge).	Used P, R, F1-score and accuracy with 5-fold cross-validation to assess the classifier performance in each project separately. The assessment includes comparing different representation techniques (word unigram and TF-IDF), lists of keywords (non-project-specific, project-specific, term frequency and unigrams), and ML algorithms (KNN, NB, and SVM).
S22	Extracted different types of features. Used dataset derived from user comments\reviews to handle the class imbalance problem (over-sample minority class).	Used SVM with different types of features in identifying and classifying NFRs. Additionally, applied various re-sampling strategies using a hybrid dataset (requirements statements and user reviews) to handle the imbalanced class problem in NFRs classification.	PROMISE and user reviews (Amazon software reviews)	Applied 10-fold cross-validation to assess the performance of the classifier using P, R, $F1$-score. Compared the performance of different classifiers built using different proportions of features, learning tasks (binary and multi-class classifiers), and re-sampling techniques to handle the imbalance problem.

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S23	Proposed a pre-processing technique that standardizes and normalizes requirements before applying classification algorithms. Further, compared machine learning methods in automatically classifying NFRs (e.g., clustering, NB, topic model)	Used DT with a proposed pre-processing technique (rule-based method) for standardizing and normalizing the requirements. Further, investigated the performance of several ML methods in classifying NFRs.	PROMISE	Performed a 10-fold-cross validation to measure FR/NFR classifier's performance in terms of *P*, *R*, *F*1-score, Kappa, number of correctly and not correctly classified requirements in relation to baseline (un-preprocessed requirements). Used *P* and *R* with 5-fold-cross validation to compare various ML algorithms (LDA, BTM, hierarchical, K-means, Hybrid, MNB) in classing NFRs into further 10 categories.

ID.	Main Contribution	Technique	Dataset	Evaluation Method
S24	Analyzed developers' discussions (via StackOverflow) based on NFR types.	Used LDA to extract topics that are then annotated according to NFRs types using pre-defined word lists. Additionally, analyzed the extracted topics to explore hot, unresolved, or trendy NFR categories.	21.7m posts and 32.5 m comments from Stack Overflow	Measured recall rate and precision rate aginst a manual validation for each period (month). Additionally, the classifier results were used to analyze NFRs types (e.g., hot NFRs by assessing topic frequency, unresolved NFRs by analyzing unanswered questions, and difficult NFR by measuring successfully answered questions).
S25	Used deep learning in requirement classification and word2vec for feature representation.	Used word2vec to represent textual features and CNN to classify requirement.	PROMISE	Performed 10 cross-validations and averaged precision, recall, f-measure, and confusion matrix.

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S26	Used CNN in binary requirements classification and word2vec for requirements representation	Used CNN classifier with Word2Vec representation in binary requirements classification.	SecRe + PROMISE	Applied 10-fold cross-validation with accuracy, precision, recall and F2-score to evaluate the CNN classifier in each dataset separately. Additionally, compared the classifier performance with baseline (NB), which separately applied two different features selection techniques (word count and TF-IDF).
S27	Used domain-independent data sets to identify security requirements using an unsupervised classifier (One-Class SVM).	Used One-Class SVM model, which trained on domain-independent data sets to identify security requirements.	Two available, unlabeled datasets (i.e., National Vulnerability Database and Common Weakness Enumeration database) for training and SecRet for testing.	Evaluated using 10-fold cross-validation and reported a weighted average of precision, recall and F-score for each specification in SecRec (i.e., CPN, ePurse and GPS). Additionally, compared the classifier with the NB classifier in their previous work (S6).

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S28	Evaluated the performance of *sklearn* classifiers in NFRs classification (execution time and classification performance).	Compared the performance of various classification methods (e.g. NB, DT, KNN, and SVM) in sklearn library.	PROMISE	Performed 10 -fold cross-validation with variance, precision, recall, and f1-score to measure the sklearn classifiers' performance in terms of classification results and execution time.
S29	Used keywords extracted from SIG (Softgoal Interdependency Graph) catalogues in NFR classification.	Applied Stochastic Gradient Descent (SGD) algorithm to classify requirement into three categories (security, performance and usability). Additionally, used a set of keywords obtained from SIG catalogues (through a mapping study) and from thesaurus dictionary using Visuwords [2].	Part of Promise (187 sentences)	Assessed precision, recall, f1-score for both binary and multi-class classification. Compared the performance of the classifiers with and without the keywords extracted from the thesaurus dictionary.

[2] `https://visuwords.com/` last accessed March 2021

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S30	Automated the process of the labeling requirements and use SMOTE for the imbalanced-class problem.	Used supervised learning (SVM, KNN and Random Forests) to build multi-label NFR classifiers. The labels were automatically generated using WordNet and cosine similarity. SMOTE was applied to handle the imbalanced class problem.	1366 requirement from two different datasets (Waterloo and Geolocation).	Measured the accuracy for each ML algorithm with and without using SMOTE.
S31	Investigated the performance of using different vectorization (representation) methods in NFRs classification	Used supervised learning (Logistic Regression, NB, SVM, and Random Forests) with different vectorization methods (TF, TF-IDF, word2vec, Doc2vec, SCDV) in binary NFRs classification.	PROMISE	Performed out-of-sample bootstrap validation method to measure precision, recall and f1-score. Compared the performance of each vectorization technique with each ML algorithm.
S32	Used LDA in requirement classification and mapping—linking NFR and FR with system requirements.	Used LDA to extract topics from requirements. Used the extracted topics for requirement classification.	200 requirements from WARC (Web ARChive) system	Used accuracy to assess the classification results.

ID.	Main Contribution	Technique	Dataset	Evaluation Method
S33	Used app change-logs to classify app reviews according to NFRs classes.	Used app changelogs for training NB in classifying user reviews into NFRs classes.	6000 sentences in user reviews collected from iBooks (in Apple App Store), WhatsApp and TripAdvisor (in Google Play). Additionally, 2005 app change-logs collected from 30 apps in the App Store	Performed holdout method (80%-20%) to measure precision, recall and f1-score. Additionally, compared the performance of the classifiers with KNN, Bagging and DT.
S34	Used interpretable ML techniques to select features in binary requirement classification.	Used SVM with linguistic features to classify requirements into functional and quality aspects. The linguistic features were extracted using statistical co-occurrence analyze and interpretable ML techniques.	1502 requirements collected from 8 datasets: PROMISE, ESA Euclid dataset, Helpdesk system, a user management application, Dronology dataset for Unmanned Aerial Systems, ReqView for requirements management tool, Leeds University's Library online management system, Web Architectures for Services Platforms (WASP).	Performed hold-out method with precision, recall, F1 score, accuracy, and ROC to validate and test the classifier performance. Additionally, compared the classifier performance with the one proposed by Kurtanović and Maalej. In this compression, more evaluation methods were used, including p-fold, k-fold, train fitness.
S35	Used external knowledge to extract n-gram, which is used with SVM to classify requirements.	Used statistical language models and external knowledge to extract relevant terms in requirements documents. Additionally, applied SVM with ration kernel to classify requirements.	PROMISE and two more datasets for different application domains (Ticketing system and environmental and road conditions monitor system).	Assessed the performance of the classifier using standard metrics precision, recall, and F score. Compared the performance of different length of n-gram and cost in non-contiguous n-gram kernels.

ID.	Main Contribution	Technique	Dataset	Evaluation Method
S36	Used Fuzzy Similarity KNN (FSKNN) in NFRs classification	Used fuzzy similarity measure with KNN to classify NFRs according to ISO / IEC 25010 categories.	1342 requirements from different datasets (including PROMISE)	Perform holdout method to measure accuracy, precision, and recall in different similarity threshold values.
S37	Used word2vec in NFR classification (unsupervised learning).	Used word2vec to measure the similarity between requirement statements and NFR classes. Each NFR class was represented by a set of indicator terms taken from the literature.	931 requirements collected from PROMISE and available CCHIT ambulatory dataset.	Measured TF, FP, FN, P, R, F-score for each dataset to assess the classifier performance. Compared the classifier's performance with and without apply pre-processing techniques.
S38	Used two kinds of neural network models ANN and CNN to classify NFRs.	Used two kinds of neural network models (ANN and CNN) to classify NFRs into four classes (operability, performance, security, and usability)	PROMISE	Performed 10-cross validation to measure precision, recall, F1-score for each class. Reported the results achieved by applying each classifier (CNN and ANN) separately.
S39	Empirical comparison among feature extraction and classification techniques.	An empirical comparison of four feature extraction methods (BOW, character-level, word-level and n-gram level of TF-IDF technique) with seven machine learning algorithms (NB, GNB, BNB, KNN, SVM, SGD SVM, and DT) in classifying NFRs.	PROMISE	Used precision, recall, and F1-score to perform the empirical comparison of different representation technique and classification algorithms.

ID.	Main Contribution	Technique	Dataset	Evaluation Method
S40	Used RNN model to classify NFRs	Used RNN model to classify NFRs with word2vec algorithm for features representation.	PROMISE	Performed 10-fold cross-validation to measure precision, recall and F-score. Additionally, compared the performance of RNN (LSTM) with CNN, RNN(GRU) for classifying NFRS into 10 classes.
S41	Collected data using crowd-sourcing approach[3] and used it in NFRs classification.	Applied DT to classify requirements into FR and NFR using a dataset collected using a crowdsourcing approach.	Requirements collected from 118 people using a crowdsourcing approach.	Performed the holdout method (60%-40%) to measure precision, recall and accuracy. Compared the performance of DT with NB algorithms.
S42	Extracted and classified NFRs from user stories in agile development projects.	Used CNN to extract and classify NFRs from user stories in agile development projects.	1,675 stories collected from an available user stories dataset [4], annotated into 7 NFRs classes.	Performed 10-fold cross-validation and holdout (70%-30%) methods separately to measure precision, recall, and f-score for each class. Additionally, reported the results obtained for different decision thresholds of CNN (0.5 and 0.6).

[3]Using a large number of people in solving a distributed problem. The people (participants) are invited through an open call.

[4] `https://data.mendeley.com/datasets/7zbk8zsd8y/1` last accessed 23 Feb 2021

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S43	Proposed a refinement and resolution process for Just in Time (JIN) requirements[5].	Used LDA with a pre-defined wordlist to extra NFRs from JIT requirements. The extracted requirements are used in the investigation of the suitability of a proposed refinement and resolution process for NFRs.	7698 requirements from open-source software projects (Firefox, Lucene, and Mylyn).)	Manually: one of the authors filtered the correct predictions.
S44	Conducted qualitative analysis to determine the presence of NFR categories in user reviews over different application domains. Investigated the performance of ML in classifying user reviews into different NFRs. Proposed dictionary-based method to classify user reviews into different NFRs	Applied Binary Relevance(BR) with different ML algorithms (i.e., SVM and NB) to classify user reviews into NFRs categories (multi-label classification). Used domain names (i.e., app category) and sentiment analysis score as classification features.	6,000 reviews sampled from 24 different iOS mobile apps.	Performed holdout method (70%-30%) to measure Subset Accuracy, Hamming Score, Hamming Loss, Recall, Precision, and F2-Measure. Additionally, compared the performance of the classifiers with different classification features and pre-processing techniques.

[5]The requirements that focus on elaboration when the implementation begins. These requirements are refined frequently by developers through, for example, writing comments.

ID.	Main Contribution	Technique	Dataset	Evaluation Method
S45	Applied a linear classifier to detect security requirements in open source software projects (JIT RE). Used a new feature extraction method based on logistic regression models.	Used linear classifier to detects security requirements. The features were calculated using five regression models: one applied Cleland-Huang method (S1), and each model of the remaining four models computes the probability of security requirement based on the proposed metrics (i.e., features): complicity (e.g., length and user involvement) and external resources(commits and URLs).	4249 requirements collected from 3 open-source software projects: web services engine (Axis2), business rule management system (Drools), and a geographic system (GeoServer).	Performed 10-fold cross-validation to measure P, R, F2-score of the proposed classifier. Compared the performance of the proposed classifier as a whole in different threshold values and against the performance of each sub-regression model.
S46	Use CNN for detecting security issues in tracking systems.	Use CNN to classify requirements (issues) into security and non-security.	96,614 requirements selected from different sources, including popular projects on GitLab or GitHub and Cisco Systems.	Performed holdout method with test set loss, accuracy, and AUC to assess the classifier performance. Compared the classifier's performance with different datasets (open source issues and industrial user stories) and different CNN architecture configuration.

ID.	Main Contribution	Technique	Dataset	Evaluation Method
S47	Used BERT in NFRs classification.	Used BERT in different NFRs classification tasks.	PROMISE and a relabeled version of PROMISE provided by Dalpiaz et al. (S34)	Performed Holdout (75%-25%), 10-fold cross-validation, p-fold and a leave-one-project-out cross-validation (loPo) methods in measuring precision, recall and F-score for each class. Evaluated the classifier with different datasets and classification tasks (binary and multi-class classification).
S48	Used ontological information in defining linguistic features of security requirements (linguistic rules and keywords).	Used DT to identify security requirements. Extracted linguistic features by defining a set of security requirements linguistic rules and keywords.	PROMISE, and SeqReq	Performed 10 cross-validations with precision, recall, and f-measure to assess the classifier performance. Compared the classifier performance against two different ML algorithms (NB and Logistic Regression) and with the classifier proposed by Knauss et al. (S6). Applied haulout to assess the classifier's generalizability by applying it in another domain.

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S49	Investigated the use of RNN in security requirement classification. Empirically compared different ML algorithms in security requirement classification.	Conducted empirical comparison of the performance of eight different ML classifiers with two text representation techniques (word encoding and word embedding) in classifying security requirements.	SecRe	Performed 10 cross-validations with accuracy and execution time to assess the performance of the classifiers. Showed the ROC curve of the classifiers that have the highest results (e.g., ensemble-based boosted tree).
S50	Empirically compared feature selection techniques with ML algorithms in NFRs classification with a recently released dataset.	Compared different feature selection and representation techniques (BoW, TF-IDF and CHI with TF-IDF) with different ML algorithms (LR, SVM, NB, KNN) in NFRs classification.	PROMISE_exp	Performed 10 cross-validations with precision, recall, and f1-score to assess the performance of the different classifiers.

ID.	**Main Contribution**	**Technique**	**Dataset**	**Evaluation Method**
S51	Proposed a Fusion model that represents an input requirement by combining its classification scores from four different CNN classifiers, each of which has a different representation method. The new representation is used in NFRs classification with a logistic regression model.	Built four CNN classifiers, each of which uses different representation techniques (TF, TF-IDF, word2vec, BERT) to represent an input requirement. Used logistic regression with the new representation to classify the requirement based on NFRs class.	1846 requirement sentences from PURE dataset [6]	Used holdout method (70%-30%) with accuracy, precision, recall, and f-score to assess the classifier performance. Compared to the performance of four classifiers (CNN, NB, SVM, LR) with four representation techniques (TF, TF-IDF, w2v, BERT) separately to use the best combination in the proposed method.

Table A.2: A comprehensive overview of the included 51 studies in the systematic review

[6]`https://zenodo.org/record/1414117#.YDPaFJP7RE4` last accessed 22 Feb 2012

A.4 Detailed Information about the Studies

No	Size	No.classes	No studies	Studies IDs	Classes
1	58	4	1	S16	Authentication-authorization, Access control, cryptography-encryption and data integrity
2	171	3	1	S5	Usability, Security,and Look And Feel
3	183	3	1	S35	Usability, Operational, Performance
4	187	3	1	S29	Security, Performance and Usability
5	200	2	1	S32	FN vs NFRs
6	246	2	1	S31	Operational vs non-Operational, Performance vs non-Performance, Security vs non-Security, Usability vs non-Usability
7	300	2	1	S20	FR, NFRs
8	306	2	1	S37	Fn vs NFRs
9	370	11	1	S13	Usability, Security, Operational, Performance, Look and Feel, Availability, Scalability, Maintainability, Legal, Fault Tolerance, and Portability
10	369	10	4	S23,S28, S40, S47	Usability, Security, Operational, Performance, Look and Feel, Availability, Scalability, Maintainability, Legal, Fault Tolerance
11	510	2	3	S6, S26, S27, S510, S49	Security vs Non-security
13	568	12	1	S15	Security, Performance, Accessibility, Accuracy, Portability, Safety, Legal, Privacy, Reliability, Availability, Interoperability, other

No	Size	No.classes	No studies	Studies IDs	Classes
13	625	10	1	S4	Availability, Legal, Look and feel, Maintainability, Operational ,Performance, Scalability, Security, Usability, Features
14	625	2	4	S4, S22, S23, S26, S31, S47, S48	FN vs NF or Sec vs non-sec (S48)
15	625	12	2	S25, S37	Functional, Availability, Legal, Look and feel, Maintainability, Operational, Performance, Scalability, Security, Usability, Fault tolerance, and Portability
16	684	10	1	S1	Availability, Legal, Look and feel, Maintainability, Operational,Performance, Scalability, Security, Usability, Functional
17	765	2	1	S2	FN vs NFR
18	914	5	1	S38	maintainability, operability, performance, security, and usability.
19	969	2	1	S50	FN vs NFR
20	969	11	1	S50	Availability, Legal, Look & Feel, Maintainability, Operability, Performance, Scalability, Security, Usability, Fault Tolerance, Portability
21	969	12	1	S50	Functional, Availability, Legal, Look & Feel, Maintainability, Operability, Performance, Scalability, Security, Usability, Fault Tolerance, Portability
22	1,342	7	1	S36	Reliability, performance efficiency, operability, security, compatibility, maintainability, transferability

No	Size	No.classes	No studies	Studies IDs	Classes
23	1,366	12	1	S30	Time Behavior, Resource Utilization, Capacity, User Error Protection, Aesthetic User Interface, Availability, Operability, Fault Tolerance, Recoverability, Integrity, Reusability, and Adaptability
24	1,502	2	1	S34	FN , quality aspect
25	1,675	7	1	S42	Performance, Compatibility, Usability, Reliability, Security, Maintainability, Portability
26	1,846	6	1	S51	Reliability, Performance, Security, Availability, Usability, others
27	3,000	7	1	S21	Security,Reliability, Performance, Lifecycle, Usability, Capability, System Interface
28	3,140	7	1	S8	FR, Design Constraints, NFRs (security, efficiency, reliability, functionality, and usability/utility)
29	3,140	8	1	S18	Suitability, Accuracy, security, perability, understandability, attractiveness, time behavior, resource utilization
30	4,000	6	1	S19	NFRs (reliability, usability, portability, and performance), FR, others
31	6,000	6	1	S33	FRs, Usability, Reliability, Portability, Performance, others
32	6000	4	1	S44	Dependability, performance, supportability, usability
33	7,698	2	1	S43	NFR vs non-NFRs
34	10,963	6	1	S11	Confidentiality,Integrity,Availability, Identification & Authentication, Accountability, Privacy.

No	Size	No.classes	No studies	Studies IDs	Classes
35	11,876	15	2	S9, S14	Access control, Audit, Availability, Capacity and Performance, Legal, Look and feel, Maintainability, Operational, Privacy, Recoverability, reliability, Security, Usability, other, non applicable
36	14,249	2	1	S45	Security, Non-security
37	96,614	2	1	S46	Security, Non-security
38	54.2 m	6	1	S24	Maintainability, Functionality, Portability, Efficiency, Usability, and Reliability

Table A.3: Distribution of studies by dataset size and a number of classes

Study ID	**Security related**	**Performance related**	**Usability related**
S1	Security	Performance, Maintainability, Availability, Scalability, Legal	Usability, Operationality, Look and Feel
S2	N/A	N/A	N/A
S3	Integrity	Performance, Maintainability, Portability, Dependability, Deployability	Usability, Desgin
S4	Security	Performance, Maintainability, Availability, Scalability, Legal	Usability, Operationality, Look and Feel
S5	Security	N/A	Usability, Look and Feel
S6	Security	N/A	N/A
S7	N/A	Maintainability, Portability, Efficiency, Reliability	Usability, Functionality
S8	Security Efficiency, reliability	Functionality, usability/utility	
S9	Security, Access control, Audit, Privacy	Availability, Capacity and Performance, Legal, Maintainability, Recoverability, Reliability	Usability, look and feel, Operationality
S10	N/A	N/A	N/A
S11	Confidentiality, Integrity, Identification & Authentication, Accountability, Privacy	Availability	N/A
S12	N/A	N/A	N/A
S13	N/A	N/A	N/A
S14	Security	Performance, Reliability scalability	Usability
S15	Security, Safety, Privacy	Performance, Accuracy, Portability, Legal, Reliability, Availability, Interoperability	Accessibility
S16	Authentication-Authorization, Access control, Cryptography- Encryption, Data integrity	N/A	N/A
S17	Authentication-Authorization, Access control, Cryptography- encryption, Data integrity	N/A	N/A
S18	N/A	Efficiency (Time behaviour, Resource utilization)	Functionality (Suitability, Accuracy, Security), Usability (Operability, Understandability, Attractiveness)
S19	N/A	Reliability, Portability, Performance	Usability
S20	N/A	N/A	N/A
S21	Security	Reliability, Performance, Lifecycle, Capability	Usability, System Interface
S22	Security	Performance	Usability, Operationality

Study ID	Security related	Performance related	Usability related
S23	Security	Availability, Maintainability, Operability, Performance, Scalability, Fault Tolerance, Legal & Licensing	Usability, Look & Feel.
S24	N/A	Reliability, Efficiency, Maintainability, Portability	Functionality, Usability
S25	Security	Availability, Maintainability, Operability, Performance, Scalability, Fault Tolerance, Portability, Legal	Usability, Look & Feel
S26	Security	N/A	N/A
S27	Security	N/A	N/A
S28	Security	Availability, Maintainability, Operability, Performance, Scalability, Fault Tolerance, Legal	Usability, Look & Feel
S29	Security	Performance	Usability
S30	Integrity	Time Behavior, Resource Utilization, Capacity, Availability, Operability, Fault Tolerance, Recoverability, Reusability, Adaptabilit	, User Error Protection, Aesthetic User Interface
S31	Security	Operational, Performance	Usability
S32	N/A	N/A	N/A
S33	N/A	Reliability, Portability, Performance	Usability
S34	N/A	N/A	N/A
S35	N/A	Performance	Usability, Operationality
S36	Security	Reliability, Performance Efficiency, , Compatibility, Maintainability, Transferability	Operability
S37	Security	Availability, Maintainability, Performance, Scalability, Fault Tolerance, and Portability, Legal	Operability, Usability, Look & Feel
S38	Security	Maintainability, Performance	Usability, Operability
S39	N/A	N/A	N/A
S40	Security	Availability, Maintainability,Performance, Scalability, Fault Tolerance, Legal	Usability, Operability, Look & Feel
S41	N/A	N/A	N/A
S42	Security	Performance, Compatibility, Reliability, Maintainability, Portability	Usability
S43	N/A	N/A	N/A

Study ID	**Security related**	**Performance related**	**Usability related**
S44	N/A	Dependability, Performance, Supportability	Usability
S45	Security	N/A	N/A
S46	Security	N/A	N/A
S47	Security	Availability, Maintainability, Performance, Scalability, Fault Tolerance, Legal	Usability, Operability, Look & Feel
S48	Security	N/A	N/A
S49	Security	N/A	N/A
S50	Security	Availability, Maintainability, Performance, Scalability, Fault Tolerance, Portability, Legal	Operability,Usability, Look & Feel.
S51	Security	Reliability, Performance, Availability	Usability
Frequency	34	31	31

Table A.4: NFR categories and their identification in the selected studies

Score estimation methods	**No. studies**	**Studies ID**
K-fold cross	29	S2, S4, S5, S6, S7, S8, S9, S10, S11, S13, S16, S19, S20, S21, S22, S23, S25, S26, S27, S28, S34, S38,S40, S42, S45, S47, S48, S49, S50
Holdout	10	S33, S34, S36, S41, S42, S34 S44, S46, S48, S51
Pflod	3	S1, S34, S47
A new testing domain	3	S6, S34, S48
Train fitness	1	S34
Out-of-sample bootstrap	1	S31

Table A.5: The distribution of studies by the score estimation methods illustrated in Figure 3.11

Tool	**No. studies**	**Studies ID**
Weka	12	S2, S7, S8, S9, S11, S16, S19, S20, S21, S30, S33, S48
Scikit-learn	10	S22, S26, S27, S29, S31, S34, S39, S44, S50,S51
GATE	4	S8, S17, S18, S28
SVMlight framework	1	S3
Mallet	2	S13, S24
Gensim	3	S26, S37, S51
TensorFlow	3	S26, S38, S51
LIBSVM	1	S35

Table A.6: The distribution of the studies by ML tools used to draw Figure 3.10

Appendix B

A Supplement for Chapter 4

This Appendix is for Chapter 4. It includes tables used to draw the figures represented in that chapter 4. Also, it shows the results of our initial exterminate of building a word embedding model (word2vec) with 3009 requirements. These results are represented in Table B.3, illustrating that building effective word2vec for requirements requires a large dataset.

B.1 Detailed Information about the Studies

Tables B.1 and B.2 provide details of Figures 4.3 and 4.2 in Chapter 4, respectively.

Feature reduction techniques	**No**	**Studies IDs**
Lingustic	1	S48
Statisical	22	S1, S3, S5, S7, S9, S11, S16, S19, S20, S25, S26, S29, S31, S33, S35, S40, S41, S45, S46, S49, S50, S51
Both	4	S2, S22, S23, S34

Table B.1: Distribution of supervised ML-methods per feature reduction technique

B.2 Initial Experiment to Build wor2vec Model

Table B.3 shows the results for measuring similarity between two words using a word2vec model. The model was trained on 3009 requirements collected from different requirement specifications and datasets. We used *gensim* to build the model and *NLTK* for pre-processing requirements. We built two variants word2vec model skip-gram and

Study ID	**Minority size**	**Majority size**	**Imbalance level**
S1	10	62	6.2
S2	270	495	1.8
S4	10	255	25.5
S5	38	67	1.8
S6	187	323	1.7
S8	9	787	87.4
S9	43	3568	83
S11	204	3787	18.6
S13	10	67	6.7
S16	12	18	1.5
S18	24	214	8.9
S19	121	2183	18
S22	54	67	1.2
S23	255	370	1.5
S25	1	255	255
S26	255	370	1.5
S28	10	67	6.7
S29	54	67	1.2
S30	225	1141	5.1
S31	255	370	1.5
S34	545	938	1.7
S35	255	370	1.5
S38	113	345	3.1
S40	10	67	6.7
S42	25	165	6.6
S35	793	13456	17
S47	255	370	1.5
S48	66	559	8.5
S49	187	323	1.7
S50	12	444	37
S51	62	1119	18

Table B.2: The size of minority and majority classes and balance level (majority's size divided by minority's size) per study

Words pairs	**CBOW**	**Skip-gram**
"public", "logout"	0.804	0.949
"quick", "consist"	0.987	0.973
"hotmail", "email"	0.850	0.923
"server", "antivirus"	0.794	0.961
"make", " create"	"word 'create' not in vocabulary"	

Table B.3: The semantic similarity score of different pairs of words using two variant word2vec models trained on 3009 requirements

CBOW (illustrated in Section 6.1.3). Both models showed unreliable results, as the similarity scores are always high, even when the words are not semantically related (see table B.3).

Appendix C

A Supplement for Chapter 7

To identify common usability aspects (in Chapter 7), the snowball approach is applied to conduct the literature review. This appendix reports on a systematic review of 33 usability models used to define or measure usability in software systems. The review intends to find a common view of how usability is defined (i.e., how the abstract definition of usability can be divided into a set of aspects). These definitions are used to identify the common aspects of usability, which contribute to building ML classifiers. Our review is derived from two research questions (RQ):

RQ1: How do existing models classify usability concepts in general?

RQ2: What attributes of software usability are frequently addressed among different usability models and standards?

The following sections are organized as follows: Section C.1 discusses related work. Section C.2 presents the review process, and Section C.3 analyses the review results.

C.1 Related Work

Several reviews have been conducted to identify the common aspect of usability. However, these reviews are either conducted for a specific application domain (e.g., mobile application [Wei20]), defining the common aspects based on their name [SCM$^+$19, Wei18], or not clearly showing the process of model selection and commons identification [SS17]. By contrast, this review extracts the studied general definition of usability and groups related aspects based on their definition to define the common aspects.

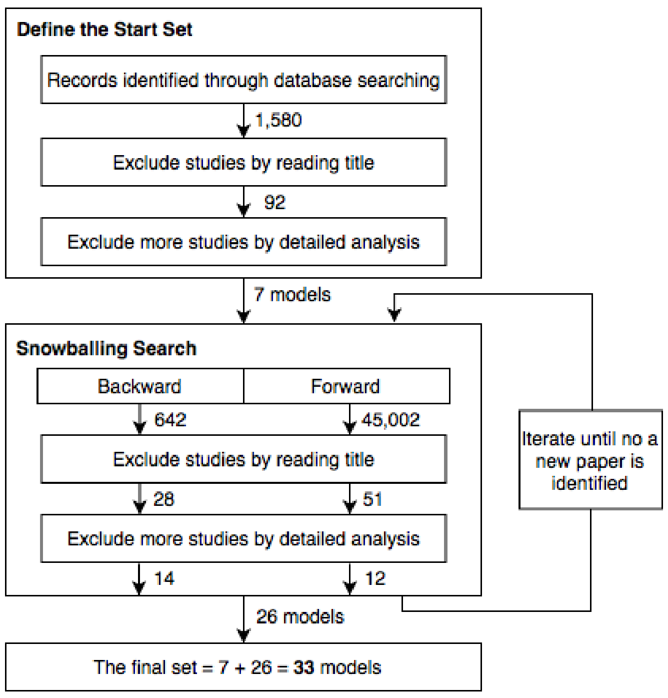

Define the Start Set
Records identified through database searching
1,580
Exclude studies by reading title
92
Exclude more studies by detailed analysis
7 models
Snowballing Search
Backward
Forward
642
45,002
Exclude studies by reading title
28
51
Exclude more studies by detailed analysis
14
12
26 models
The final set = 7 + 26 = 33 models
Iterate until no a new paper is identified

Figure C.1: The process of usability models selection based on Wohlin's snowballing method [Woh14]

C.2 Review Method and Process

We have adapted a snowball approach to answering our research questions (see Figure C.1). The snowball approach consists of three main steps: 1) identifying start set, 2) backward snowballing (looking at the reference lists), and 3) forward snowballing (looking at the citations). Below, each step is explained in detail.

C.2.1 Identification of the Start Set

The start set is identified using Google Scholar to avoid bias towards a specific publisher [Woh14]. Our search string was defined by breaking down the research questions according to the PICOC criteria (population, intervention, comparison, outcome, and context), as recommended by Kitchenham & Charters [KC07]. Table C.1 shows how the search terms were identified, and Table C.2 illustrates the details of using the search terms in Google Scholar.

Table C.2 shows that the Google Scholar retrieves 1,580 records. We manually identified the start set from the Google Scholar results by first reading the title. If the title was not clear enough to include the paper, the inclusion decision was made after reading the full text. The following inclusion and exclusion criteria were used for study

Criterion	Description	Main keywords	Alternatives
Population	Software Usability	Usability	-
Interventions	-		
Comparison	-		
Outcomes	Common Aspects	Aspects	Goals, Attribute, Component, and Criteria
Context	Usability definition	Usability model and standards	taxonomy

Table C.1: PICOC criteria to define the search string for the start set

Date of search	December 2020
Search Terms	Usability AND (model OR standard OR taxonomy) AND (Goal OR Aspect OR attribute OR component OR Criteria)
Search query	*allintitle: System OR Software OR model OR standard OR taxonomy OR Goal OR Aspect OR attribute OR component OR Criteria "Usability"*
Filter	Year: 1990 - 2020 Words occur: in the Title
No. of records retrieved	1,580

Table C.2: Details of search terms in Google Scholar to identify the start set

selection:

Inclusion:

1. It provides a new usability model that defines usability at a high-level.

2. It is published in the period from 1990 to 2020

Exclusion:

1. It is not in English

2. It is a secondary study (i.e., literature review)

3. It is not peer-reviewed (e.g., thesis, presentations), unless books.

4. It is an analytical or comparative study of software usability aspects.

5. It does not show a difference from existing models (e.g., explicitly mentioned that it uses a part of an existing one or applies an existing model)

6. It does not provide sufficient definition of the aspects.

7. It proposed for specific users (e.g., children), places (e.g., country), domains (mobile application or websites), software component (i.e., interface).

No	References	Year	Source Type	Database	Ref. Count	Citation Counts
1	Hasan and Al-Sarayreh [HAS15]	2015	Conference Paper	ACM	16	18
2	Nassa [Nas12]	2015	Journal Paper	ISOpress	9	
3	Gupta et al. [GAS14]	2014	Conference Paper	IEEE	27	33
4	Dubey et al. [DGR12]	2012	Journal Paper	ResearchGate	36	41
5	Alonso-Ríos et al. [ARVGMRMB09]	2009	Journal Paper	Taylor & Francis	19	146
6	Winter et al. [WWD07]	2007	Conference Paper	Springer	24	86
7	Seffah et al. [SDKP06]	2006	Journal Paper	Springer	49	782

Table C.3: The initial start set of the relevant papers for snowballing

8. It is for a specific attribute of usability (e.g., the effectiveness or learnability)

At the end of the selection process, we identified 7 relevant papers, which were then included in our start set (see Table C.3).

C.2.2 Backward Snowballing

We conducted a backward snowballing search on the references of each paper in the first set. The process is similar to that used to identify the start set; starting by reviewing the date and title, then moving to read the full text in case the title is not clear enough to make the inclusion decision. Relevant papers will be added to the starting set. This process is repeated until there is no new study in the starting set. In total, 642 papers were examined by backward snowballing and 14 were selected.

C.2.3 Forward Snowballing

We conducted a forward snowballing search on the citations of each paper in the start set, including those added by the backward snowballing search. The citations of each paper in the start set were identified on Google Scholar. Each citation paper was reviewed similarly to what we did in the backward snowballing search: checking title and date first, then the full text to make the inclusion decision. Relevant papers were added to the starting set. This process was repeated until no new papers were identified. In total, 45,002 papers were checked and 12 were selected.

Data item	Description
Bibliographic information	Authors, title, database, publication type,and publication year
Model Name	the term used to define the aspect
Aspects & their definition	Usability aspects and their definition

Table C.4: Data extraction form

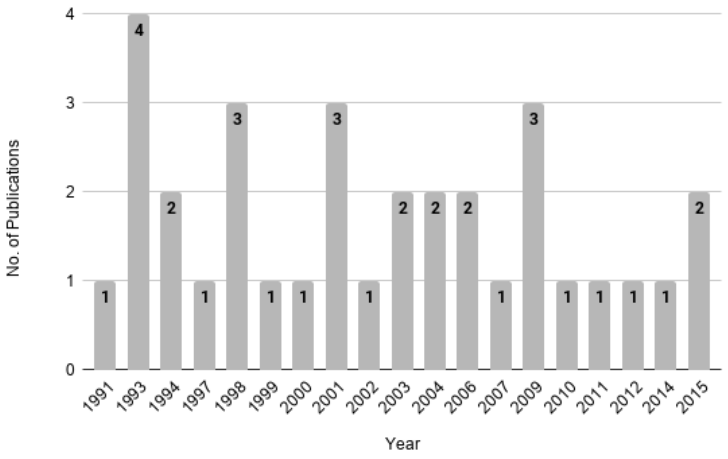


Figure C.2: Distribution of the selected publication in the period from 1990 to 2020. IF the year does not appear in Y-aix, that means there is no related publication identified in this year.

At the end of forward snowballing, 33 relevant papers were selected as the final set for our review.

C.2.4 Extracting and Synthesizing the Data

After selecting the related publications, we extracted data from each one using the data extraction form (illustrated in Table C.4). As the table shows, two types of data were extracted: the data required for answering the research questions and for displaying the bibliographic information of the study. The extracted data were stored in an Excel file.

Figure 2 shows the distribution of the selected studies published between 1991 and 2015. These publications were listed in 13 different databases. Figure C.3 show the frequency of studies published in different databases. Most of these publications are conference papers (10 publications), followed by standards (4 publications), journal papers (9 publications), books (9 publications), and a handbook (1 publication).

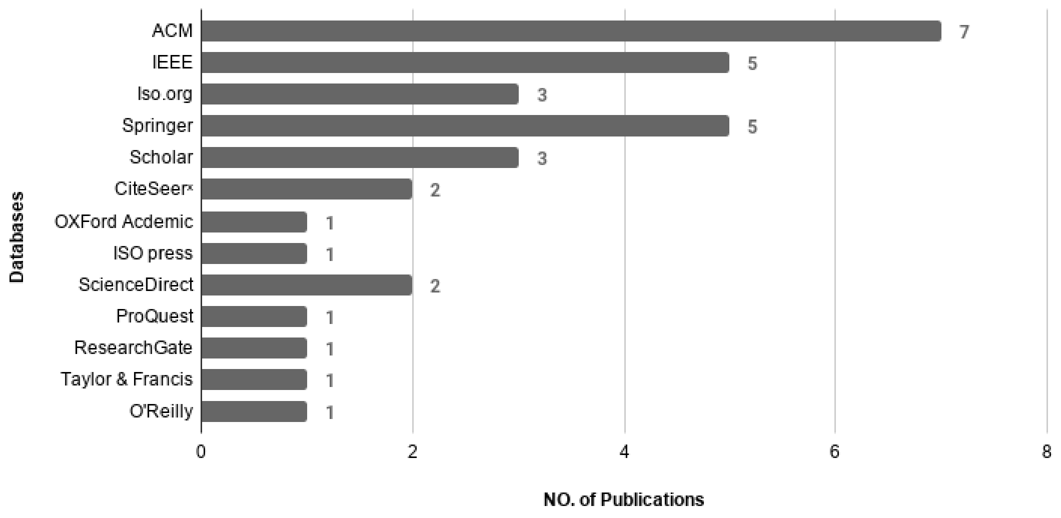


Figure C.3: The frequency number of selected publications per database

C.3 Results

This section provides the answers to the review research questions.

C.3.1 Existing Classification Models

RQ1: How do existing models classify the usability concept in general?

We identified 33 unique usability models. In total, these models consist of 222 aspects. Table C.5 lists the models that define usability and usability aspects.

No	Model	Year	Model Name	Usability Aspects
1	Hasan and Al-Sarayreh [HAS15]	2015	Usability attributes and measures	Efficiency, Effectiveness, Productivity, Satisfaction, Accessibility, Universality, Learnability, Operability, Appropriateness recognizability, User error protection, User interface aesthetics
2	Nassar [Nas12]	2015	Common criteria for usability	Consistency, User control, Ease of learning, Flexibility, Error management, Reduction of excess, Visibility of system status
3	Gupta and et al. [GAS14]	2014	Usability attributes	Efficiency, Effectiveness, Productivity, Satisfaction, Security, Universality.
4	Dubey et al.[DGR12]	2012	An integrated usability model	Efficiency, Effectiveness, Satisfaction, Safety, Comprehensibility.
5	ISO/IEC 25010 [fSEC$^+$11]	2011	Usability in quality in use model Usability in product quality model	Efficiency, Effectiveness, Satisfaction, Freedom from risk, Context coverage Accessibility, Operability, Learnability, Appropriateness recognizability, User error protection, User interface aesthetics.
6	Dubey et al. [DRS10]	2010	Environment Integrated Usability model	Acceptance, Accessibility, Attractiveness, Control, Ease of Use, Effectiveness, Efficiency, Emotion, Few Errors, Flexibility, Internationality, Learnability, Likeability, Memorability, Minimal Action, Minimal Memory load, Operability, Productivity, Safety, Satisfaction, Trustfulness , Understandability, Usability compliance, User Guidance
7	Alonso-Ríos et al. [ARVGMRMB09]	2009	Usability Taxonomy	Efficiency, Operability, Subjective satisfaction, Safety, Knowability, Robustness
8	Shackel [Sha09]	2009	Usability Definition	Effectiveness, Learnability, Flexibility, Attitude

No	Model	Year	Model Name	Usability Aspects
9	Sauro and Lewis [SL09]	2009	Prototypical usability metrics	Task times, Completion rates, Errors, Post task satisfaction, Post-test satisfaction.
10	Winter et al. [WWD07]	2007	2-dimensional quality model for usability	Goals (Efficiency, Effectiveness, Satisfaction, Safety). Attributes (Adaptability, Customizability, Controllability, Consistency, Simplicity, Relevance, Unambiguousness, Guardedness, Conformity, Existence)
11	Seffah et al. [SDKP06]	2006	Usability measurement model	Efficiency, effectiveness, productivity, satisfaction, learnability, safety, trustfulness, accessibility, universality, usefulness.
12	Adikari & McDonald [AM06]	2006	Conceptual Usability Attribute Model	Learnability , Memorability, Functional correctness, Efficiency, Error Tolerance, Flexibility, and Satisfaction
13	Shneiderman & Plaisant [SP04]	2004	The criterion of "user-friendly"	Time for users to learn specific functions, Speed of task performance, Rate of errors by users, User retention of commands over time, Subjective user satisfaction.
14	Folmer [FVGB04]	2004	Usability Attributes	Learnability, the Efficiency of use, Reliability, Satisfaction.
15	Abran et al. [AKSS03]	2003	An enhanced usability model	Efficiency, Effectiveness, Satisfaction, Security, Learnability
16	Dix et al. [DDF$^+$03]	2003	Usability design principles	Flexibility, Learnability, Robustness
17	Monk [Mon02]	2002	Components of usability	Ease of learning, ease of use, task fit, enjoyment, effective communication, and dependability.
18	Ferré et al. [FJWC01]	2001	Usability attributes	Learnability, Efficiency, User retention over time, Error rate, Satisfaction.

No	Model	Year	Model Name	Usability Aspects
19	Quesenbery [Que01]	2001	Usability Characteristics	Efficiency, Effectiveness, Engaging, Easy to learn, Error tolerant.
20	ISO/IEC 9126-1 [II04]	2001	Usability Characteristics	Understandability, Learnability, Operability, Attractiveness
21	Frojkaer et al.[FHH00]	2000	Usability aspects	Effectiveness, Efficiency, Satisfaction
22	Constantine et al. [CL99]	1999	Usability Rules and Principles	Rules: Access, Efficacy, Progression,Support, Context. Principles: structure, simplicity, visibility, feedback, tolerance, and reuse.
23	Lecerof et al. [LP98]	1998	Usability definition	Relevance, Efficiency, Attitude, Learnability and Safety.
24	ISO 9241-11 [Iso98]	1998	Usability definition	Effectiveness, Efficiency and Satisfaction
25	IEEE Std. 1061 [C$^+$98]	1998	Usability definition	Understandability, Ease of learning, Operability, Communicativeness
26	Wixon & Wilson [WW97]	1997	Usability attributes	Usefulness, Learnability (initial performance), Efficiency (long-term performance), Error rates, Memorability, First impressions, Advanced feature usage, Satisfaction or likability, Flexibility, Evolvability.
27	Preece et all. [PRS$^+$94]	1994	Usability definition	Throughput, Attitude, Flexibility, Learnability
28	Nielsen [Nie94]	1994	Usability definition	Efficiency, Satisfaction, Learnability, Memorability, Errors
29	Löwgren [Löw93]	1993	Usability definition	Relevance, Efficiency, Learnability, Attitude
30	Preece et al. [PBU93]	1993	Usability aspects	Safety, Effectiveness, Efficiency, and Enjoyableness.

No	Model	Year	Model Name	Usability Aspects
31	Hix et al [HH93]	1993	Common usability attributes	Initial performance, Long-term performance, Learnability, Retainability, Advanced feature usage, First impression, Long term user satisfaction.
32	Grady [Gra92]	1992	Measurable goals	User Documentation, Consistency, Aesthetics, Human Factors.
33	Bevan et al. [BKM91]	1991	Usability Measures	Acceptability, Ease of Use (user performance and satisfaction).

Table C.5: The usability models and definitions used to identify the common set

C.3.2 Common Aspects

RQ2: What attributes of software usability are frequently addressed among different usability models and standards?

By studying the identified models, we found that there are eight common usability aspects: Efficiency, Effectiveness, Satisfaction, Safety, Adaptability, Learnability, Users error tolerance, and Aesthetics. The common aspects were identified by analyzing the definitions of the aspects discussed in the previous section. Table C.6 shows the appearance of the common usability aspects of the 33 models studied.

NO	**Model**	**Common Aspects**							
		Efficiency	**Effectiveness**	**Satisfaction**	**Safety**	**Adaptability**	**Learnability**	**Users error protection**	**Aesthetics**
1	Hasan and Al-Sarayreh [HAS15]	Efficiency	Effectiveness, Productivity	Satisfaction	-	Accessibility, Universality	Appropriateness recognizability, Learnability, Operability	User error protection	User interface aesthetic
2	Nassar [Nas12]	-	-	-	-	Flexibility	Consistency, Reduction of excess, Ease of learning	User control, Error management	Visibility of system status
3	Gupta and et al. [GAS14]	Efficiency	Effectiveness, Productivity	Satisfaction	Security	Universality	Memorability	-	Aesthetics (under Satisfaction)
4	Dubey et al.[DGR12]	Efficiency	-	Satisfaction	Safety	Effectiveness	Comprehensibility	-	Attractiveness (under Satisfaction)
5	ISO/IEC 25010 [fSEC$^+$11]	Efficiency	Effectiveness	Satisfaction	Freedom from risk	Flexibility, Operability, Accessibility	Appropriateness recognizability, Learnability	User error protection	User interface aesthetics
6	Dubey et al. [DRS10]	Efficiency	Effectiveness, Productivity	Satisfaction, Likeability, Trustfulness	Safety	Accessibility, Control, Flexibility, Internationality, Operability	Minimal Action, Understandability, Learnability, Memorability load, Minimal Memory, User Guidance	Few Errors	Attractiveness, Emotion
7	Alonso-Ríos et al. [ARVGMRMB09]	Efficiency	-	Subjective satisfaction	Safety	Operability	Knowability	Robustness	Aesthetics (under satisfaction).
8	Shackel [Sha09]	-	Effectiveness	Attitude	-	Flexability	Learnability	-	-

NO	**Model**	**Common Aspects**							
		Efficiency	**Effectiveness**	**Satisfaction**	**Safety**	**Adaptability**	**Learnability**	**Users error protection**	**Aesthetics**
9	Sauro and Lewis [SL09]	Task time	Completion Rate, & Errors	post-task satisfaction & Post-Test Satisfaction	-	-	-	-	-
10	Winter et al. [WWD07]	Efficiency	Effectiveness	Satisfaction	Safety	Adaptability, Customizability, Guardedness, Controllability	Consistency, Conformity, Simplicity, Un-ambiguousness, Relevance	-	-
11	Seffah et al. [SDKP06]	Efficiency	Effectiveness, Productivity, Usefulness	Satisfaction & Trustfulness	Safety	Universality, Accessibility	Learnability	-	-
12	Adikari & McDonald [AML06]	Efficiency	Functional Correctness	Satisfaction	-	Flexibility	Learnability, Memorability	Error Tolerance	-
13	Shneiderman & Plaisant [SP04]	Speed of performance	Error rates	Subjective satisfaction	-	-	Retention over time	-	-
14	Folmer [FVGB04]	Efficiency of use	Reliability	Satisfaction	-	-	Learnability	-	-
15	Abran et al. [AKSS03]	Efficiency	Effectiveness	Satisfaction	Security	-	Learnability	-	-
16	Dix et al. [DDF$^+$03]	-	-	-	-	Flexibility	Learnability	Robustness	-
17	Monk [Mon02]	-	Effective communication	-	-	-	Ease-of-use, task fit, Ease-of-learning	-	Enjoyment
18	Ferré et al. [FJWC01]	Efficiency	Error rate	Satisfaction	-	-	Learnability, User retention over time	-	-
19	Quesenbery [Que01]	Efficiency	Effectiveness	-	-	-	Easy to Learn	Error tolerant	Engaging

NO	Model	Common Aspects							
		Efficiency	**Effectiveness**	**Satisfaction**	**Safety**	**Adaptability**	**Learnability**	**Users error protection**	**Aesthetics**
20	ISO/IEC 9126-1 [II04]	-	-	-	-	Operability	Understand-ability, Learnability	-	Attractiveness
21	Frojkaer et al.[FHH00]	Efficiency	Effectiveness	Satisfaction	-	-	-	-	-
22	Constantine et al. [CL99]	-	-	-	-	-	Structure, Simplicity, Reuse, Visibility	Tolerance	Feedback
23	Lecerof et al. [LP98]	Efficiency	Relevance	Attitude	Safety	Flexibility	Learnability	-	-
24	ISO 9241-11 [Iso98]	Efficiency	Effectiveness	Satisfaction	-	-	Learnability	-	-
25	IEEE Std. 1061 [C$^+$98]	-	-	-	-	Operability	Understandability, Ease of learning	-	Communicative-ness
26	Wixon & Wilson [WW97]	Efficiency	Error rates	Satisfaction or Likability	-	Flexibility, Evolvability	Memorability	-	First impressions
27	Preece et all. [PRS$^+$94]	Throughput	Throughput	Attitude	-	Flexibility	Memorability	-	-
28	Nielsen [Nie94]	Efficiency	-	Satisfaction	-	-	Learnability, Memorability	Errors	-
29	Löwgren [Löw93]	Efficiency	Relevance	Attitude	-	-	Learnability	-	-
30	Preece et al. [PBU93]	Efficiency	Effectiveness	Enjoyably	Safety	-	-	-	-
31	Hix et al [HH93]	-	-	First impression & Long-term user satisfaction	-	-	Learnability, Retainability	-	-
32	Grady [Gra92]	-	-	Aesthetics	-	-	Consistency, User Documentation	-	-

NO	Model	Common Aspects							
		Efficiency	**Effectiveness**	**Satisfaction**	**Safety**	**Adaptability**	**Learnability**	**Users error protection**	**Aesthetics**
33	Bevan et al. [BKM91]	-	Performance	Satisfaction	-	-	-	-	-
Appearance Frequency		23	23	26	10	17	30	10	13

Table C.6: The appearance of the common usability aspects in in the 33 existing models studied