12-17-2020 11:00 AM

Automatically Classifying Non-functional Requirements with Feature Extraction and Supervised Machine Learning Techniques

Mahtab EzzatiKarami, *The University of Western Ontario*

Supervisor: Prof. Nazim H. Madhavji, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Recommended Citation
EzzatiKarami, Mahtab, "Automatically Classifying Non-functional Requirements with Feature Extraction and Supervised Machine Learning Techniques" (2020). *Electronic Thesis and Dissertation Repository*. 7549.
https://ir.lib.uwo.ca/etd/7549

Abstract

Abstract. Context and Motivation: Non-functional requirements (NFRs) of a system need to be classified into different types such as usability, performance, etc. This would enable stakeholders to ensure the completeness of their work by extracting specific NFRs related to their expertise. **Question/Problem:** Because of the size and complexity of requirement specification documents, the manual classification of NFRs is time-consuming, labour-intensive, and error-prone. We thus need an automated solution that can provide a highly accurate and efficient categorization of NFRs. **Principal ideas/results:** In this investigation, using natural language processing and supervised machine learning (SML) techniques, we investigate with feature extraction techniques including Part Of Speech-tagging based, Bag of Words (BoW) ,and Term Frequency-Inverse Document Frequency (TF-IDF) combined with SML algorithms including Support Vector Machine (SVM), Stochastic Gradient Descent (SGD) SVM, Linear Regression (LR), Decision Tree (DT), Bagging DT, Extra Tree, Random Forest (RF), Gaussian Naïve Bayes (GNB), Multinomial Naïve Bayes (MNB), and Bernoulli Naïve Bayes (BNB). **Contribution:** The proposed strategy consists of three different combinations of the above-mentioned techniques. SVM with TF-IDF, LR with POS and BoW, and MNB with BoW all achieved recall values higher than 0.90, precision values above 0.87, and execution times less than 0.1s. In addition, we validated these classifiers using a case-study dataset where they promise results of recall values over 0.90 and precision values over 0.92.

Keywords

Non-functional Requirements, Classification, Supervised Machine Learning, Feature Extraction, Text Analysis

Summary for Lay Audience

Non-functional requirements (NFRs) describe a set of quality attributes required for software such as performance, reliability, availability, etc. Extracting NFRs from software requirement specification (SRS) documents and classifying them into different types can provide stakeholders with specific NFR types based on their concerns. Since the functional and non-functional requirements are mixed within the same SRS, it requires a lot of human effort for distinguishing them and can be an error-prone process. In this thesis research, we studied how accurately we can automatically identify non-functional requirements from SRS documents and classify them into different types, in particular, usability, performance, security, and operational requirements. Our proposed solution can support different stakeholders such as architects to whom architecturally significant requirements e.g. performance, efficiency, and interface. are important in choosing software architectural decisions; business analysts to whom business-related NFRs e.g. security, availability, and usability are important in the business; and developers with specific expertise e.g. user interface, security, and database.

Acknowledgments

Thanks to:

- Department of Computer Science for education, infrastructure and scholarship support.
- Professor Nazim H. Madhavji for his guidance and kind support throughout my M.Sc. program.
- My sister, Mahdiyeh for her encouragement, support, and being my backbone.
- Mom and Dad for forever loving me and supporting me even from miles away.

Table of Contents

List of Tables

List of Figures

Glossary of Abbreviations

RE	Requirement Engineering
NFR	Non-functional Requirement
FR	Functional Requirement
UI	User Interface
SML	Supervised Machine Learning
SRS	Software Requirement Specification
SVM	Support Vector Machine
LR	Logistic Regression
DT	Decision Tree
NB	Naïve Bayes
RF	Random Forest
NLP	Natural Language Processing
BoW	Bag of Words
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
SSRS	Subsystem Requirements Specification
POS	Part of Speech
SMO	Sequential Minimum Optimizer
GNB	Gaussian Naïve Bayes
MNB	Multinomial Naïve Bayes
BNB	Bernoulli Naïve Bayes

Glossary of Terms

True Positives	True Positives (TP) are the number of instances that are abnormal or anomalies (such as high response time of a system) and the model also predicts it as abnormal.
True Negatives	True Negatives (TN) are the number of instances that are normal points and the model also predicts it as normal.
False Positives	False Positives (FP) are the instances that are predicted as abnormal when they are normal data points.
False Negatives	False Negatives (FN) are the number of instances that are predicted as normal by the models but are abnormal data points.
Precision	Precision is the proportion of requirements that were correctly predicted as NFRs over the total number of data points that were predicted as either FRs or NFRs.
Recall	Recall is the proportion of requirements that were correctly predicted as NFRs over the total number of requirements.
Tokenization	the process of splitting the text into words, phrases, symbols, or other meaningful elements called tokens
POS Tagging	POS tagging is the process of marking up a word in a corpus to a corresponding part of a speech tag based on its context and definition.
Stemming	Stemming is the process of reducing inflected words to their word stem.
Lemmatization	Lemmatization is an alternative approach to stemming that considers the morphological analysis and the linguistic context of the term.

Chapter 1

1 Introduction

Identification and classification of non-functional requirements constitute essential steps providing categorized requirements to the stakeholders according to their concerns. In the following subsections we will cover the issues and concerns that have recently emerged in this area, and then we will demonstrate how through using natural language processing and machine learning techniques we can tackle these problems and provide a highly accurate automatic solution.

1.1 Requirement Engineering

Requirement engineering (RE) is a set of activities through which the requirements are gathered by identifying different stakeholders involved in the project and their needs. These requirements should be documented for future analysis, communication, and subsequent implementation [1]. RE is one of the prominent activities in the software development process because it has the most dominant impact on the capabilities of the product [2].

RE should be able to specify the formal requirement specification correctly, as these specifications will act as a contract between stakeholders and the software development team. If errors like producing incomplete and incorrect requirements go undetected in the requirement engineering phase until a later stage of software development, the whole development process and maintenance would be very costly. It should be mentioned that there could be different formats of data in requirement documents such as natural language texts, diagrams, tables, etc.

1.2 Requirement Types

Requirements are classified as functional and non-functional which can be further drilled into more specific categories. Functional requirements describe what the system should do [3] and non-functional requirements specify how well the system performs its intended functions. Non-functional requirements are also called quality requirements [4].

The success of a software system depends on how well both FRs and NFRs are covered during the requirement engineering activities [5]. One well known survey [6] found that in software projects where NFRs were not considered, a failure rate of 60% or higher was observed. As an example, a U.S. Army intelligence-sharing application which cost USD$ 2.7 billion to develop, was found to be unusable when deployed within a realistic operating environment because of capacity, performance, and usability issues [5]. As another example, in the London Ambulance tragedy, the computer-aided dispatch system for ambulances lost track of its locating system, which led to the death of 46 people within only a few hours of its initial deployment [7]. It was reported that one of the main reasons for this failure was an incomplete set of requirements found in the requirement specification phase due to leaving out key stakeholders in the elicitation process. The delivered system had issues with reliability, performance, and scalability which made it fail in cases with invalid data and a large amount of incident information [8].

1.3 Importance of Non-functional requirements to different stakeholders

Ideally, NFRs of a system are consciously considered throughout all stages of development, from initial design through implementation and quality assurance to deployment and ongoing evolution [9]. In practice, they are often explicitly addressed late in development, during system-level testing, and in an ad hoc manner [10]. Recent studies have focused on the importance of extracting NFRs from software documents (e.g. software requirement specification documents) in the early stages of software development [11].

Furthermore, requirements may serve different purposes, from highlighting security vulnerabilities to measuring scalability necessities to assigning general look-and-feel [12]. Identification of all requirements of a specific type (i.e. security-related) allows each of stakeholders involved in the software project to focus on particular non-functional aspects for the system and to assess project completeness. For instance, software specialists can immediately locate which requirements interest them without the need to peruse through the entire SRS (e.g., the UX designer is likely interested in look-and-feel requirements).

Another group of stakeholders who could benefit from the identification and classification of NFRs are architects. In software engineering, non-functional requirements and software architectures (SA) are closely related as Rick Kazman et.al have also demonstrated in their research [13]. For example, architects can justify choosing a layered architectural style in terms of maintainability and portability, or choosing a particular database technology in terms of efficiency [14]. Software architects regularly deal with NFRs as part of their SA design responsibility. They must identify a system's NFRs and determine how architectural decisions affect the NFRs' fulfillment.

Based on the study [14], David Ameller et. al concluded that NFRs generally drive four types of architectural decisions:

- Architectural patterns: For example, when working on a very large project that spans 8-12+ month, non-functional requirements will change like other requirements. In order to support later changes, an architect of the system might decide to choose a layered architecture. As another example, if a system is ultra-reliable, it should have more detailed and accurate "dependability" objectives, and choosing a more dependability-supportive architectural pattern like the one proposed in [15] will be required.
- Implementation strategies: Different types of requirements might need strategies at a detailed architectural level. For example, architectures might duplicate the tables of a database to reduce the access time based on the specific performance requirement.
- Transversal decisions: Some NFRs could influence the whole architecture of the system. For example, if maintainability is a required quality of a system, it can affect the type of components used in the system such as using third-party components and especially open source software (OSS).
- Technological platforms: NFRs might be achieved by technological choices in the database, the middleware, and so on. For example, high availability requirement of a system might only be assured by Oracle. As another example, queries of the system database might be implemented directly in JDBC (Joint Database Connectivity) instead of Hibernate due to efficiency reasons [14].

Considering the business side of the projects, business analysts need to ensure the completeness of their work by satisfying business-related NFRs like security, availability, and usability. Failure in the identification of the business-critical non-functional requirements of a particular task like response time, security, availability, performance, reliability, portability, maintainability, quality of user interaction with system, etc. could lead to key information being overlooked or deferred [16].

It is important to emphasize that the entire set of requirements is hardly known to business analysts at the start of the project [17]. As the project progresses, a better understanding of the project is achieved resulting in more clarity on the projects requirements set. Often, this leads to business side pushing for additional requirements (primarily NFRs) that stakeholders want to see in the final product. With additional requirements, if the stakes associated with the project are high, the project organization might want to satisfy these demands. The end results are escalating costs, high schedule slippage, overworked team members, and team attrition [18]; thereby contributing to project failures.

The situation is likely to improve if business analysts could be provided with a tool to identify and classify the specific NFRs they look for in the project. Using such a tool at the beginning of the project will ensure them that they have the entire set of the related NFRs. This will help them focus on the evaluation of these NFRs in terms of business objectives, so that they can discuss their perspective with the business organization. This can assist the project organization to better negotiate the project NFRs by leaving out those whose relative contribution to business objectives is not significant [15].

In human resource allocation and optimization [19], different developers possess different expertise in handling various aspects of software development. Different tasks in development may require different expertise from the developers. Thus, a match of developers and tasks is at the core of the success of software development. When it comes to different requirements of a system, NFRs tend to be properties of a system as a whole; hence not all NFRs can be verified for individual components of the system [20]. This makes the classification of NFRs into different types necessary in development because such a classification can be used in the formation of different types of development tasks.

Consequently, those tasks are assigned to developers aggregately according to their expertise. For instance, the NFRs with “performance” type and the NFRs with “maintainability” type should be dealt by developers with different expertise. Project managers forward the NFRs of the type “look and feel” to UI (User Interface) design experts of the system so that the satisfaction of these NFRs can be ensured throughout the whole project.

To show the importance of identification and classification of NFRs for developers of a system, we could take a security-critical software system as an example. In such a system, formal method with model checking of the correctness of specification on system security is used to determine whether or not the ongoing behavior holds the system’s structure. However, a serious drawback of model checking is the state explosion problem because the size of the global state graph is (at least) exponential in the size of the program text. Thus, an exigent demand from system developers is to identify the security-related NFRs from the requirement documents and to express them using formal specifications, and then to conduct verification of the satisfaction of system’s properties based on those security specifications.

Moreover, functional requirements can be measured as either satisfied or not satisfied, but non-functional requirements cannot simply be measured by a linear scale as degree of satisfaction [21]. In system testing, for example, project developers and test engineers can customize their test strategy at the beginning of development process based on the classification of NFRs and the specific metrics required for each of NFR categories. Therefore, system behaviors of NFRs could be reported directly to the project manager.

1.4 Problem Analysis

It seems that identification and classification of non-functional requirements in a software project could be an easy task, but the manual task of labeling what category a requirement falls under could be challenging, tedious and time-consuming, considering that some of the stakeholders involved in the project (e.g. users) are non-technical people. Requirements gathered from these non-technical stakeholders could lack technical details and any mention of specific requirement types. This will make the manual requirement labeling require domain-expertise, which can be limited and expensive. Moreover, because requirement documents are prepared during meetings with stakeholders and requirement gatherings, most NFRs might somehow be hidden across the requirements that mainly specify functionality. This can make identification and classification task error-prone and labor-intensive [12].

The software documentations are often neglected despite persistent pleas from educators and practitioners [22]. Open-source projects usually include archived mailing lists, message boards, administrative manuals, installation manuals, and user manuals. Government and industry-based standards also contain critical requirement-related information. All of these materials can be potential resources for the identification of NFRs. Therefore, the high number of available sources in a large-scale project makes the NFR classification task tedious, complex, and time-consuming.

These observations highlight the need for an automated solution for the identification and classification of non-functional requirements in a software project. The goal of our research is to aid stakeholders to extract relevant non-functional requirements more effectively in available unconstrained requirement documents through automated natural language processing, with the help of supervised machine learning (ML) techniques. ML algorithms have been shown to have considerable practical importance in many application domains. This is especially true for domains where large databases are available [23]. Because requirements specification documents are mainly given in natural language, ML can be useful by emulating human processing. By automatically identifying both functional and non-functional requirements and additionally classifying non-functional requirements into sub types, various stakeholders involved in the project, whether technical people such as developers or architects, or non-technical stakeholders such as customers (e.g. stakeholders budgeting the project), can use this system as a highly accurate and fast solution. They can use such a tool any time in the project development process. It removes most of the laborious effort which can consequently help with the project budget as well.

Research Questions

The goal of this research is to classify NFRs according to the stakeholders' concerns. In this research work, we narrowed our focus to four non-functional requirements including usability, security, performance, and operation quality attributes, since they are the most common types of NFRs that have been mentioned in our dataset. Therefore, given a dataset of all requirements, we aim to first identify which requirements are functional and which ones are non-functional and then we will further classify the non-functional requirements into four mentioned types. By identifying and classifying the requirements written in natural language successfully, we will be able to take the first step to provide stakeholders with their concern-based non-functional requirements. It is important to note that in this work, the quality attributes refer to the product quality aspects. In the future, we hope to extend this work to encompass process quality aspects such as risk, cost, and effort.

This thesis asks the following two research questions:

1. RQ1: How effective are supervised machine learning techniques in automatically identifying FRs from NFRs and classifying 4 types of NFRs including usability, performance, security, and operation NFRs?
2. RQ2: Which combination of feature extraction techniques and supervised machine learning algorithms gives the best results for RQ1?
    a. What are the most informative features for each of usability, security, performance, and operation NFR types?

By answering these research questions and proposing an efficient and accurate solution, we provide the required means and strategies to further develop a concern-based NFR classification tool which can be used by stakeholders.

Solution Approach

Going through the related work in Chapter 3, we will see that there have been several attempts in the classification of non-functional requirements using machine learning approaches. However, there are a few main challenges visible in those works. In most cases, these challenges have not been mentioned, or even if these problems have been recognized by the researchers, no specific solution has been offered. As it is shown in [57], between different techniques of machine learning approaches, supervised, semi-supervised, and unsupervised, supervised learning is the most popular type of ML in the area of requirement classification. Therefore, we felt a need to deal with some of the challenges that already existed in this field by doing a comprehensive comparative study of non-functional requirement classification using supervised machine learning techniques. Although the published research works have attained promising results, our research improves upon these results significantly as will be shown in the later chapters. Analysis of the available studies in identification and classification of non-functional requirements reveals a general process pattern for applying machine learning-based approach to identify and classify NFRs in a textual document [57]. This process is divided into three major phases which is shown in Figure 1.

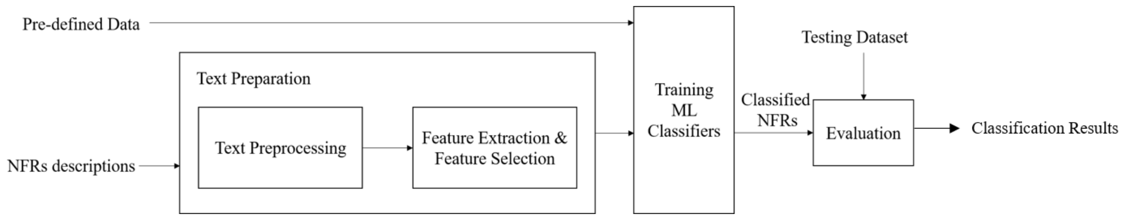


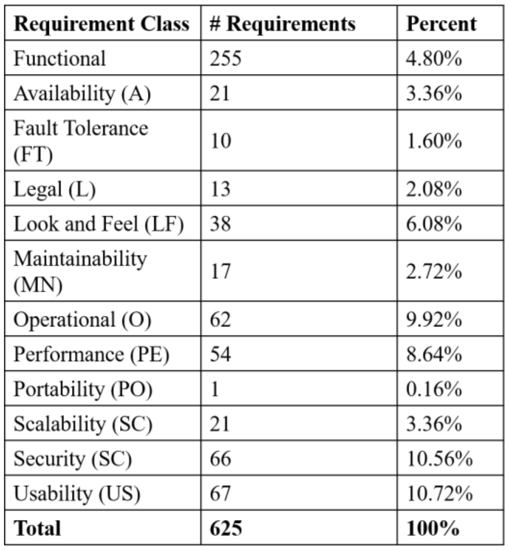
Figure 1: A general process of applying ML algorithms to classify NFRs in textual requirements documents.

As illustrated in Figure 1, pre-defined data is a precondition for building NFRs classifier using ML algorithms. Supervised Learning algorithms requires a labeled dataset. Additionally, there is a text preparation phase, which involves applying natural language text processing techniques and ML algorithms; and the evaluation phase, which is concerned with assessing an ML algorithm’s approach to classify NFRs. We will get into more details of the process in methodology section.

Predefined Dataset

Since 2006, when Cleland-Huang et al. made their dataset publicly available as NFR PROMISE dataset, this dataset has been used by most of the researchers in the field for the task of non-functional requirement classification [18]. This dataset consists of 15 SRS documents developed as term projects by MS students at DePaul University. Although NFR PROMISE dataset is still a valuable dataset today, and it encouraged researchers to work on the task of NFR classification, the requirements were originated from university student projects, not from requirement documents of a production system [58]. Table 1 presents an overview of the NFR PROMISE dataset that includes 12 classes and 625 requirements. The table shows that some classes of the NFR dataset are underrepresented. The NFRs Portability (PO) and Fault Tolerance (FT) are very rare in the dataset.

Table 1: Overview of PROMISE dataset with the number of requirements available in each class



Requirement Class	**# Requirements**	**Percent**
Functional	255	4.80%
Availability (A)	21	3.36%
Fault Tolerance (FT)	10	1.60%
Legal (L)	13	2.08%
Look and Feel (LF)	38	6.08%
Maintainability (MN)	17	2.72%
Operational (O)	62	9.92%
Performance (PE)	54	8.64%
Portability (PO)	1	0.16%
Scalability (SC)	21	3.36%
Security (SC)	66	10.56%
Usability (US)	67	10.72%
Total	**625**	**100%**

Potential issues with such datasets are the result of the absolute rarity of some concepts as well as the within-class imbalances [60]. The issue with rare instances of a target class can make the classification difficult. When the concept itself has a sub concept with limited instances, additional difficulty might arise when classifying a minority concept due to within-class imbalances [60]. This is the first main issue using PROMISE dataset for the classification task.

Moreover, the total number of requirements in PROMISE dataset is 625 out of which 255 are functional requirements and 370 are non-functional requirements. There is a common belief in machine learning that there is not enough data and more data gives more benefits [59]; however, this cannot accurately be validated till the results of a research work can be tested utilizing larger datasets. That said, as illustrated in [34], training a classification model for classifying NFRs using PROMISE dataset, the curves of the training scores and cross validation scores got closer to each other with a larger training size and the model’s performance stabilizes with a training size bigger than 400 requirements. Small number of requirements in the NFR RPOMISE dataset and not utilizing any techniques to expand the training dataset are the second concern related to using this dataset.

Evaluation Phase

In this phase, the evaluation of the NFR classifier is conducted using various techniques to determine the developing classifier’s effectiveness. In the classification of NFRs using supervised machine learning algorithms, the common technique is to use K-fold cross validation for which the input dataset is randomly divided into K folds. Each fold has the same data size; one is used for testing, while the others are used for training. The learning algorithm is run K times, and the average K results are calculated to produce a single result to measure the NFR classifier’s performance. Considering performance evaluation metrics used in this field, there are a few noticeable issues. As it is shown in [18], precision and recall are the two most common evaluation metrics used in the classification of NFRs.

Precision measures the total number of correctly classified NFRs in respect to the number of NFRs retrieved. It is defined as P = true positive / (true positive + false positive). Recall measures the percentage of NFRs that were correctly classified and is defined as R = true positive / (true positive + false negatives) [12]. True positive represent the number of correctly classified requirements, false positive represent the number of incorrectly classified requirements, true negatives represent the number of requirements correctly not classified, and false negatives are the number of requirements incorrectly not classified [3].

Precision and recall are often used together [48] and there is a trade-off between them. Precision ensures that all retrieved requirements are truly relevant, whereas recall focuses on retrieving all relevant requirements. It is arguable which value is more important to measure the classification results. [12] focused on achieving a high recall (76.7%) but lower precision (12.4%) as it was believed that rejecting irrelevant requirements manually from a set of retrieved requirements was easier than reading an entire document looking for missing requirements. On the other hand, [39] argued that precision was more important if recall was acceptable for automatic classification to avoid large amounts of irrelevant NFRs from being misclassified as relevant (false positive), which might frustrate users. There are some studies that achieved higher recall and lower precision [12, 25], while there are other studies that achieved higher precision and lower recall such as [5]. There is a study which achieved high recall and high precision [47]; however, this study only used a machine learning approach to detect NFRs and could not classify them.

Furthermore, in industry, we will encounter large-scale projects with thousands of requirement data. Hence, in order to provide stakeholders with an efficient tool for concern-based NFR classification, execution time of the NFR identification and classification is of paramount importance. Among all the studies that have been done so far, only two recent works have included execution time as a complementary performance metric [61]. A comparison of execution times between our work and the related work is provided in Chapter 5.

That said, there is a potential to use classification techniques with the propose of improving recall and precision metrics and achieving high values in both of them. Also, including execution time as a complementary evaluation metric will enable us to propose the best strategy for the classification of non-functional requirements.

Industrial Dataset

Supervised Learning requires a set of NFRs that were correctly classified to their categories. Considering that the goal of classifying NFRs into their specific types is to propose solutions to be used by practitioners in the industry, one of the best sources to use in either training the classifiers or validating the proposed tools is industrial dataset. As indicated in [18], out of 24 studies the researchers reviewed in the field of requirement classification, only five studies had used industrial datasets. Out of those five, only one was focused on the classification of non-functional requirements using supervised machine learning techniques. Other four either did a different classification task on NFRs like tracing changes or they used techniques other than supervised machine learning techniques (e.g. using information retrieval techniques). This shows the potential to do a case study using industrial dataset to validate the achieved results.

Ensemble Techniques

Reviewing the related works in this field, there have been a few works utilizing ensemble techniques which can produce more accurate results by creating multiple models and combining them together. In the recent study [17], the researchers included ensemble techniques to classify a dataset of 58 security requirements and their results indicated that ensemble classifiers can perform well. To the best of our knowledge, only one recent work [7] used Extra Trees as an ensemble technique for the identification of non-functional requirements. This indicates a potential to investigate ensemble techniques such as bagging trees, boosted trees, random forest, etc.

Key Contributions

With all that said, there is a need for a study that can propose a solution using supervised machine learning algorithms while handling the mentioned problems such as using more balanced dataset with greater number of requirement instances, working toward achieving both high recall and precision values, providing execution times of the techniques, etc.; which can be trusted to some level if not completely, considering the fact that there is more space for more validation techniques.

- In this work, we use the combination of two datasets. First, the NFR PROMISE dataset for the purpose of comparison with other works and second the PURE dataset which is composed of publicly available industrial requirement specifications documents. By using the combination of these two datasets, we achieved a fairly balanced dataset with more than 500 requirements.
- Our proposed strategy enables the classifiers to achieve high precision and high recall with more focus on recall values. We will discuss the reasoning although there is always more space to argument on the prioritization of these two metrics.
- We cover a fairly large set of combinations of feature extraction techniques including syntactic POS-tagging-based adopted from [2], TF-IDF, and BOW, with supervised machine learning techniques including support vector machine, Stochastic Gradient Descent SVM, Decision Tree, Linear Regression, Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes and ensemble techniques including Extra

Trees, Bagging Decision Tree, Boosted Decision Tree, Rand Random Forest. This can enable comparisons with related works and potentially be a good comparative study in the field.

- Our study also includes a case study by using an industrial dataset to validate our results which is required in this field considering that lack of industrial data as one of the main challenges in this field [18].

1.5 Organization

The rest of this thesis is organized as follows. Chapter 2 reviews the background for this project which briefly provides the basis for software requirement engineering, natural language processing, and supervised machine learning techniques for classification. Chapter 3 summarizes the related works in finding NFRs from software documents. Chapter 4 describes our methodology to identify and classify non-functional requirements and details of our implementation. Chapter 5 presents our evaluation process along with the results and discussion of our study. Finally, chapter 6 summarizes our work, discusses limitation and validity issues, and suggests possible future research direction.

Chapter 2

2 Background

2.1 Requirements

Software requirements describe the behaviours, properties or attributes of a target system, and they are determined in the early stages of a software development process to identify what should be built in the system [3]. In describing the intended system requirements, a software requirement document is an official document that is created before the project has started development to coordinate every stakeholder regarding what a system should do [3, 4].

Requirements may be described either in an unofficial high-level description as abstract requirements or they can be more specific with details. The abstract requirements are composed of the stakeholders' perspective usually using natural language, informal diagrams or some notations specific to the problem such as mathematical equations for a control system.

The detailed requirements are system requirements expressed in an abstract model of the system such as mathematical models, UML or similar diagrams, or object class hierarchies, etc. These models are usually annotated with natural language texts [3]. Independent of the extent of requirement description details, requirement standards and textbooks usually classify requirements into functional requirements and non-functional requirements (NFRs) [22].

Functional requirements specify what the system does [3] and non-functional requirements, which are also called quality requirements, determine how the system performs its intended functions [4]. As an example, a functional requirement might declare that a system should authenticate a system user identity; a non-functional requirement might declare that the system should be capable of finishing the authentication process in less than four seconds [3].

As NFRs address important quality aspects of the system, such as security, performance, reliability, etc., they are essential for the successful functioning of the software. If NFRs are not met, they can result in inconsistent, poor quality software with which customers and developers are unsatisfied. This in turn will lead to extra costs and time to fix the system because it was not developed with regards to the required NFRs [24].

NFRs can include a large number of categories as Chung et al. have identified 156 NFR categories, but it should be noted that there is not a formal complete list of non-functional requirements that can serve all the needs of various application domains in different circumstances. As a result, for each project, the considered NFR groups should be customized based on the project’s needs and the system domain [24].

Some previous studies have worked on classifying NFRs into a taxonomy [24, 25, 26]. Figure 1 shows a taxonomy proposed by Afreen et.al in [25]. This taxonomy classifies NFRs on the basis of commonly used NFRs, definitions and attributes of NFRs, and conflicting NFRs.

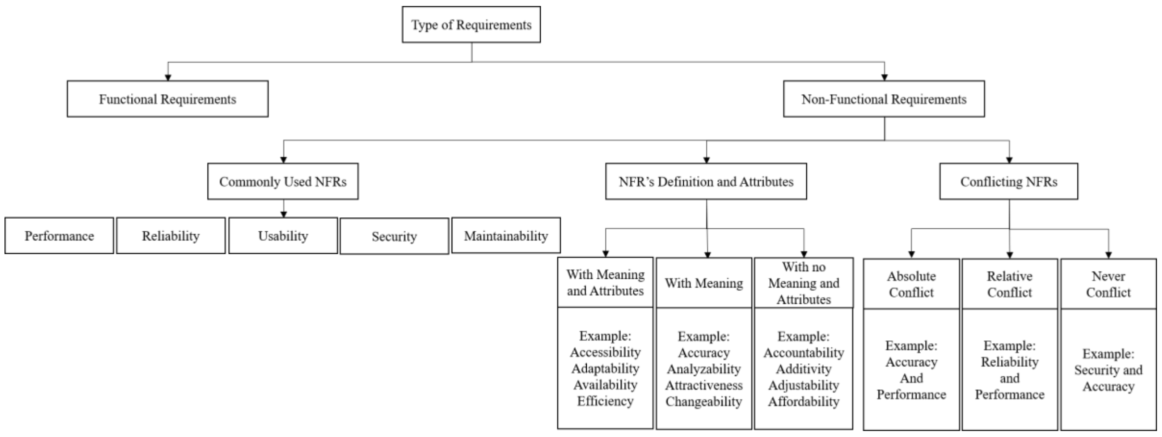

Type of Requirements
Functional Requirements
Non-Functional Requirements
Commonly Used NFRs
NFR’s Definition and Attributes
Conflicting NFRs
Performance
Reliability
Usability
Security
Maintainability
With Meaning and Attributes
With Meaning
With no Meaning and Attributes
Absolute Conflict
Relative Conflict
Never Conflict
Example: Accessibility Adaptability Availability Efficiency
Example: Accuracy Analyzability Attractiveness Changeability
Example: Accountability Additivity Adjustability Affordability
Example: Accuracy And Performance
Example: Reliability and Performance
Example: Security and Accuracy

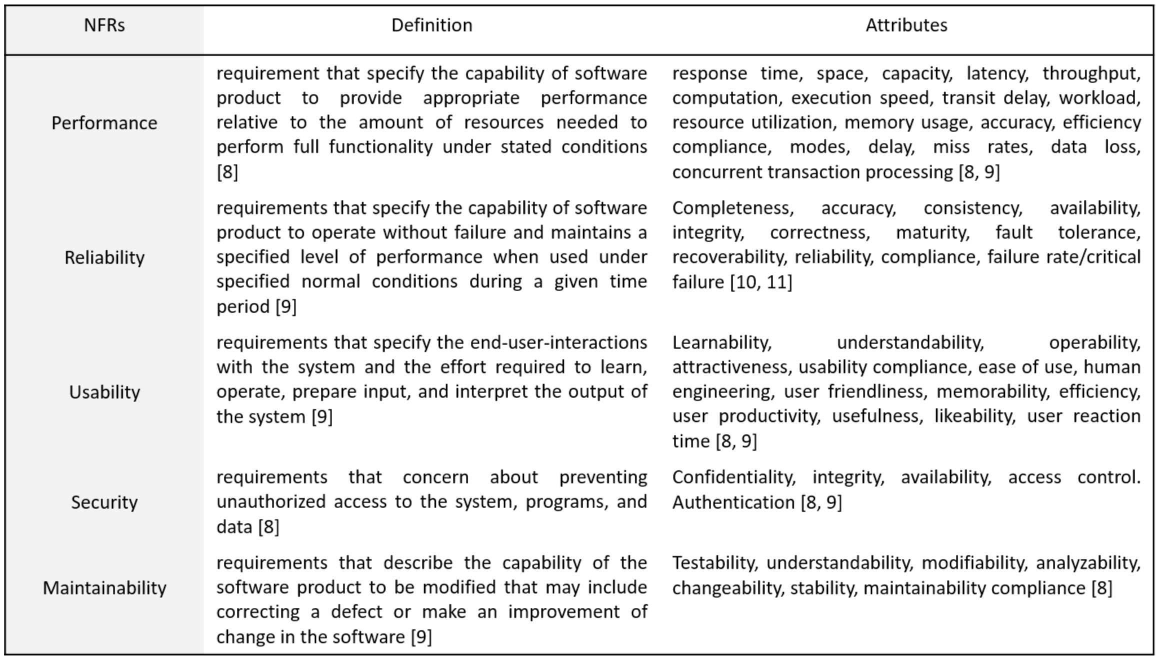
Figure 2: Classification of non-functional requirements [6]

Commonly used NFRs are the non-functional requirements that are useful for the success of different application domains such as information systems, web-based systems, etc., and they include performance, reliability, usability, security, and maintainability [25]. Table 1 shows the definition and attributes of the most commonly used NFRs.

The objective of classifying NFRs based on their definition and attributes is to identify the requirements based on whether they have been clearly defined in the literature, like performance, security, usability, etc., or whether they have been supported by their attributes, like accessibility, adaptability, efficiency, etc. Afreen et al. classified NFRs on the basis of the following: (i) with meaning and attributes, (ii) with meaning, (iii) with no meaning and attributes [25].

Conflicting NFRs refer to those non-functional requirements that may have some kind of conflict with other NFRs [27, 28]. They are further categorized into absolute conflicting NFRs, such as performance; relative conflicting NFRs, such as reliability; and never conflicting NFRs, such as security.

Table 2: The Most Commonly Considered NFRs in NFR taxonomy proposed by Afreen et.al in [25]



NFRs	Definition	Attributes
Performance	requirement that specify the capability of software product to provide appropriate performance relative to the amount of resources needed to perform full functionality under stated conditions [8]	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 [8, 9]
Reliability	requirements that specify the capability of software product to operate without failure and maintains a specified level of performance when used under specified normal conditions during a given time period [9]	Completeness, accuracy, consistency, availability, integrity, correctness, maturity, fault tolerance, recoverability, reliability, compliance, failure rate/critical failure [10, 11]
Usability	requirements that specify the end-user-interactions with the system and the effort required to learn, operate, prepare input, and interpret the output of the system [9]	Learnability, understandability, operability, attractiveness, usability compliance, ease of use, human engineering, user friendliness, memorability, efficiency, user productivity, usefulness, likeability, user reaction time [8, 9]
Security	requirements that concern about preventing unauthorized access to the system, programs, and data [8]	Confidentiality, integrity, availability, access control. Authentication [8, 9]
Maintainability	requirements that describe the capability of the software product to be modified that may include correcting a defect or make an improvement of change in the software [9]	Testability, understandability, modifiability, analyzability, changeability, stability, maintainability compliance [8]

2.1.1 Software Requirement Specification

A software requirement specification (SRS) is one of the main artifacts produced in the early stages of software development as part of a contract between the client and the service provider to ensure that the software meets their needs [4]. SRS is used to describe what software will do and how it will be expected to perform to fulfill all stakeholder needs. A requirement specification document is designed to assist the communication between the technical stakeholders, such as analysts, developers, and testers on one side, and non-technical stakeholders, such as the clients and the product managers, on the other side.

SRS documents are usually written in natural language and they contain all system requirements, including the functional requirements and the non-functional requirements [20]. They can also include definitions of special concepts, examples, diagrams, and cross references [30]. In this project, our focus will be on SRS documents written in English and we will use the requirements defined using natural language. Diagrams, tables, notations, etc. are not in the scope of our work.

2.2 Stakeholders

System stakeholders are groups or individuals who are interested in the actions of a corporation and will be affected by those actions. Stakeholders have the ability to directly or indirectly influence the organization's requirements [31].

Stakeholders can be system end-users, managers, developers, engineers responsible for the maintenance of the system, people who are involved in the organizational processes affected by the system, customers of the organization who will use the system to provide other services, or regulator certification authorities, etc. [3].

It is essential for the success of the project to meet all stakeholder needs. Stakeholders may begin with desires and expectations that may include unclear, ambiguous statements difficult to use for software engineering activities. These statements must be coalesced into a set of clear and concise requirements for the system [32].

It should be noted that stakeholders typically do not request non-functional requirements but they implicitly expect these features to be part of any system they use. For example, a user could easily say, "I want the system to store different formulae for calculations". Rarely will you hear them say, "the system should be able to perform calculations on 1000 records in 20s", even though they might expect it to work this way. In order to make sure their expectations will be met, it is important to enable them to check the requirements they care about the most and make sure their expectations are covered [33].

2.3 Classification of Requirements

Requirement classification refers to the identification of requirements as belonging to a specific category to highlight their role in the project. Two examples of classification tasks are (1) distinguishing between functional and non-functional requirements, and (2) determining whether a non-functional requirement is related to concerns such as security, performance, usability, etc. [12].

NFRs are predominant because they determine areas essential for the wellbeing of the system as a whole rather than just a single feature in a module. Attaining them requires not only careful design decisions in the early stages of the software development life cycle, but also continuous efforts throughout the entire process.

Unfortunately, as Kurtanovi´c et al. maintain [34], most NFRs are identified later in the software development process [34, 35] and they are often described vaguely [36], which might cause engineers to neglect their importance in the project [11]. Classifying requirements can encourage transparency and organization within the SRS and can stimulate awareness toward crucial system concerns that should be as much of a priority to engineers as feature development.

However, manually classifying requirements calls for engineers who have expertise in the respective areas (i.e., proper identification of security requirements calls for security knowledge). This resource barrier can discourage project managers and engineers from properly assessing NFRs throughout the development process, delaying neglected or unidentified issues and thus amplifying the risks for defects, performance inadequacies, and technical debt. Automating this task can alleviate the need for domain-experts, which in turn can promote the practice within the software community. Machine learning methodologies can provide the required means to automate requirement classification.

2.4 Machine Learning

Machine learning (ML) is an application of artificial intelligence (AI) that provides systems the ability to automatically learn from characteristics of previous samples to be able to make data predictions [37]. Machine learning algorithms use statistics in an attempt to extrapolate patterns in massive amounts of data [36], making ML a powerful mechanism for solving problems rather than relying on humans to identify all the possible cases a system can handle [11].

Machine learning techniques and algorithms vary widely but they can be generally divided into two categories: supervised learning and unsupervised learning. Supervised learning refers to the category of learning algorithms that enables the processing and classifying of data using machine language. Supervised learning uses labeled data, which is a dataset that has been classified, to infer a learning algorithm. The dataset provides the model information of the number of classes and it lets the model focus on analyzing the features within the data that can distinguish these particular classes through the use of machine learning algorithms [38]. Unsupervised learning does not need any explicit labeling within the dataset, and is used to learn particular patterns in a way that reflects the statistical structure of the overall collection of patterns [37].

Machine learning can be utilized for a variety of domains where there can be large databases, including valuable implicit regularities, to be discovered. In requirement classification, as SRS documents provide large requirement data in text format, they have great potential to utilize machine learning techniques for the classification task at hand. This thesis will use supervised learning techniques for the identification and classification of functional and non-functional requirements [25].

Classification

Classification is a process of categorizing a given set of data into classes. The classification predictive modeling is the task of approximating the mapping function from input variables to determine output variables. The main goal is to identify which class/category the new data will fall into [27]. More formally, with an input vector x, a classification algorithm needs to formulate a function $f : R^n \to \{1, \dots, k\}$ so that $y = f(x)$, outputting the predicted class y [22].

As an example, heart disease detection can be identified as a classification problem. Since there can be only two classes i.e. a patient has heart disease or does not, it is a binary classification task. The classifier, in this case, needs training data to understand how the given input variables are related to the class. And once the classifier is trained accurately, it can be used to detect whether a particular patient has heart disease or not. If a classification task has more than two classes, it will be a multi-class classification.

Various types of learners, also known as classifiers, can perform the classification task. In the following subsections, those learners used in this thesis will be discussed rather than performing a deep dive into the mathematics or algorithms. Hence, the purpose of these sections is to provide a brief introduction to the nature and construction of these classifiers.

2.4.1 Support Vector Machine

The Support Vector Machine (SVM) is a supervised machine learning classifier that represents the training data as points in space. It separates these points into positive categories and negative ones by defining the optimal hyperplane that divides the two classes of data and maximizes the distance between the hyperplane and the closest data samples [39].

Given training examples as pairs $(\overrightarrow{x_i}, y_i)$, where $\overrightarrow{x_i}$ is the weighted feature vector of the ith training example and $y_i \in \{-1,1\}$ is the label of the example, the search for such a hyperplane can be expressed as an optimization problem of minimizing $\frac{1}{2} ||\vec{w}||^2$ subject to $y_i \, (\vec{w}.\vec{x}_i - b) \geq 1, \forall_i$, where $\vec{w}$ is a vector perpendicular to the hyperplane, and b defines the position of the hyperplane. As shown in Figure 2, the learned hyperplane is defined by a subset of positive and negative training examples, known as positive and negative support vectors respectively.

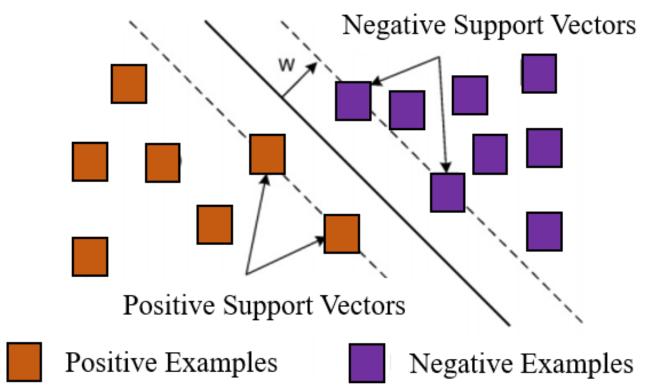


Figure 3: A linear separable Support Vector Machine

Once $\vec{w}$ and b are learned, SVM computes a score for unlabeled data represented by feature vector $\vec{x}$ using the decision function $f(\vec{x}) = \vec{w}.\vec{x} - b$. The sign of the score is used to predict the document label. That is, the document is labeled positive if $f(\vec{x}) \geq 0$, and negative otherwise [30].

2.4.2 Linear Logistic Regression

Logistic Regression (LR) is a probabilistic classifier that uses supervised machine learning to classify an input observation into one of two classes (binary classification) or into one of many classes (multi-class classification) depending on the type of the classification task at hand [40].

Let's consider a training corpus of input/output pairs $(\vec{x_i}, y_i)$ where each of a single input observation $\vec{x_i}$ is represented by a vector of features $[x_1, x_2, \ldots, x_n]$. The output y_i can be 1, which means the observation is a member of the class, or it can be 0, which means the observation is not a member of the class. We need to know the probability $P(y = 1|x)$ so we can decide if the observation is a member of the class or not [40].

LR solves this task by learning a vector of weights and a bias term from a training set. Each weight w_i is a real number, and is associated with one of the input features x_i. The weight w_i represents how important that input feature is to the classification decision, and can be positive or negative based on whether it is associated with the class or not. The bias term is also a real number that is added to the weighted inputs [40].

After learning the weights and the bias term in training, to decide on a test instance, Eq. 1 will be used to calculate a single number z, which expresses the weighted sum of the evidence for the class.

$$z = \left(\sum_{i=1}^{n} w_i \, x_i \right) + b \tag{1}$$

Then, to create a probability, z will be passed through a sigmoid function, which has the following equation, shown graphically in Figure 3.

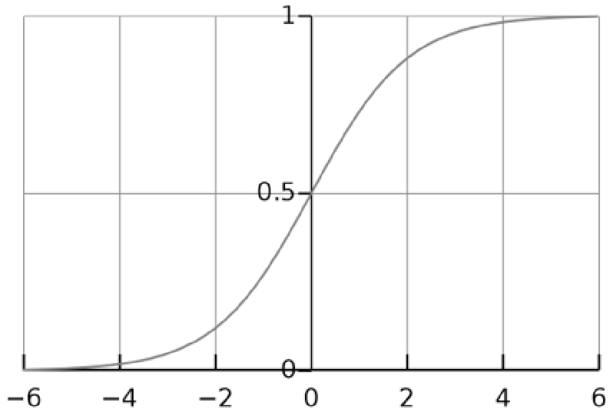


Figure 4: The sigmoid function y=1/(1+e^(−z) takes a real value and maps it to the range [0, 1]. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

As shown in Eq. 2, if the probability P (y = 1|x) is more than 0.5, the observation is a member of the class. Otherwise, it is not. Here, 0.5 is called a decision boundary.

$$\hat{y} = \begin{cases} 1 \; if \; P(y = 1|x) > 0.5 \\ 0 \; otherwise \end{cases} \tag{2}$$

2.4.3 Decision Tree

A Decision Tree (DT) algorithm is one of the supervised machine learning algorithms that learns decision rules from training data to perform a hierarchical partitioning of them [33]. There are both internal and leaf nodes in a decision tree classifier. In DT, all nodes with outgoing edges are internal nodes that correspond to an attribute. All other nodes are leaf nodes that correspond to a class. The decision predicates in the internal nodes will be based on the presence or absence of those attributes in the given data [41].

A decision tree contains two main processes: construction of the tree and classification. Forming a DT is a top-down building procedure starting at the root with the whole training dataset. The goal is to find the best test attribute in each decision node of the tree to reduce as much as possible the mixture of classes between each subset created by the test. This process will continue for each sub decision tree until reaching the leaves and determining their matching classes [35].

Classifying new data is based on the induced tree. Therefore, to classify an object, we begin from the root of the tree, we assess the corresponding test feature and we take the branch relative to the test's result. This process is repeated until a leaf is reached. The new object is then classified to the class labeling the leaf. There are several algorithms for building decision trees such as ID3 and C4.5 algorithm which are considered the most popular ones [35].

2.4.4 Naïve Bayes

The Naïve Bayes (NB) classifier is a probabilistic approach based on applying Bayes theorem [37]. Given the context of the class, the NB classifier assumes that all the features in data instances are independent of each other, which may not be true. The goal of the model is to find the class that maximizes the conditional probability for each class with the independence assumption.

Although the model seems oversimplified because of the independence assumption, it is known to work effectively on real-world problems [38]. This paradox is explained by the fact that classification estimation is only a function of the sign (in binary cases) of the function estimation; the function approximation can still be poor while classification accuracy remains high [39].

As this thesis involves textual requirements, we will explain NB in terms of document analysis. Given categories $C = \{c_1, \dots, c_{|C|}\}$ and documents $D = \{d_1, \dots, d_{|D|}\}$, the probability P that the document $\bar{d}_j$ belongs to category $\bar{c}_i$ is computed in Bayes Theorem [38, 41] as shown in Eq. 3.

$$P(c_i|\bar{d}_j) = \frac{P(c_i)P(\bar{d}_j|c_i)}{P(\bar{d}_j)} \tag{3}$$

Where document $\bar{d}_j = \langle w_{1j}, \dots, w_{|T|j} \rangle$ is represented by a vector of weights for each term in the vocabulary set T from all documents D. The topic of vector representation will be further elaborated upon.

$P(\bar{d}_j)$ is the probability that a random document in the corpus is represented by vector $\bar{d}_j$, and $P(c_i)$ is the probability that a random document in the corpus is of category c_i [28]. Because the number of possible variations for $\bar{d}_j$ is too high, computing $P(c_i|\bar{d}_j)$ is improbable. To ease this restriction, the above-explained independence assumption is applied [41]. Eq. 4 is the formula for the independence assumption [38]:

$$P(c_i|\bar{d}_j) = \prod_{k=1}^{|T|} P(w_{kj}|c_i) \tag{4}$$

2.4.5 Ensemble Methods

Ensemble method is a machine learning technique that combines several models called base learners to solve the same problem in order to produce one optimal predictive model [42]. For that purpose, ensemble techniques build a group of hypotheses to describe the data rather than finding one best hypothesis. Then, a voting approach will be applied to those hypotheses to predict the class of new data points [43].

More precisely, an ensemble method constructs a set of hypotheses $\{h_1, \dots, h_k\}$, chooses a set of weights $\{w_1, \dots, w_k\}$ and builds the voted classifier $H(x) = w_1 h_1(x) + \cdots + w_k h_k(x)$. The classification decision of the combined classifier is shown in Eq. 5 [43]:

$$H = \begin{cases} +1 & H(x) \geq 0 \\ -1 & H(x) < 0 \end{cases} \tag{5}$$

There are two major steps to build an ensemble learning algorithm. The first step is to build separate base learners in a way that the resulting set of learners is precise and diversified [44]. Being precise means each of the base learners should have fairly low error rate in predicting classes for new data points whereas being diversified refers to disagreement between the base learners in many of their predictions [42]. The generation process of the base learners can be parallel or sequential where the generation of one base learner affects the generation of subsequent learners. The second step in an ensemble-based system is the mechanism used to combine the individual classifiers [42].

There are many effective ensemble-based algorithms. The following will present the algorithms used in this thesis and will briefly explain how each of them works. For simplicity purposes, these methods will be explained considering the binary classification. Consider X and y as the instance space and the set of class labels, respectively, assuming $y \in \{-1, +1\}$.

Bagging

Bagging (short for Bootstrap Aggregation) algorithm is one of simple yet effective ensemble-based algorithms. Given a training dataset $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ where $x_i \epsilon X(i = 1, \dots, m)$, and $y_i \epsilon y(i = 1, \dots, m)$, bagging trains T independent base learners, each trained by sampling with replacement where the size of sampling is the same or the percentage of the training dataset size. The diversity in the ensemble is ensured by the variations within the bootstrapped replicas on which each classifier is trained. Also, it is reinforced by using a relatively weak classifier whose decision boundaries vary with respect to relatively small discrepancies in the training data [43]. Linear classifiers such as linear SVM could be a good candidate for this purpose. Bagging combines these learners by majority voting, and the most-voted class is predicted [43]. The pseudo-code of Bagging algorithm is provided in Figure 4.

```
Input: Data set $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$;
        Base learning algorithm S;
        Number of learning rounds T.
Process:
  for $t = 1, \dots, T$:
        $D_t = Bootstrap(D)$;    % Generate a bootstrap sample from $D$
        $h_t = S\ (D_t)$    % Train a base learner $h_t$ from the bootstrap sample
  end.
Output: $H(x) = argmax_{y \in Y} \sum_{t=1}^{T} l(y = h_t(x))$    % the value of $l(a)$ is 1 if $a$ is *true* and 0 otherwise
```

Figure 5: The Bagging Algorithm

Random Forest

One of creative variants of Bagging is Random Forest (RF) algorithm which is the tree-based ensemble classifier trained by the bagging mechanism [43]. In bagging, successive trees are built independently using a bootstrap sample of the training dataset and they do not depend on earlier trees. Random forests add an additional layer of randomness to bagging by incorporating random subset selection of features in addition to constructing each tree using a different data bootstrap [43].

More accurately, RF includes two major methods. The first method is using random feature subspace which enables a much faster construction of trees. The second is out-of-bag estimates which means, unlike standard trees, in which each node is split using the best split among all features, in RF, each node is split using the best among a subset of predictors randomly chosen at that node. This method enables the possibility of evaluating the relative importance of each input feature [44].

This strategy turns out to perform pretty well compared to many other classifiers as Breima demonstrated by running RF on 20 datasets from different data domains. His results showed that the performance of RF is superior to other learning algorithms such as SVM [47].

Extremely Randomized Trees

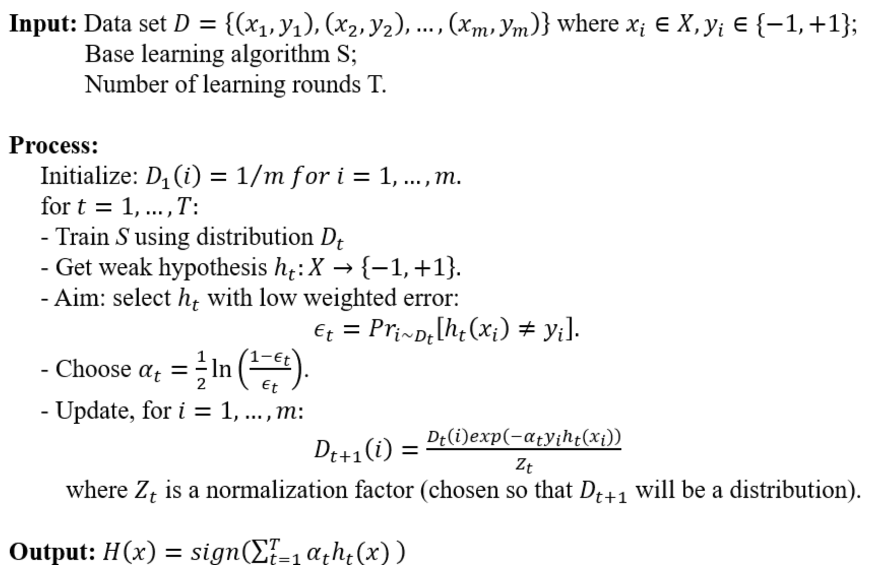
Extremely Randomized Tress also known as Extra Tree classifiers are ensemble learning methods fundamentally based on decision trees. The Extra Tree algorithm constructs an ensemble of unpruned decisions based on a top-down procedure [43]. This classifier is similar to RF in a sense that it randomizes certain decisions and subsets of data to reduce over-learning the data. However, there are two main differences between RF and Extra Tree: Extra Tree classifier splits nodes by explicitly randomizing cut-points, and to grow the trees, it uses the complete training set rather than a bootstrap sample [43].
It should be noted that randomly choosing cut-points while splitting a tree node, combined with ensemble averaging, diminishes the variance in extra tree classifier more than the weak randomization technique used in random forest. Also, using complete training dataset rather than bootstrap replicas will lead to minimized bias. Given the simplicity of the node splitting procedure, Extra Tree performs faster than random forest [43].

Boosting

Boosting is an ensemble method based on the idea of building a highly accurate classifier by combining many relatively weak and inaccurate classifiers [43]. Boosting uses simple majority voting similar to bagging; however, there is a major difference between the two ensemble techniques. In bagging, classifiers are trained in parallel on the bootstrapped replicas of the training data, which means all the instances in the original training dataset has the same chance of being used by each of the classifiers. Nevertheless, in boosting, individual classifiers will be trained sequentially, and the training dataset for each of the subsequent classifier will focus on where the previous classifier misclassified instances [45].

The most widely used boosting algorithm is AdaBoost which we will briefly explain here [51]. Consider we are given m labeled examples in a dataset $D = \{(x_1, y_1), \dots, (x_m, y_m)\}$, where $x_i \epsilon X(i = 1, \dots, m)$, and $y_i \in \{-1, +1\}$. On each round, $t = 1, \dots, T$, a distribution D_t is computed, and equal weights are assigned to all training examples.

From the training dataset and D_t, the algorithm generates a weak learner $h_t: X \rightarrow \{-1, +1\}$ by calling the base learning algorithm. Then, the algorithm uses the training examples to test h_t and based on the results, it increases the weights of the misclassified examples. Thus, the weight distribution will be updated as D_{t+1} with new attained weights. From the training dataset and D_{t+1} Ada Boost generates another weak learner by calling the base learning algorithm again. This process will be repeated for T times and weights of the weak learners are defined in the training process. Afterwards, the final learner is derived by weighted majority voting of the T weak learners [41]. The pseudo-code of AdaBoost is shown in Figure 5.



```
Input: Data set $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ where $x_i \in X, y_i \in \{-1, +1\}$;
       Base learning algorithm S;
       Number of learning rounds T.

Process:
    Initialize: $D_1(i) = 1/m \; for \; i = 1, \dots, m.$
    for $t = 1, \dots, T$:
    - Train $S$ using distribution $D_t$
    - Get weak hypothesis $h_t: X \rightarrow \{-1, +1\}$.
    - Aim: select $h_t$ with low weighted error:
                    $\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
    - Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
    - Update, for $i = 1, \dots, m$:
                    $D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
      where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).

Output: $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$
```

Figure 6: The AdaBoost algorithm

Extreme Gradient Boosting

Gradient Boosting is a machine learning technique that produces a model using ensemble of base prediction models, usually decision trees. It builds the model in a stage-wise mode like the other boosting methods, and generalizes base models by allowing optimization of a random variant loss function.

Extreme Gradient Boosting (XGBoost) builds on the idea of gradient boosting. Basically, the training is done using an additive strategy: Given an input with a vector of descriptors x_i, a tree ensemble model uses K additive functions to predict the output.

$$\hat{y}_l = \emptyset(x_i) = \sum_{k=1}^{K} f_k(x_i), \qquad f_k \in F_i \qquad 6$$

In Eq. 6, F_i is the set of all possible regression trees. The f_k function at each of K steps maps the descriptor values in x_i to a certain output. It is the function we need to learn, containing the structure of the tree and the leaf scores. However, there are minor improvements in the regularized objective which turned out to be helpful in practice. Specifically, XGBoost tries to minimize the regularized objective as shown in Eq. 7:

$$\mathcal{L}(\emptyset) = \sum\nolimits_i l(\hat{y}_l, y_i) + \sum\nolimits_k \Omega(f_k) \qquad 7$$

where $\Omega(f) = \gamma^T + \frac{1}{2}\lambda\|\omega\|^2$

In the equation above, l is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_l$ and the target y_i. The second term penalizes the complexity of the model in terms of number of leaves in the tree T and vector of scores on leaves ω. It helps to smooth the final learned weights to avoid over-fitting. We expect the regularized objective will tend to select a model employing simple and predictive functions [59].

2.5 Natural Language Processing

In this thesis, we are going to use requirements that are written in natural language. It should be noted that requirements can appear in a verity of forms such as lists of individual words, sentences, paragraphs, short texts potentially including special characters, and etc. [38]. Before applying machine learning techniques to this data, three main steps should be employed to transform the data into features that can be used by machine learning classifiers. The first step is preprocessing the text data using Natural Language Processing (NLP) techniques which we will cover in this section. The other two steps are feature extraction and feature selection which will be covered in the following sections.

2.5.1 Text Preprocessing

In natural language processing, text preprocessing is the practice of cleaning and preparing text data into a “well-defined sequence of linguistically meaningful units” [32]. This is the first step of transforming the data into a form that can be used by machine learning classifiers. Although NLP is relevant to all languages, the following subsections explore standard preprocessing techniques in English as we will use requirement documents written in English.

Case Folding

In text preprocessing, the case folding process aims to change all the letters in a text document into lowercase letters.

Removing Special Characters

Special characters are usually non-alphanumeric characters like []{}()/ that add extra noise to the dataset and should be removed in order to remove discrepancies during the assignment of polarity. For example, “That’s good:”— if the special character (:) is not removed, it might concatenate with the word (good) and make it unavailable in the dictionary [23]. To overcome such problems, it is best to remove special characters.

Tokenization

Tokenization is the process of splitting the text into words, phrases, symbols, or other meaningful elements called tokens. This involves defining the boundaries of a word, separating tokens by whitespace and punctuation, as well as splitting contractions [23].

Stop Word Removal

Many words in documents recur frequently but are essentially meaningless as they are used to join words together in a sentence. For example, some of the most common words such as “a,” “an,” or “the” do not influence the semantics of a software requirement. Due to their high frequency of occurrence, their presence in text mining presents an obstacle in understanding the content of the documents. Removing these words can help with this problem [23].

POS Tagging

POS tagging is the process of marking up a word in a corpus to a corresponding part of a speech tag based on its context and definition. This task is not straightforward as a particular word may have a different part of speech based on the context in which the word is used [23]. Consider this sentence as an example: “They refuse to permit us to obtain the refuse permit”; The word “refuse,” in this sentence is used twice and has two different meanings. The first “refuse” is a verb and it means deny while the second “refuse” is a noun and it means trash. Thus, POS tags of these two words will be different.

Stemming

Stemming is the process of reducing inflected words to their word stem. For example, the words: “presentation,” “presented,” and “presenting” could all be reduced to a common representation “present.” This is a widely used procedure in text processing for information retrieval based on the assumption that posing a query with the term “presenting” implies an interest in documents containing the words “presentation” and “presented” [23].

Lemmatization

Lemmatization is an alternative approach to stemming that considers the morphological analysis and the linguistic context of the term. Words appear in various forms flexed for grammatical reasons, but have the same meaning. The goal of lemmatization is to reduce flexural forms [23]. The word “better,” has “good” as its lemma. This link is missed by stemming while the word “walk,” is the base form for the word “walking”, and hence this is matched in both stemming and lemmatization.

2.5.2 Vector Representation

In order to extract features from documents to carry out classification, the text needs to be converted to some form of vector representation that provides quantitative characteristics of the text.

Bag of Words

The Bag of Words (BoW) vector representation model is one of the techniques to extract features from a text by representing unstructured text as numeric vectors, where it constructs a word presence feature set from all the words of an instance [5]. In this research, requirements have been converted into numeric vectors such that each document is represented by one vector (row).

TF-IDF

In addition to BoW, another vector representation is term frequency-inverse document frequency (TF-IDF) [41], which is formed by calculating the TF-IDF weight for each term *t* in document d by multiplying $tf_{t,d}$ and idf_t as shown in Eq. 8 where $tf_{t,d}$ is the frequency of word *t* in requirement document d and idf_t is the inverse document frequency of terms occurring in various documents. This weighting technique assigns a higher score to rare words and a lower one to words occurring frequently across all requirements [23].

$$tf_{t,d} = f_{t,d}$$

$$idf_t = log\frac{N}{df_t} \quad 8$$

$$tfidf_{t,d} = tf_{t,d}.idf_t$$

2.6 Model Validation

In machine learning, model validation is referred to as the process where a trained model is evaluated with a testing dataset. The testing dataset is a separate portion of the same dataset from which the training set is derived. The main purpose of using the testing dataset is to test the generalization ability of a trained model [38]. We have used cross validation to enhance the accuracy of the classifiers in this project.

2.6.1 Cross Validation

Statistical uncertainty can arise for small validation sets as a single validation set might not properly represent the dataset as a whole. A technique to counteract this dilemma is k-fold cross validation: repeating the training and testing procedure on k randomly selected, non-overlapping partitions of the dataset and taking the average score from all k folds [11]. A common choice for k is 10, resulting in ten validation trials. This strategy ensures that the classifier gets a chance to train and test on different portions of the dataset, reducing the influence of unique characteristics that might not be representative of the entire dataset.

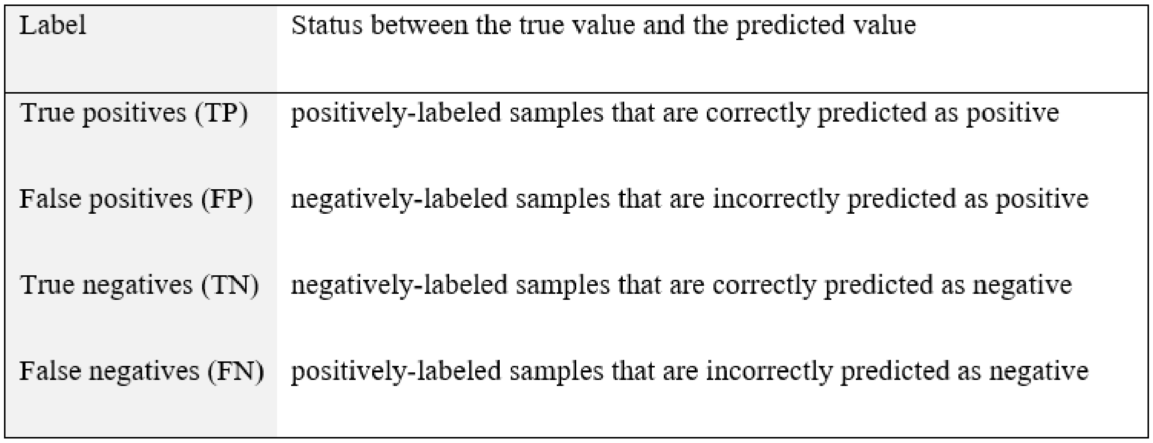
2.7 Performance Metrics

Machine learning tasks employ a suite of metrics to measure the performance of classifiers. The subsections below define the measures considered throughout this work.

2.7.1 Confusion Matrix

A confusion matrix is often used for binary classification tasks, showcasing how well the items in a validation set are classified and providing more details on the performance of the classifier. Table 2 displays the different labels a class prediction can take given the status between the true value and the predicted value.

Table 3: Class Prediction Labels given the status between the true valued the predicted value



Label	Status between the true value and the predicted value
True positives (TP)	positively-labeled samples that are correctly predicted as positive
False positives (FP)	negatively-labeled samples that are incorrectly predicted as positive
True negatives (TN)	negatively-labeled samples that are correctly predicted as negative
False negatives (FN)	positively-labeled samples that are incorrectly predicted as negative

2.7.2 Accuracy

Accuracy is the percentage of correctly classified samples overall. If N is the size of the validation set, then:

$$accuracy = \frac{TP + TN}{N} \qquad 9$$

Accuracy is a primitive measure as it does not give us information about how well a model is classifying a specific class. For example, if the validation set has two positive samples and eight negative samples, and the classifier predicts all ten samples to be negative, then it achieves a seemingly decent accuracy of 80%. However, upon closer inspection, the model classified everything as negative and failed to gather features differentiating the two classes, making it a weak classifier.

2.7.3 Recall and Precision

To counteract the inadequacies of the accuracy measure, machine learning studies often supplement their metrics with recall, precision, and their harmonic mean. The following definitions describe the metrics in terms of classifying the positive class.

Recall is the percentage of positively-labeled samples that are successfully predicted:

$$recall = \frac{TP}{TP + FN} \qquad 10$$

Precision is the percentage of positively predicted samples that are actually labeled positive.

$$precision = \frac{TP}{TP + FP} \qquad 11$$

In order to score models in multi-class classification, we need to caluclate these metrics differently using either micro-average scores or macro-average scores.

The micro-average precision and recall score is calculated from the individual classes' true positives (TPs), true negatives (TNs), false positives (FPs), and false negatives (FNs) of the model. The macro-average precision and recall score is calculated as arithmetic mean of individual classes' precision and recall scores.

Micro-Average and Macro-Average Precision Scores for Multi-class Classification:

For multi-class classification problem, micro-average precision scores can be defined as sum of true positives for all the classes divided by all of the positive predictions. The positives prediction is sum of all true positives and false positives. Eq. 12 indicates the precision micro average:

$$PrecisionMicroAvg = \frac{TP_1 + TP_2 + \cdots + TP_n}{TP_1 + TP_2 + \cdots + TP_n + FP_1 + FP_2 + \cdots + FP_n} \qquad 12$$

Macro-average precision score can be defined as the arithmetic mean of all of the precision scores of different classes. Eq. 13 indicates the precision macro average:

$$PrecisionMacroAvg = \frac{Prec_1 + Prec_2 + \cdots + Prec_n}{n} \qquad 13$$

Micro-Average and Macro-Average Recall Scores for Multi-class Classification:

For multi-class classification problem, micro-average recall scores can be defined as sum of true positives for all the classes divided by the actual positives (and not the predicted positives). Eq. 14 indicates the recall micro average:

$$RecallMicroAvg = \frac{TP_1 + TP_2 + \cdots + TP_n}{TP_1 + TP_2 + \cdots + TP_n + FN_1 + FN_2 + \cdots + FN_n} \qquad 14$$

Macro-Average recall score can be defined as the arithmetic mean of all the recall scores of different classes. Eq. 15 indicates the recall macro-average:

$$RecallMacroAvg = \frac{Recall_1 + Recall_2 + \cdots + Recall_n}{n} \qquad 15$$

2.8 Tools

To assist in our research endeavors, we fortunately have access to well-developed opensource tools for document processing, classifier construction, and model training.

2.8.1 Natural Language Processing Toolkit

Natural Language Processing Toolkit (NLTK)[1] is a platform for building python programs to work with NLP tasks. NLTK provides a suit of text processing libraries for classification, tokenization, stemming, lemmatization, tagging, and etc.

2.8.2 Scikit-Learn

Scikit-Learn[2] is a well-established machine learning package available for Python programs. The library includes a rich suite of machine learning implementations, allowing us to employ their versions of traditional classifiers such as Linear Regression, Decision Tree, etc. *Scikit-Learn* also supplies many utility functions for data preprocessing, model validation, and metric computations.

2.9 Datasets

We have used three datasets in this research. In the next subsections, we will provide more details about each of them. Three different types of classification tasks can be done on these datasets: (1) binary classification of NF versus F requirements, (2) binary classification of NFR type, and (3) multi-label classification of various NF requirement types.

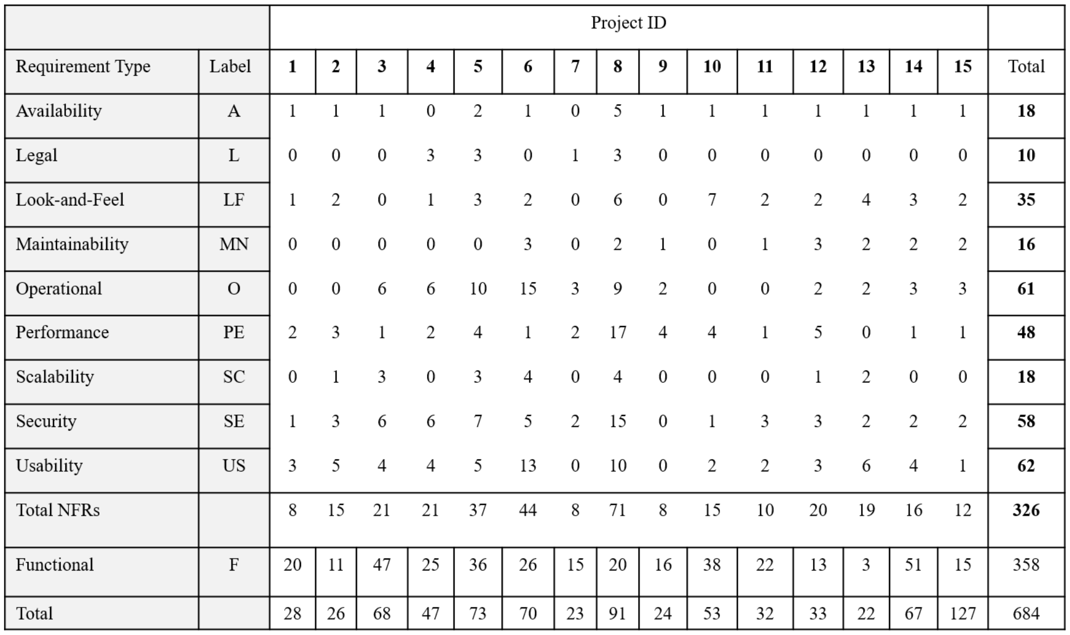
---

[1] https://www.nltk.org/

2 https://scikit-learn.org/stable

2.9.1 Quality Attributes Datasets

Table 4: PROMISE dataset, broken down by project and requirements type



		Project ID															
Requirement Type	Label	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	Total
Availability	A	1	1	1	0	2	1	0	5	1	1	1	1	1	1	1	**18**
Legal	L	0	0	0	3	3	0	1	3	0	0	0	0	0	0	0	**10**
Look-and-Feel	LF	1	2	0	1	3	2	0	6	0	7	2	2	4	3	2	**35**
Maintainability	MN	0	0	0	0	0	3	0	2	1	0	1	3	2	2	2	**16**
Operational	O	0	0	6	6	10	15	3	9	2	0	0	2	2	3	3	**61**
Performance	PE	2	3	1	2	4	1	2	17	4	4	1	5	0	1	1	**48**
Scalability	SC	0	1	3	0	3	4	0	4	0	0	0	1	2	0	0	**18**
Security	SE	1	3	6	6	7	5	2	15	0	1	3	3	2	2	2	**58**
Usability	US	3	5	4	4	5	13	0	10	0	2	2	3	6	4	1	**62**
Total NFRs		8	15	21	21	37	44	8	71	8	15	10	20	19	16	12	**326**
Functional	F	20	11	47	25	36	26	15	20	16	38	22	13	3	51	15	358
Total		28	26	68	47	73	70	23	91	24	53	32	33	22	67	127	684

The Quality Attributes dataset[3], also known as the PROMISE corpus, is a compilation of requirements specifications for 15 software projects developed by MS students at DePaul University as a term project for a Requirements Engineering course [12]. The dataset consists of 326 non-functional requirements of nine types and 358 functional requirements. The distribution of requirement types of these 15 projects is shown in Table 3; and Table 4 provides examples of each type of requirement.

[3] http: //openscience.us/repo/requirements/requirements-other/nfr.html.

Table 5: Examples of requirements of different types from PROMISE dataset

Label	Requirement Text
A	"Aside from server failure the software product shall achieve 99.99% up time."
L	"The product shall comply with the estimates laws relating to recycled parts usage."
LF	"The look and feel of the system shall conform to the user interface standards of the smart device."
MN	"Application updates shall occur between 3AM and 6 AM CST on Wednesday morning during the middle of the NFL season."
O	"The product shall run on the existing hardware for all environments."
PE	"The search for the preferred repair facility shall take no longer than 8 seconds. The preferred repair facility is returned within 8 seconds."
SC	"System shall be able to handle all of the user requests/usage during business hours."
SE	"Only valid data shall be entered into the system. No invalid data shall be entered into the system."
US	"Users shall feel satisfied using the product. 85% of all users will be satisfied with the product."
F	"A non-clinical class shall specify the course name lecture room requirements and instructor needs."

2.9.2 PURE: Dataset of Public Requirements Documents

Another dataset used for this research was PURE (PUblic REquirements) dataset, which is originally a dataset of 79 publicly available natural language requirement documents collected from the web [46]. The documents cover multiple domains and range from university projects to documents of public companies. The dataset can be used for different natural language processing tasks that are typical in requirements engineering, such as model synthesis, abstraction identification, document structure assessment, ambiguity detection, and requirement classification. In this research, for the purpose of requirement classification, we have used six documents provided in this dataset which included the functional and non-functional requirements of the system to which they belong.

The distribution of requirement types of these six projects is shown in Table 5, and examples of each type of requirement is provided in Table 6.

Table 6: PURE dataset, broken down by project and requirements type

	Project ID						
Requirement Type	**1**	**2**	**3**	**4**	**5**	**6**	Total
Help Module	2	0	0	0	0	0	**2**
Support Module	5	0	0	0	0	0	**5**
Audit Module	8	0	0	0	0	0	**8**
Access Module	10	0	0	0	0	0	**10**
Ease of Use	9	0	0	0	0	0	**9**
Usability	47	6	1	0	0	0	**54**
Availability	6	1	1	0	0	0	**8**
Performance	4	9	0	2	0	3	**18**
Scalability	1	0	0	0	0	0	**1**
Safety	0	1	0	2	0	1	**4**
Security	0	6	1	1	7	2	**17**
Efficiency	0	5	0	0	0	0	**5**
Maintainability	0	4	1	0	0	2	**7**
Portability	0	2	1	0	0	0	**3**
Testability	0	4	0	0	0	0	**4**
Reliability	0	0	0	0	0	5	**5**
Total NFRs	92	38	5	5	7	13	**160**
Functional	7	6	25	30	27	41	136
Total	99	44	30	35	34	54	296

2.9.3 Case Study Dataset

We have also used a separate dataset which is extracted from a company's Subsystem Requirements Specification (SSRS) document. The SSRS is a structured collection of requirements such as functions, performance, design constraints, and attributes which describe a specific subsystem, its operational environments, and external interfaces. The SSRS establishes the scope for this system, and is used as a basis for its development. It is also a direct input to the subsystem and software architecture phases and is used as the basis for developing test cases and procedures. An example of NFR sentence of the case-study dataset is shown in Figure 6 and the distribution of requirement types of this dataset is shown in Table 7.

Table 7: Examples of requirements of different types from PURE dataset

Requirement Type	Requirement Text
Help Module	“The help should be accessible to the users both in the offline and online mode.”
Support Module	“The support solution should be accessible to the users both from within the application and also outside the application through a browser interface.”
Audit Module	“The System must maintain the audit trail for as long as required, which will be at least for the life of the case to which it refers.”
Access Module	“If a user performs a quick or advanced search, the System must never include in the search result list any record which the user does not have the right to access.”
Ease of Use	“The System must employ a single set of user interface rules, or a small number of sets to provide a familiar and common look and feel for the application.”
Usability	“Links that open new browser windows or pop-up windows should be clearly marked.”
Availability	“Unplanned downtime for the System must not exceed 1 hour per rolling three month period.”
Performance	“Upon the USB being plugged in the system shall be able to be deployed and operational in less than 1 minute.”
Scalability	“The System should be saleable and must not have any features which would preclude use in small or large police stations, with varying numbers of cases handled.”
Safety	“The system will do periodic backups through a live internet connection.”
Security	“The system shall detect consecutive failed login attempts.”
Efficiency	“The system shall automatically compress image files that are too large in size.
Maintainability	“The system shall utilize interchangeable plugins.”
Portability	“The system shall be easy to migrate or backed up via another USB drive.”
Testability	“The system should be able to create test environment of web order system.”
Reliability	“The Digital Home System must be highly reliable with no more than 1 failure per 10,000 hours of operation.”
Functional	“The THEMAS system shall control the heating and cooling units that are designated as part of the THEMAS system.”

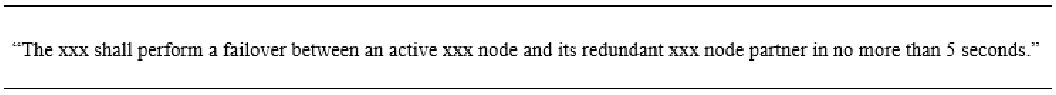

“The xxx shall perform a failover between an active xxx node and its redundant xxx node partner in no more than 5 seconds.”

Figure 7: An Example of NFR in the Case-Study Dataset

Table 8: Case Study dataset extracted from BOS SSRS document, broken down by project and requirements type

Requirement Type	Total
External Interface	169
Security	38
Scalability	15
Performance	12
Reliability	9
Availability	9
Safety	5
Compatibility	5
Total NFRs	262
Functional	1559
Total	1821

Chapter 3

3 Related Work

Informal textual descriptions written in natural language are a prevalent means for requirement specification in early stages of software projects [39]. These textual descriptions include both functional and non-functional requirement statements. Classification of requirements into functional and non-functional classes as well as categorization of NFRs into specific types such as performance, security, availability, etc. allows filtering relevant requirements based on stakeholder concerns. A number of attempts have been made to automate the process of detection and classifying requirements from requirement documents. In this section we will go through the related research works in this area.

3.1 NFR Classification Studies

In 2006, Cleland-Huang et al. performed the first notable exploration of NFR extraction and classification from software development artifact. In this work, they attempted to detect and classify NFRs from 15 SRS documents developed as term projects by MS students at DePaul University [12]. Their approach assumes that different types of NFRs can be distinguished by certain keywords as indicator terms. Thus, they trained some NFR-specific terms for each of the NFRs from the training data, and then tried to classify each given document as one or more classes of NFRs according to the function of the occurrence of indicator terms. The model worked well for detecting the most of the NFRs appearing in the text, with a classification recall of 0.81. However, the precision was low (0.12) due to a high rate of false positives. They also made their dataset available to the PROMISE repository, and since then, it has been used by other researchers to build their NFR classification models and compare it to the original work.

In 2008, Hussain et. al proposed a classification approach to automate the process of detecting NFR sentences by using a text classifier equipped with a part-of-speech (POS) tagger [47]. They used the NFR PROMISE dataset provided by [12] for their classification task. The classifier works at sentence level as the characteristics of FR and NFR remain within the scope of sentences. The results reported in this paper outperformed the previous work [12] in the field and they attained a higher accuracy of 98.56% using 10-folds-cross-validation over the data used by [12]. The investigated results showed that the method significantly improved the classification performance. Nonetheless, their work did not classify different types of NFRs and they only classified requirements into functional and non-functional.

In 2010, using the NFR PROMISE dataset, Casamayor et al. replicated the study of Cleland-Huang et al. [12] using a semi-supervised approach for the automatic identification and classification of NFRs, so as to decrease the need for manually labeled data and to improve the results of the original work [11]. They used a multinomial naïve Bayes classifier coupled with an Expectation Maximization (EM) [48] algorithm to boost the classifiers performance. To this end, they provided labels for each instance of unlabeled dataset on the function learned by the small labeled training set. The results provided by this research work approached the maximum theoretical performance on the dataset with a given algorithm. Empirical evaluation of their approach showed higher classification accuracy of about 0.75 compared to the cases in which a fully supervised approach was used.

In 2011, Zhang et al. conducted an empirical study on using text mining techniques to classify NFRs automatically [39]. They used NFR PROMISE dataset for their experiments. Three kinds of index terms, which were at different levels of linguistical semantics, as N-grams, individual words, and multi-word expressions (MWE) were used in the representation of NFRS. Then, they utilized Support Vector Machine (SVM) with linear kernel as their classifier. Their results showed that index term as individual words with Boolean weighting outperformed the other two index terms and using MWEs did not enhance representation of individual words significantly. Also, they observed that automatic classification produces better performance on categories of large sizes than that on categories of small sizes, and they concluded that based on their experimental results for automatic classification of NFRs, individual words are the best index terms in text representation of short NFRs' description. They also surmised that they should collect as many as possible NFRs of software system. Compared to the original study performed by

Cleland-Huang et al. [12], they reported higher precision but lower recall on the same dataset.

In 2013, Slankas et al. proposed a tool named NFR-Locator to identify NFR-related sentences in a text document in two steps [5]. In the first step, natural language text was parsed into an internal representation called Sentence Representation (SR) based on the Stanford Type Dependency Representation (STDR) [49]. SR is a tree-like representation in which each sentence is represented as a directed graph where vertices are words and edges are relationships between words. In the second step of the process, SRs are used to classify the sentence into specific NFR categories or not applicable (NA) category if they do not specify any NFRs. They proposed a modified version of the k-nearest neighbor (k-NN) classifier that computed a custom distance function based on similarity of SRs and assigns NFR categories based on similar sentences.

The highest accuracy was achieved by using Sequential Minimum Optimizer (SMO), with F1-measure of 0.60. The maintained that k-NN achieved an F1 measure of 0.54, outperforming the optimal Naïve Bayes classifier with F1 measure of 0.32. They found that all of the evaluated documents contained NFRs; however, they also found that the types of NFRs available in each document could vary. For example, DUA documents contained high frequencies of legal and privacy NFRs compared to other types of documents. Moreover, in analyzing specific NFR categories, they found particular features unique to each category that made them suitable candidates to use as a feature in classification and to improve the accuracy of the models.

In 2014, Riaz et al. developed a tool-assisted process called Security Discoverer (SD) that took as input a set of natural language requirement artifacts and a trained classifier [50]. Their proposed process automatically identified security-relevant sentences in the artifacts and classified them according to the security objectives, either explicitly stated or implied by the sentences. They executed a variety of classifiers including K-Nearest Neighbor, Multi Nominal Naïve Bayes, and Sequential Minimal Optimization (SMO) classifier on the document set. They used a stratified 10-fold cross validation and computer the precision, recall, F1-score measure in order to test each of the considered classifiers. In this work, they classified 10963 sentences in six different documents from healthcare domain and extracted corresponding security objectives. Their proposed tool could correctly classify 82% of the security objectives for all the sentences (precision) and identified 79% of all security objectives implied by the sentences within the documents (recall).

In 2016, Jindal et al. gathered a total of 58 security-based specifications from the NFR PROMISE dataset [51]. The selected security requirement descriptions were then further analyzed by applying a series of text mining steps. They applied preprocessing techniques including tokenization, stop word removal, and stemming. In this work, they used Information Gain measure which was based on ranking all the selected features retrieved from preprocessing, and thereafter they selected the top 'N' features on the basis of their rank. As the final step of text mining, TF-IDF weighting approach was used to assign an appropriate weight to all 'N' ranked features.

Then, they used J48 decision tree method to classify all 58 security-based descriptions into four types of security requirements: authentication authorization (AA), access control (AC), cryptography encryption (CE) and data integrity (DI). Corresponding to each type of security requirement, a binary prediction model was developed and their performance was evaluated using AUC value and sensitivity. Their results showed that the model corresponding to DI outperformed the other three models (i.e. AA, AC, and CE) with AUC and sensitivity of 0.83 and 80%. And the models corresponding to AA and AC achieved moderate results with the values of AUC being 0.72 and 0.77 respectively. CE model did not perform very good with AUC of 0.69.

In 2017, Lu et. al proposed automatic requirements classification approach to classify app user reviews into four types of NFRs (i.e. reliability, usability, portability, and performance), functional requirements, and others [52]. This approach combined four classification techniques including BOW, TF-IDF, CHI2 and AUR-BOW with three machine learning algorithms Naïve Bayes (NB), J48, and Bagging to automatically classify user reviews. Their results showed that the combinations of AUR-BOW with Bagging achieved the best result with a precision of 0.71 and a recall of 0.72 and they concluded that machine learning algorithm Bagging was more suitable for NFRs classification from user reviews than Naïve Bayes and J48. This research reported that in an automatic classification, using imbalanced dataset performed worse when the size of certain types was obviously smaller. Nevertheless, no solution has been provided for the mentioned imbalanced dataset problem.

In 2017, Kurtanović and Maalej [34] proposed a supervised machine learning approach using NFR PROMISE dataset for the classification of FRs and four subcategories of NFRs (i.e. usability, security, operational, and performance). They developed a supervised machine learning approach employing various features including meta-data, lexical, syntactical, bag of words, bigrams and trigrams, filtering stop-words, punctuation, and POS. They employed under- and over- sampling strategies to handle the imbalanced classes in the dataset and cross-validated the classifiers using precision, recall, and F1 metrics in a series of experiments based on the Support Vector Machine classifier algorithm. They achieved a precision and recall up to about 0.92 for automatically identifying FRs and NFRs. For the identification of specific NFRs, they achieved the highest precision and recall for security and performance NFRs with 0.92 precision and 0.9 recall. In their paper, they demonstrated that part of speech tags were the most informative features.

In 2017, Deocadez et al. applied three self-labeling algorithms including self-training, RASCO (Random Subspace Method for Co-training), and Rel-RASCO (Relevant Random Subspace Co-training) as Semi-Supervised Classification (SSC) techniques in order to automate the classification of requirements in reviews of App Store [12]. They collected 932388 reviews from App Store after which they performed stratified random sampling to select 300 reviews as their ground-truth set and manually categorized them using FURPS (Functionality, Usability, Reliability, Performance, and Supportability) model. They preprocessed their dataset by applying standard text mining techniques of Weka tool, removing numbers, removing two-letter words and other symbolic characters, which resulted in a total of 212 attributes. They performed their experiments using 10-fold cross validation and applying self-training, RASCO, and Rel-RASCO. Each of these SSC algorithms was executed with K-Nearest Neighbor, C4.5, Naïve Bayes, and SVM with SMO. They used accuracy as their evaluation metric and evaluated the classification performances of their classifiers with four different training ratios (10%, 30%, 50% and 70%). Their results indicated that the performance of Self training, RASCO, and Rel-RASCO with C4.5, SMO, and NB increased as the labeled ratio went up from 10% to 30% after which point it became stable. The Self-Training algorithm consistently reinforced its performance accuracy as the labeled ratio escalated regardless of the used base learner.

In 2017, Shakeri et. Al contributed an approach for preprocessing requirements to reduce the inconsistency of requirement specifications before applying classification algorithms [53]. They used NFR PROMISE dataset for their work. To preprocess the data, they first applied Hussain et al. approach of POS-tagging-based feature extraction technique [47]; then they applied entity tagging to blind all context-based products and users by assigning them names as PRODUCT and USER, respectively; then they applied temporal tagging to recognize and normalize temporal expressions; and finally, they used regular expressions to increase the weight of the influential words for each type of NFRs. They performed topic modeling techniques using Latent Dirichlet Allocation (LDA) algorithm, and Biterm Topic Model (BMT) method; and Clustering using Hierarchal and K-means algorithms; and Naïve Bayes classification algorithm. Their preprocessing approach improved the accuracy of F-NFR classification from 89.92% to 95.04% and it improved the performance of all above-mentioned classification methods for NFR classification especially for LDS and BMT (their precision and recall doubled). Among the machine learning algorithms: LDA, BTM, Hierarchical, K-means, Hybrid and Binarized Naïve Bayes, Binarized Naïve Bayes achieved the highest performance for sub-classifying NFRs.

In 2018, Tóth and Vidács conducted a comparative study of the performance of various classifiers in labeling non-functional requirements [54]. They used NFR PROMISE dataset for their experiments and preprocessed it by removing punctuation characters and stop words and stemming the remaining words using Porter Stemmer process. Then they converted the dataset to TF-IDF representation. Different classifiers including Multinomial-, Gaussian- and Bernoulli Naïve Bayes, Support Vector Machine with linear kernel, Linear Logistic Regression, Label Propagation, Label Spreading, Decision Tree, Extra Tree, Extra Trees as an ensemble method, K-Nearest Neighbor as well as Multi-Layer Perceptron methods were applied on the dataset. They executed their experiments using 10-fold cross validation and evaluated them by precision and recall complemented with execution time. Their results showed that classifiers like Support Vector Machine with precision of 0.89 and recall of 0.65, and Linear Regression with precision of 0.87 and recall of 0.67 produced the best result. However, considering execution time in addition to precision and recall, Multi Nominal Naïve Bayes with the best execution time and with precision of 0.87, and recall of 0.68, was the best choice in practice for classification of requirement sentences.

In 2019, Haque et al. conducted an empirical study using NFR PROMISE dataset to automatically classify NFRs to find out the best pair of feature extraction techniques and machine learning algorithms [55]. They combined four feature extraction techniques including BOW, TF-IDF (character level), TF-IDF (word level), and N-gram with seven machine learning techniques consisting of Multinomial-, Gaussian- and Bernoulli Naïve Bayes, K-Nearest Neighbor, Support Vector Machine, SGD SVM, and Decision Tree algorithms. To train their models, they first preprocessed the data by removing special characters, case-folding, removing stop words, and tokenization, and then they applied the mentioned classifiers. They measured the performance of these classifiers with statistical analysis including precision, recall, F1-score, and accuracy. They found that, SGD SVM classifier achieved best results where precision, recall, F1-score, and accuracy of 0.66, 0.61, 0.61, and 0.76. Additionally, TF-IDF (character level) feature extraction technique illustrated higher average score than others. The highest precision, recall, and F1-score was reported by SGD SVM technique for all feature extraction methods. However, SGD SVM best performed with TF-IDF (character level) with precision, recall and F1-score of 0.70, 0.62, and 0.63.

In 2020, Kobilica et al. [56] conducted an empirical study to evaluate the performance of different supervised machine learning algorithms such as Tree based techniques, LR, NB, SVM, and Ensemble-based techniques such as Boosted Trees and Bagged Trees in classifying security requirements, using the publicly available SecReq dataset. They also experimented with two deep learning techniques including Recurrent Neural Network (RNN)-based Long Short-Term Memory (LSTM) network and various k-nearest neighbors (KNN). They specifically focused on the analysis of the fast preprocessing techniques.

They combined the above-mentioned classifiers with two fast text preprocessing techniques including word encoding and word embedding. In order to avoid overfitting and to have better generalization of the results, we trained all classifiers using 10-fold cross validation.

Their results showed that LSTM network achieved the best accuracy (84%) among the unsupervised learning algorithms. Considering LSTM’s nature as a deep learning approach and its longer training time, it might not be suitable for fast classification task. Whereas Boosted Ensemble achieved the highest accuracy (80%), among supervised algorithms, it had one of the worst training time (18 seconds). That said, the authors pointed out that Weighted-KNN achieved an accuracy value of 71% within less than a second of training time. Such a result could potentially indicate that KNN-based classification should be considered first in security requirement classification tasks.

3.2 Analysis of Related Work

Going through the related work, we can see that there have been several attempts in the classification of non-functional requirements using machine learning approaches. However, there are a few main challenges visible in these works. In most cases, these challenges have not been mentioned, or even if these problems have been recognized by the researchers, no specific solution has been offered. As it is shown in [57], between different techniques of machine learning approaches, supervised, semi-supervised, and unsupervised, supervised learning is the most popular type of ML in the area of requirement classification. Therefore, we felt a need to deal with some of the challenges that already existed in this field by doing a comprehensive comparative study of non-functional requirement classification using supervised machine learning techniques. Although the published research works have attained promising results, our research improves upon these results significantly as will be shown in the later chapters. Analysis of the available studies in identification and classification of non-functional requirements reveals a general process pattern for applying machine learning-based approach to identify and classify NFRs in a textual document [57]. This process is divided into three major phases which is shown in Figure 7.

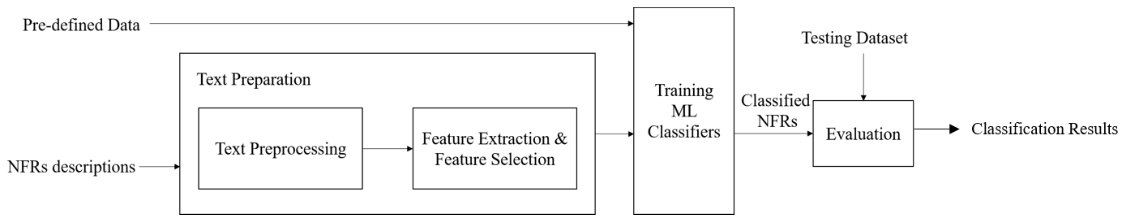


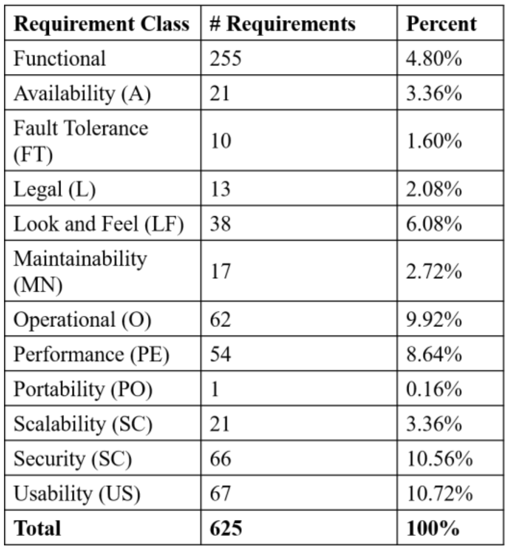
Figure 8: A general process of applying ML algorithms to classify NFRs in textual requirements documents.

As illustrated in Figure 7, pre-defined data is a precondition for building NFRs classifier using ML algorithms. Supervised Learning algorithms requires a labeled dataset. Additionally, there is a text preparation phase, which involves applying natural language text processing techniques and ML algorithms; and the evaluation phase, which is concerned with assessing an ML algorithm's approach to classify NFRs. We will get into more details of the process in methodology section.

Predefined Dataset

Since 2006, when Cleland-Huang et al. made their dataset publicly available as NFR PROMISE dataset, this dataset has been used by most of the researchers in the field for the task of non-functional requirement classification [18]. This dataset consists of 15 SRS documents developed as term projects by MS students at DePaul University. Although NFR PROMISE dataset is still a valuable dataset today, and it encouraged researchers to work on the task of NFR classification, the requirements were originated from university student projects, not from requirement documents of a production system [58]. Table 8 presents an overview of the NFR PROMISE dataset that includes 12 classes and 625 requirements. The table shows that some classes of the NFR dataset are underrepresented. The NFRs Portability (PO) and Fault Tolerance (FT) are very rare in the dataset.

Table 9: Overview of PROMISE dataset with the number of requirements available in each class



Requirement Class	# Requirements	Percent
Functional	255	4.80%
Availability (A)	21	3.36%
Fault Tolerance (FT)	10	1.60%
Legal (L)	13	2.08%
Look and Feel (LF)	38	6.08%
Maintainability (MN)	17	2.72%
Operational (O)	62	9.92%
Performance (PE)	54	8.64%
Portability (PO)	1	0.16%
Scalability (SC)	21	3.36%
Security (SC)	66	10.56%
Usability (US)	67	10.72%
Total	**625**	**100%**

Potential issues with such datasets are the result of the absolute rarity of some concepts as well as the within-class imbalances [60]. The issue with rare instances of a target class can make the classification difficult. When the concept itself has a sub concept with limited instances, additional difficulty might arise when classifying a minority concept due to within-class imbalances [60]. This is the first main issue using PROMISE dataset for the classification task.

Moreover, the total number of requirements in PROMISE dataset is 625 out of which 255 are functional requirements and 370 are non-functional requirements. There is a common belief in machine learning that there is not enough data and more data gives more benefits [59]; however, this cannot accurately be validated till the results of a research work can be tested utilizing larger datasets. That said, as illustrated in [34], training a classification model for classifying NFRs using PROMISE dataset, the curves of the training scores and cross validation scores got closer to each other with a larger training size and the model's performance stabilizes with a training size bigger than 400 requirements. Small number of requirements in the NFR RPOMISE dataset and not utilizing any techniques to expand the training dataset are the second concern related to using this dataset.

Evaluation Phase

In this phase, the evaluation of the NFR classifier is conducted using various techniques to determine the developing classifier's effectiveness. In the classification of NFRs using supervised machine learning algorithms, the common technique is to use K-fold cross validation for which the input dataset is randomly divided into K folds. Each fold has the same data size; one is used for testing, while the others are used for training. The learning algorithm is run K times, and the average K results are calculated to produce a single result to measure the NFR classifier's performance. Considering performance evaluation metrics used in this field, there are a few noticeable issues. As it is shown in [18], precision and recall are the two most common evaluation metrics used in the classification of NFRs.

Precision measures the total number of correctly classified NFRs in respect to the number of NFRs retrieved. It is defined as P = true positive / (true positive + false positive). Recall measures the percentage of NFRs that were correctly classified and is defined as R = true positive / (true positive + false negatives) [12]. True positive represent the number of correctly classified requirements, false positive represent the number of incorrectly classified requirements, true negatives represent the number of requirements correctly not classified, and false negatives are the number of requirements incorrectly not classified [3].

Precision and recall are often used together [48] and there is a trade-off between them. Precision ensures that all retrieved requirements are truly relevant, whereas recall focuses on retrieving all relevant requirements. It is arguable which value is more important to measure the classification results. [12] focused on achieving a high recall (76.7%) but lower precision (12.4%) as it was believed that rejecting irrelevant requirements manually from a set of retrieved requirements was easier than reading an entire document looking for missing requirements. On the other hand, [39] argued that precision was more important if recall was acceptable for automatic classification to avoid large amounts of irrelevant NFRs from being misclassified as relevant (false positive), which might frustrate users. There are some studies that achieved higher recall and lower precision [12, 25], while there are other studies that achieved higher precision and lower recall such as [5]. There is a study which achieved high recall and high precision [47]; however, this study only used a machine learning approach to detect NFRs and could not classify them.

Furthermore, in industry, we will encounter large-scale projects with thousands of requirement data. Hence, in order to provide stakeholders with an efficient tool for concern-based NFR classification, execution time is of paramount importance. Among all the studies that have been done so far, only two recent works have included execution time as a complementary performance metric [61]. A comparison of execution times between our work and the related work is provided in Chapter 5.

That said, there is a potential to use classification techniques with the propose of improving recall and precision metrics and achieving high values in both of them. Also, including execution time as a complementary evaluation metric will enable us to propose the best strategy for the classification of non-functional requirements.

Industrial Dataset

Supervised Learning requires a set of NFRs that were correctly classified to their categories. Considering that the goal of classifying NFRs into their specific types is to propose solutions to be used by practitioners in the industry, one of the best sources to use in either training the classifiers or validating the proposed tools is industrial dataset. As indicated in [18], out of 24 studies the researchers reviewed in the field of requirement classification, only five studies had used industrial datasets. Out of those five, only one was focused on the classification of non-functional requirements using supervised machine learning techniques. This shows the potential to do a case study using industrial dataset to validate the achieved results.

Ensemble Techniques

Reviewing the related works in this field, there have been a few works utilizing ensemble techniques which can produce more accurate results by creating multiple models and combining them together. In the recent study [17], the researchers included ensemble techniques to classify a dataset of 58 security requirements and their results indicated that ensemble classifiers can perform well. To the best of our knowledge, only one recent work [7] used Extra Trees as an ensemble technique for the identification of non-functional requirements. This indicates a potential to investigate ensemble techniques such as bagging trees, boosted trees, random forest, etc.

With all that said, there is a need for a study that can propose a solution using supervised machine learning algorithms while handling the mentioned problems such as using more balanced dataset with greater number of requirement instances, working toward achieving both high recall and precision values, providing execution times of the techniques, etc.; which can be trusted to some level if not completely, considering the fact that there is more space for more validation techniques.

- In this work, we use the combination of two datasets. First, the NFR PROMISE dataset for the purpose of comparison with other works and second the PURE dataset which is composed of publicly available industrial requirement specifications documents. By using the combination of these two datasets, we achieved a fairly balanced dataset with more than 500 requirements.
- Our proposed strategy enables the classifiers to achieve high precision and high recall with more focus on recall values. We will discuss the reasoning although there is always more space to argument on the prioritization of these two metrics.
- We cover a fairly large set of combinations of feature extraction techniques including syntactic POS-tagging-based adopted from [2], TF-IDF, and BOW, with supervised machine learning techniques including SVM, SGD SVM, DT, RR, GNB, MNB, and BNB and ensemble techniques including Extra Trees, Bagging DT, Boosted DT, Rand RF. This can enable comparisons with related works and potentially be a good comparative study in the field.
- Our study also includes a case study by using an industrial dataset to validate our results which is required in this field considering that lack of industrial data as one of the main challenges in this field [18].

Chapter 4

4 Research Methodology

The goal of this research is to classify NFRs according to the stakeholders' concerns. In this research work, we narrowed our focus to four non-functional requirements including usability, security, performance, and operation quality attributes, since they are the most common types of NFRs that have been mentioned in our dataset. Therefore, given a dataset of all requirements, we aim to first identify which requirements are functional and which ones are non-functional and then we will further classify the non-functional requirements into four mentioned types. By identifying and classifying the requirements written in natural language successfully, we will be able to take the first step to provide stakeholders with their concern-based non-functional requirements. It is important to note that in this work, the quality attributes refer to the product quality aspects. In the future, we hope to extend this work to encompass process quality aspects such as risk, cost, and effort.

This thesis asks the following two research questions:

3. RQ1: How effective are supervised machine learning techniques in automatically identifying FRs from NFRs and classifying 4 types of NFRs including usability, performance, security, and operation NFRs?
4. RQ2: Which combination of feature extraction techniques and supervised machine learning algorithms gives the best results for RQ1?
    a. What are the most informative features for each of usability, security, performance, and operation NFR types?

By answering these research questions and proposing an efficient and accurate solution, we provide the required means and strategies to further develop a concern-based NFR classification tool which can be used by stakeholders.

The general process of our classification task is shown in Figure 8. In the next sections, we will explain the details for each of these steps.

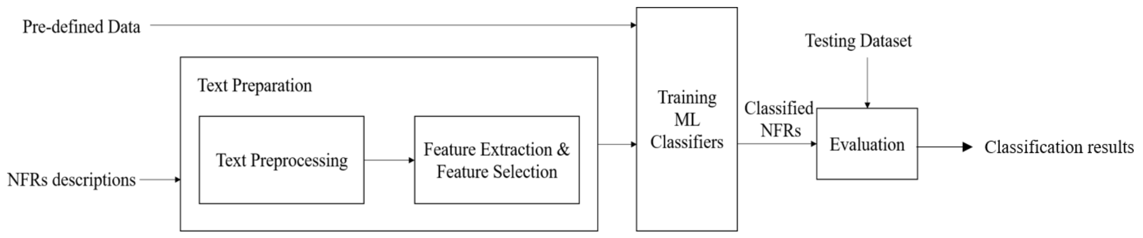


Figure 9: Process of applying ML algorithms to classification of F/NFR and classification of NFR subtypes

4.1 Pre-defined Data

As shown in Chapter 3 (Related Work), most of the researchers in previous studies used the NFR PROMISE dataset for the identification and classification of non-functional requirements from the text. That is why we also used the NFR PROMISE dataset. Besides, we used another dataset called PURE and combined the two datasets to attain a larger dataset to train our classifiers. To validate our work, we also used an industry dataset extracted from a collaborating company's requirement specification document. The rest of this section describes the details of the datasets that we utilized.

The NFR PROMISE dataset

The NFR PROMISE dataset comes in an ARFF file (typically used to load Weka programs), which is a comma-separated document. By stripping some extraneous metadata, we converted it to a standard CSV. Each requirement in the dataset includes three attributes, the project ID, the requirement text (wrapped in single quotes), and the requirement classes (indicated in Table 9). As the project ID is not of any use for our classification task, we remove that column from the dataset. The final version of the CSV file includes these columns: "Requirement Descriptions", "F/NF" column which specifies whether that requirement is a functional (F) or a non-functional (NF) requirement, and "Target" which indicates the subtypes of NFRs. Table 10 provides examples for different NFR types of the PROMISE dataset.

Table 10: Overview of NFR PROMISE dataset

Requirement Class	# Requirements	Percent
Functional	255	4.80%
Availability (A)	21	3.36%
Fault Tolerance (FT)	10	1.60%
Legal (L)	13	2.08%
Look and Feel (LF)	38	6.08%
Maintainability (MN)	17	2.72%
Operational (O)	62	9.92%
Performance (PE)	54	8.64%
Portability (PO)	1	0.16%
Scalability (SC)	21	3.36%
Security (SC)	66	10.56%
Usability (US)	67	10.72%
Total	**625**	**100%**

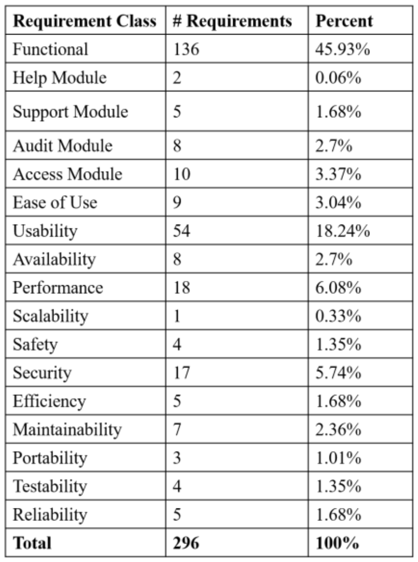
Table 11: Examples of requirements of different types from NFR PROMISE dataset

Type	Requirement Text
Availability	"Aside from server failure the software product shall achieve 99.99% up time."
Legal	"The product shall comply with the estimates laws relating to recycled parts usage."
Look and Feel	"The look and feel of the system shall conform to the user interface standards of the smart device."
Maintainability	"Application updates shall occur between 3AM and 6 AM CST on Wednesday morning during the middle of the NFL season."
Operational	"The product shall run on the existing hardware for all environments."
Performance	"The search for the preferred repair facility shall take no longer than 8 seconds. The preferred repair facility is returned within 8 seconds."
Scalability	"System shall be able to handle all of the user requests/usage during business hours."
Security	"Only valid data shall be entered into the system. No invalid data shall be entered into the system."
Usability	"Users shall feel satisfied using the product. 85% of all users will be satisfied with the product."
Functional	"A non-clinical class shall specify the course name lecture room requirements and instructor needs."

PURE: Dataset of Public Requirements Documents

Another dataset used for this research is the PURE (Public Requirements) dataset. As mentioned in the background chapter, this dataset refers to a set of 79 publicly available natural language requirement documents collected from the web [46]. As we needed a labeled dataset for our machine learning classification task, we chose the documents which contained all of the corresponding requirement categorized. Only 6 of the documents matched this requirement. We extracted all of the requirement statements and added them into a CSV file with three columns. These three columns are the same as the ones we used in the PROMISE dataset including “Requirement Descriptions”, “F/NF”, and “Target”. The requirement types of the PURE dataset are shown in Table 11, and examples of each type of NFRs are provided in Table 12.

Table 12: Overview of the PURE dataset



Requirement Class	# Requirements	Percent
Functional	136	45.93%
Help Module	2	0.06%
Support Module	5	1.68%
Audit Module	8	2.7%
Access Module	10	3.37%
Ease of Use	9	3.04%
Usability	54	18.24%
Availability	8	2.7%
Performance	18	6.08%
Scalability	1	0.33%
Safety	4	1.35%
Security	17	5.74%
Efficiency	5	1.68%
Maintainability	7	2.36%
Portability	3	1.01%
Testability	4	1.35%
Reliability	5	1.68%
Total	**296**	**100%**

Our training dataset is a combination of the PROMISE dataset and the PURE dataset with a total number of 921 requirements (391 functional and 530 non-functional requirements).

Case Study Industry Dataset

We have also used a separate dataset extracted from a company’s Subsystem Requirements Specification (SSRS) document. The conversion process of this dataset was the same as the PURE dataset. We extracted the requirement sentences and placed them in the Requirement Description column. In the second column, F/NF, we used two values “F”, or “NF” to indicat whether the requirement is functional or non-fucntional, and in column 3, Target, we added their NFR type (if they are functional, the value of this column is

“Functional”). The requirement types of the case study dataset are shown in Table 13 and the examples of this dataset are provided in Table 14.

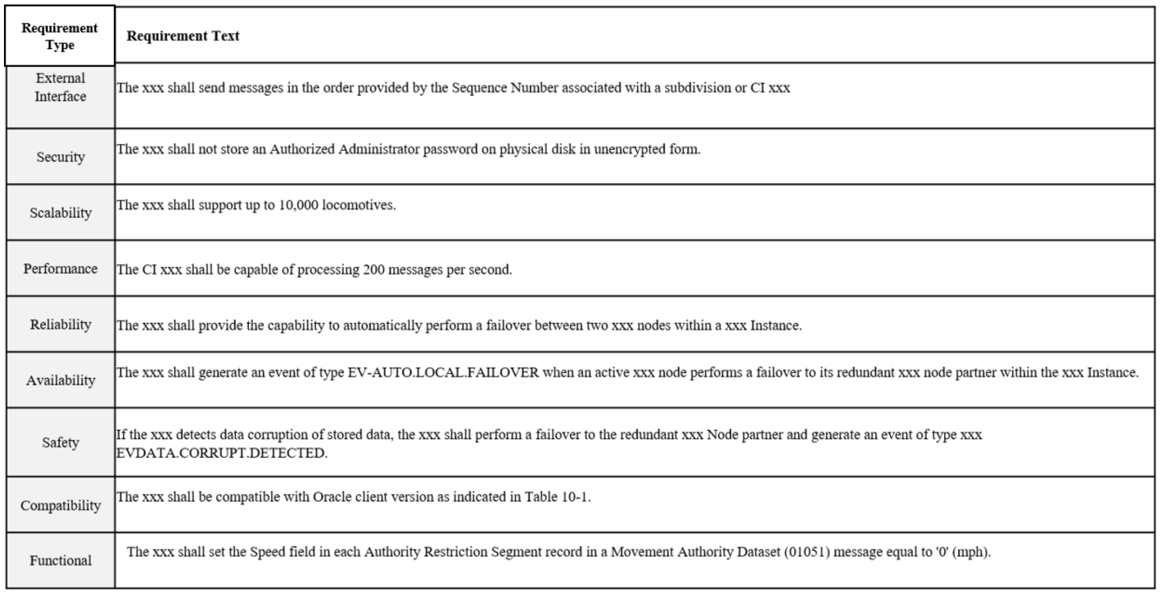
Table 13: Examples of requirements of different types from PURE dataset

Requirement Type	Requirement Text
Help Module	“The help should be accessible to the users both in the offline and online mode.”
Support Module	“The support solution should be accessible to the users both from within the application and also outside the application through a browser interface.”
Audit Module	“The System must maintain the audit trail for as long as required, which will be at least for the life of the case to which it refers.”
Access Module	“If a user performs a quick or advanced search, the System must never include in the search result list any record which the user does not have the right to access.”
Ease of Use	“The System must employ a single set of user interface rules, or a small number of sets to provide a familiar and common look and feel for the application.”
Usability	“Links that open new browser windows or pop-up windows should be clearly marked.”
Availability	“Unplanned downtime for the System must not exceed 1 hour per rolling three month period.”
Performance	“Upon the USB being plugged in the system shall be able to be deployed and operational in less than 1 minute.”
Scalability	“The System should be saleable and must not have any features which would preclude use in small or large police stations, with varying numbers of cases handled.”
Safety	“The system will do periodic backups through a live internet connection.”
Security	“The system shall detect consecutive failed login attempts.”
Efficiency	“The system shall automatically compress image files that are too large in size.
Maintainability	“The system shall utilize interchangeable plugins.”
Portability	“The system shall be easy to migrate or backed up via another USB drive.”
Testability	“The system should be able to create test environment of web order system.”
Reliability	“The Digital Home System must be highly reliable with no more than 1 failure per 10,000 hours of operation.”
Functional	“The THEMAS system shall control the heating and cooling units that are designated as part of the THEMAS system.”

Table 14: Overview of the Case Study dataset

Requirement Type	Total
External Interface	169
Security	38
Scalability	15
Performance	12
Reliability	9
Availability	9
Safety	5
Compatibility	5
Total NFRs	262
Functional	1559
Total	1821

Table 15: Examples of requirements of different types from Case Study dataset[4]



Requirement Type	Requirement Text
External Interface	The xxx shall send messages in the order provided by the Sequence Number associated with a subdivision or CI xxx
Security	The xxx shall not store an Authorized Administrator password on physical disk in unencrypted form.
Scalability	The xxx shall support up to 10,000 locomotives.
Performance	The CI xxx shall be capable of processing 200 messages per second.
Reliability	The xxx shall provide the capability to automatically perform a failover between two xxx nodes within a xxx Instance.
Availability	The xxx shall generate an event of type EV-AUTO.LOCAL.FAILOVER when an active xxx node performs a failover to its redundant xxx node partner within the xxx Instance.
Safety	If the xxx detects data corruption of stored data, the xxx shall perform a failover to the redundant xxx Node partner and generate an event of type xxx EVDATA.CORRUPT.DETECTED.
Compatibility	The xxx shall be compatible with Oracle client version as indicated in Table 10-1.
Functional	The xxx shall set the Speed field in each Authority Restriction Segment record in a Movement Authority Dataset (01051) message equal to '0' (mph).

4.2 Text Preprocessing

In this step, we first parse the csv file into a DataFrame[5] for the convenience of processing with Python, with each row in the DataFrame representing a single requirement sample. In

[4] Because of confidentiality reasons, instead of actual name of the system we use xxx.

[5] Pandas.DataFrame

this step, we use the Natural Language Toolkit, NLTK, to preprocess the data. Each of the requirement samples undergoes the following preprocessing procedures.

- First, we use case fold method to map everything to lowercase to remove all case distinctions present in the requirement statements.
- Then, we clean the requirements text string by tokenizing it into words and stripping it of punctuation. We use RegexpTokizer of the NLTK library for this purpose. After the tokenization activity, the text is transformed into a series of tokens where capitals, punctuation, and brackets are removed.
- In this step, we want to remove stop words. The stop words are the terms that do not contribute to the semantic of the text such as a, and, the, etc. As there is no single universal list of stop words used by all-natural language preprocessing tools, we remove all the tokens with less than three letters.
- After the tokenization and stop word removal activities, each token is classified according to the part of speech, i.e., the role it plays in a given text. After the POS tagging activity, each token is hence associated with a tag indicating that such a token is a verb, noun, adjective, etc. We apply the POS_tag method of the NLTK library to find all the POS-tags of requirement statements.
- After the POS tagging activity, the tokens are converted to their morphological stems. The advantage provided by stemming the words is mitigating the influence of morphological variants on a similarity measure. We use PorterStemmer to apply stemming. In this step, we also use WordNetLemmatizer to apply lemmatization.

4.3 Feature Extraction

After cleaning the requirements and applying the above-mentioned preprocessing techniques, in this step, we apply techniques to extract textual features to train the classifiers. The original features we use to convert each requirement statement to a vector is the original bag of words suggested by Slankas et al. in [1]. Bag of words employs all unique terms in the requirement sentences of the training dataset as textual features and uses term frequency as the weight of textual features. In addition to BOW features, TF-IDF scores associated with each term present in a given requirement is used in this classification framework. TF-IDF combines term frequency with inverse document frequency to get the weight of textual features, influenced by the frequency of the term in the requirement sentences of the dataset. One of the issues about BoW is that discarding word order ignores the context and in turn meaning of the words in documents (semantics). To add the semantic-related feature, we adopt a feature list proposed by Hussain et al. [48] which consists of syntactic features and specific keywords based on part-of-speech groups. In this work [48], the researchers state that considering that NFRs describe quality aspects of the system, it can be realized that some categories of words like adjectives and adverbs are likely used frequently in NFR sentences. Using their syntactic features based on POS-tagging, they could achieve an accuracy of above 95% in classifying FRs and NFRs, however, they did not provide a classification of NFR types. Following similar characteristics of NFRs, as described in [3] and adopting a syntactic feature list proposed by [2], we are motivated to add the following syntactic features:

- Number of Adjectives
- Number of Adverbs
- Number of Cardinals
- Number of Verbs
- Number of nouns
- Number of Modals

Additionally, as previous research described in [4,5] has shown, NFR statements are mostly identifiable by the use of specific keywords that belong to different part-of-speech categories. We adopted the keyword features proposed by [2] and considered the following part-of-speech groups:

- Adjective Keywords (coded as: JJ_kw)
- Adverb Keywords (coded as: RB_kw)
- Modal keywords (coded as: MD_kw)
- Determiner keywords (Coded as DT_kw)
- Verb keywords (coded as: VB_kw)
- Preposition keywords (coded as: IN_kw)
- Common Noun-keywords that appeared in singular form (coded as: *NN_kw*)
- Common Noun-keywords that appeared in plural form (coded as: *NNS_kw*)

At this point three sets of feature sets are prepared. For each of them we have Xtrain which is the data frame with requirement features and ytrain which is the data frame that includes target values. Depending on our experiment, ytrain will contain different target values. If we are classifying functional and non-functional requirements, the value of ytrain will be "F" or "NF"; if we are doing the classification of different NFR types, ytrain will include the type of the NFR such as performance, security, etc.

For the classification of functional requirement and non-functional requirement, ytrain includes 1 (for functional) or 0 (for NFR).

For multiclass classification of NFRs, ytrain contains 1 for usability, 2 for security, 3 for performance, and 4 for operational types.

For binary class classification of four considered NFRs, ytrain includes 1 if the requirement is of the type we are doing the classification for, and 0 if it is not. For example, for the binary classification of usability, if the non-functional requirement is of type usability, then ytrain value for that requirement is 1, otherwise it is 0. The distribution of training dataset is shown in Figure 9 (FRs and NFRs) and Figure 10 (subtypes of NFRs). As we can see, FRs and NFRs are almost fairly balanced, but between NFR types, the difference of the available requirement instances in each of the NFR types is considerable. For that reason, we only chose the top four NFRs (usability, security, performance, and operational) with the greatest number of requirement instances, for our NFR classification task.

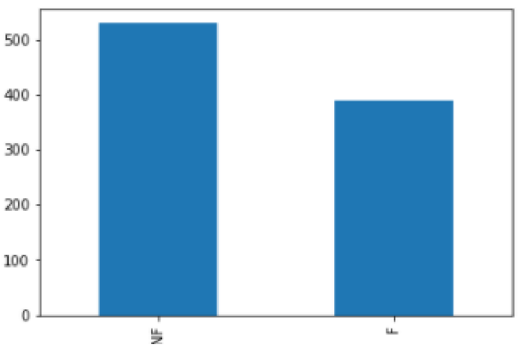


Figure 10: Distribution of functional and non-functional requirements in the training dataset

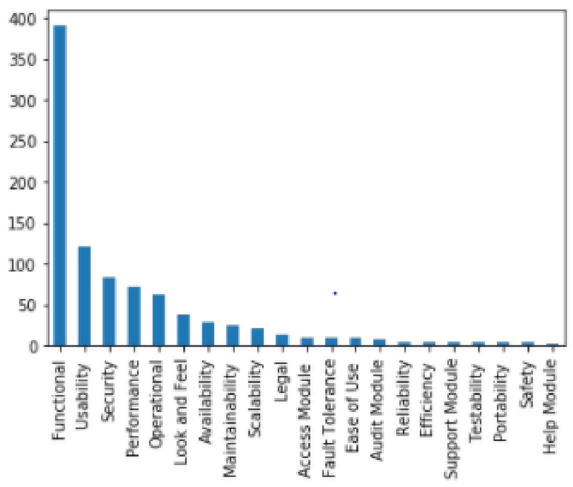


Figure 11: Distribution of sub-types of non-functional requirements in the training dataset

4.4 Feature Selection

To assess the feature importance, we also build a scoring classifier as an ensemble of tree classifiers using our training set: Adaptive Boost, Extra Tree, Gradient Boosting, and Random Forest. For each feature, we average the sum over all feature importance scores. We rank the features according to their importance and select the top 15. For each of the four NFR types including usability, security, performance, and operational, using the described scoring classifier, we will indicate the top 15 informative features in the next chapter.

4.5 Training Classifiers

We use 10 machine learning algorithms to train our classifiers including Support Vector Machine (SVM), SGD SVM, Linear Regression, Decision Tree, Bagging Tree, Ensemble Extra Tree, Random Forest, Gaussian Naïve Bayes, Multi Nominal Naïve Bayes, and Bernoulli Naïve Bayes. We chose these classifiers because as it is shown in [6] and other related work explained in chapter 3, most of these techniques have been successfully deployed with text classification.

We build these classifiers using the machine learning library Scikit[6]. The goal of our investigation is to determine the best combination of the three feature extraction techniques and 10 supervised learning techniques for the identification and classification of non-functional requirements, complemented by execution time. The pseudo code of our procedure is shown in Figure 11.

We execute the procedure using stratified[7] 10-fold cross validation in order to obtain average scores to evaluate prediction performance and to achieve a more accurate estimate of a real world's performance. Since the collection used for experiments has an unbalanced distribution of examples, stratification is used to ensure that each fold contains roughly the same proportion of examples in each class as in the original collection. In each experiment, we divide the whole dataset into 10 subsets. The 9 out of 10 subsets are used for training and the remaining subset is used for testing. We repeat the experiments 10 times and the evaluation metrics including precision and recall are calculated for each test and each training class by the corresponding sklearn function and the results are averaged for each classifier. It should be noted that for our second experiment, doing a multi-class classification for non-functional requirements, we are using macro-average scores for both recall and precision metrics by using "average=macro" in our sklearn setting.

As mentioned before, we will do two different classification tasks on NFRs. On our first experiment, we will train a multi-class classifier and in the other, we will train binary classifiers for each of the NFR types we are doing a classification.

Binary Classifiers

As previously explained, classification is a predictive modeling problem that involves assigning a class label to an example. Binary classification are those tasks where examples are assigned exactly one of two classes. Multi-class classification is those tasks where examples are assigned exactly one of more than two classes.

[6] http://scikit-learn.org/, accessed on March 2017

[7] sklearn.model selection.StratifiedKFold

- Binary Classification: Classification tasks with two classes.
- Multi-class Classification: Classification tasks with more than two classes.

Some algorithms are designed for binary classification problems. Examples include:

- Logistic Regression
- Perceptron
- Support Vector Machine

As such, they cannot directly be used for multi-class classification tasks. Instead, we need to use methods to split a multi-class classification problem into multiple binary classification datasets and train a binary classification model each. Two methods to this includes:

- One-vs-Rest
- One-vs-One

One-vs-Rest for Multi-Class Classification

One-vs-Rest is a method for using binary classification algorithms for multi-class classification. It involves splitting the multi-class dataset into multiple binary classification problems. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident. A possible downside of this approach is that it requires one model to be created for each class. For example, for our NFR classification task (usability, security, performance, and operation), four classes will require four models. This could be an issue for large datasets or very large number of classes.

One-vs-One for Multi-Class Classification

One-vs-One is another method for using binary classification algorithms for multi-class classification. Like one-vs-rest, one-vs-one splits a multi-class classification dataset into binary classification problems. Unlike one-vs-rest that splits it into one binary dataset for each class, the one-vs-one approach splits the dataset into one dataset for each class versus every other class. For example, for our NFR classification task with four classes, usability, security, performance and operation, we will have six binary classification datasets as follows:

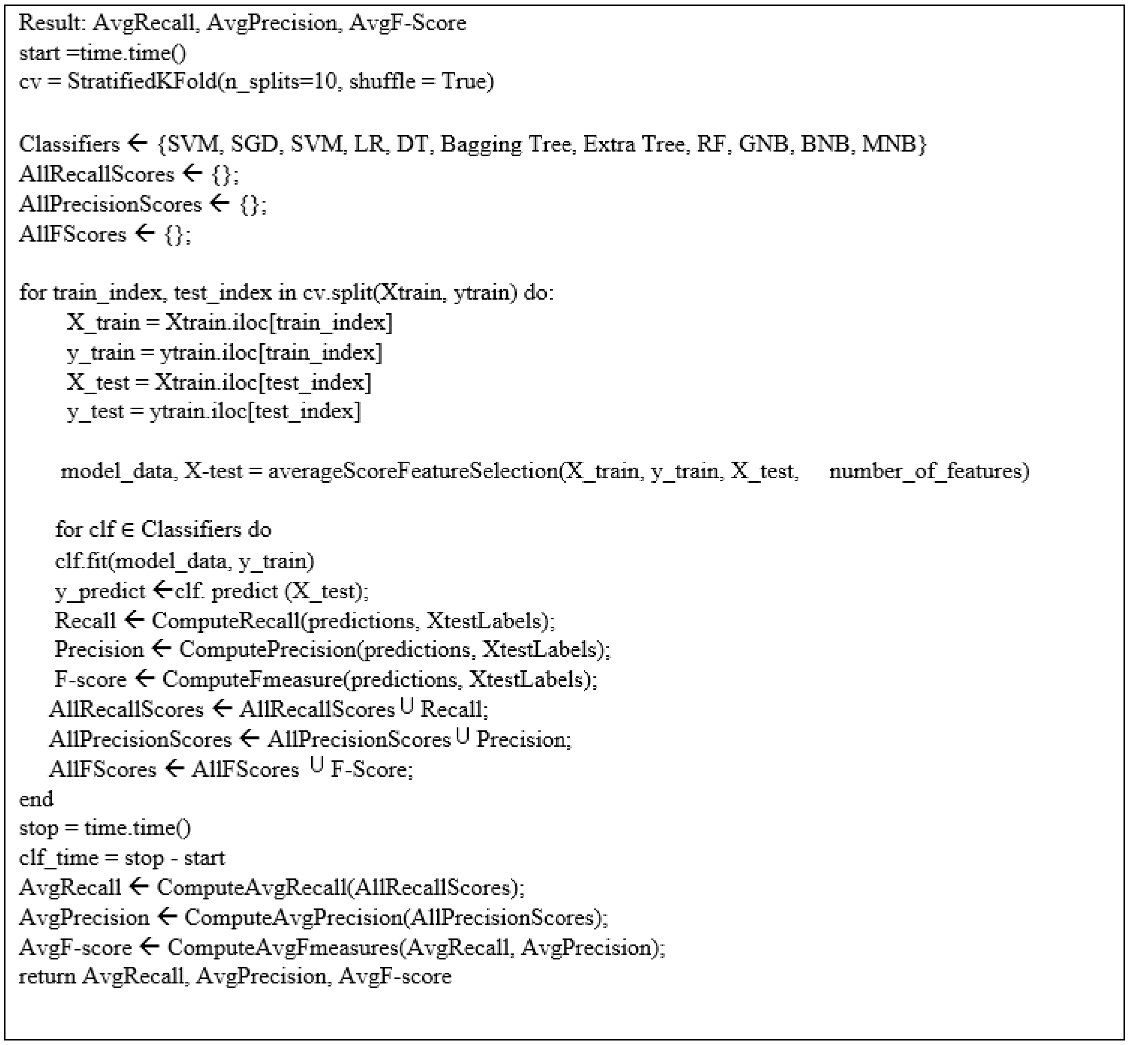
- usability vs. security
- usability vs. performance
- usability vs. operation
- security vs. performance
- security vs. operation
- performance vs. operation

Each binary classification model may predict one class label and the model with the most predictions or votes is predicted by the one-vs-one strategy. This approach is suggested for support vector machines and related kernel-based algorithms. This is because the performance of kernel methods does not scale in proportion to the size of the training dataset and using subsets of the training data may counter this effect [58].

The support vector machine implementation in the scikit-learn is provided by the SVC class and supports the one-vs-one method for multi-class classification problems. We used this option by setting the “decision_function_shape” argument to “ovo”.

4.6 Validation using Case Study Dataset

After we complete the execution and achieve the results which we are going to present and discuss in the next chapter, we repeat the procedure with a whole different dataset we mentioned above. it should be noted that unlike the NFR PROMISE dataset, our case-study dataset belongs to an industry project which can give us a sense of real-world project requirements. That said, using a case study dataset is our next level of validation. It should be mentioned that there is more space to validating our techniques, especially using actual requirements dataset. In the next chapter, the results of using our proposed techniques with the case study dataset will be presented and discussed as well.



```
Result: AvgRecall, AvgPrecision, AvgF-Score
start =time.time()
cv = StratifiedKFold(n_splits=10, shuffle = True)

Classifiers ← {SVM, SGD, SVM, LR, DT, Bagging Tree, Extra Tree, RF, GNB, BNB, MNB}
AllRecallScores ← {};
AllPrecisionScores ← {};
AllFScores ← {};

for train_index, test_index in cv.split(Xtrain, ytrain) do:
    X_train = Xtrain.iloc[train_index]
    y_train = ytrain.iloc[train_index]
    X_test = Xtrain.iloc[test_index]
    y_test = ytrain.iloc[test_index]

    model_data, X-test = averageScoreFeatureSelection(X_train, y_train, X_test,     number_of_features)

   for clf ∈ Classifiers do
   clf.fit(model_data, y_train)
   y_predict ←clf. predict (X_test);
   Recall ← ComputeRecall(predictions, XtestLabels);
   Precision ← ComputePrecision(predictions, XtestLabels);
   F-score ← ComputeFmeasure(predictions, XtestLabels);
  AllRecallScores ← AllRecallScores ∪ Recall;
  AllPrecisionScores ← AllPrecisionScores ∪ Precision;
  AllFScores ← AllFScores ∪ F-Score;
end
stop = time.time()
clf_time = stop - start
AvgRecall ← ComputeAvgRecall(AllRecallScores);
AvgPrecision ← ComputeAvgPrecision(AllPrecisionScores);
AvgF-score ← ComputeAvgFmeasures(AvgRecall, AvgPrecision);
return AvgRecall, AvgPrecision, AvgF-score
```

Figure 12: Pseudo Code of the procedure for NFR classification

Chapter 5

5 Research Results and Discussion

Throughout this chapter, we will present the results of our investigation on the training dataset which is composed of two datasets: the NFR PROMISE dataset and the PURE dataset. We will also present the results of our investigation using the case-study industry dataset. In this chapter, we will look into the performance of suggested models for our classification task and we will compare the results under different settings.

In order to answer to the research questions RQ1 and RQ2 (see Chapter 4), we will perform three different classification tasks. First, we will perform a binary classification to classify requirements into FRs and NFRs. In the second experiment, we will perform a multi-class classification to classify four types of NFRs: usability, security, performance, and operation. In the third classification task, we will perform binary classification for each of the above-mentioned four NFR types to investigate whether performing binary classification can possibly improve our results or not. To answer to RQ1, we will choose the best results based on the evaluation metric including recall, precision, and execution time. It should be noted that precision and recall are often used together [5], and there is a trade-off between them. Precision ensures that all retrieved requirements are truly relevant, whereas recall focuses on retrieving all relevant requirements. It is arguable which value is more important to measure the classification results [5, 47] focused on achieving a high recall as it was believed that rejecting irrelevant requirements manually from a set of retrieved requirements was easier than reading an entire document looking for missing requirements. On the other hand, [12] argued that precision was more important if recall was acceptable for automatic classification to avoid large amounts of irrelevant NFRs from being misclassified as relevant (false positive). We will compare our results with the related works at the end of this section. For this research project, we will make our comparisons based on both precision and recall; however, at the outset, recall values will be the base of our comparisons as recall focuses on retrieving all related requirements, and a tool with high recall (as well as acceptable precision) can ensure stakeholders of the completeness of their concern-based NFR sets.

To answer RQ2, we will propose a combination of feature extraction techniques and supervised machine learning algorithms that produced the best results in RQ1. Finally, in order to assess the feature importance, we will use binary classifiers for each of the four NFR types (usability, security, performance, operation) and introduce the top 15 informative keywords for each of the four mentioned NFRs. As it was mentioned in Chapter 4, Section 4.3, our scoring classifier is an ensemble of tree classifiers which consists of Adaptive Boost, Extra Tree, Gradient Boost, and Random Forest. For each feature, we average the sum over all feature importance scores. Then we rank the features according to their importance and select the top 15.

After we choose the best combination of feature extraction techniques and SML algorithms, we will test the proposed solution with a separate industry dataset as well.

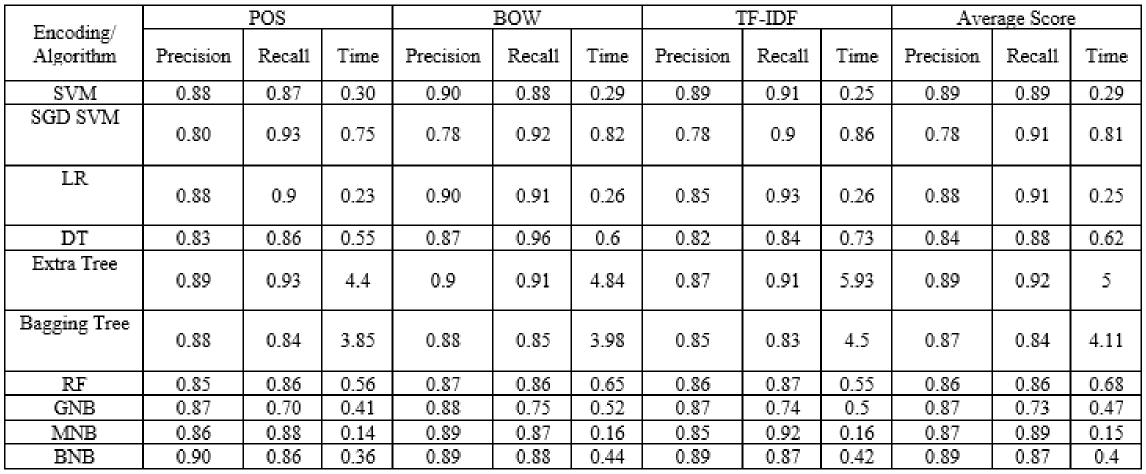
5.1 RQ1: Classification of FRs and NFRs

In this classification task, we are using our training dataset which consists of 921 requirements (530 NFRs and 390 FRs).

We perform the classification of FRs and NFRs with three different feature sets. Syntactic POS-tagging based features, BoW, and TF-IDF. We try 10 classifiers including SVM, SGD-SVM, LR, Decision Tree, Ensemble Extra Tree, Bagging Tree, Random Forest, Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes with each of these feature sets. The results are shown in Table 15.

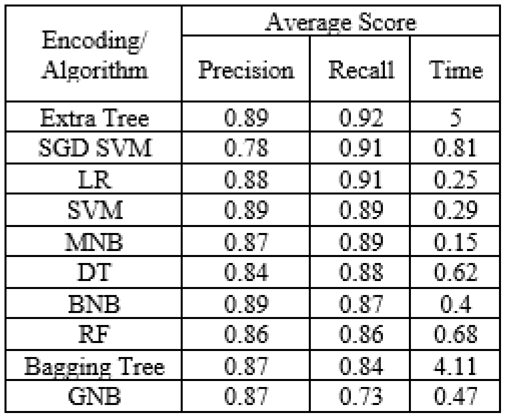
Generally, all applied classifiers except for GNB, achieved a minimum recall of 0.83 as is shown in Table 16. Comparing precision values, all of the classifiers achieved minimum precision of 0.82 except for SGD SVM, as is shown in Table 17. This demonstrates that, without considering the specific feature extraction techniques, almost all classifiers performed well. Table 18 shows the classifiers sorted by their execution time. As indicated, MNB, LR, and SVM achieved the least execution time.

Table 16: Results of the classification of FRs and NFRs



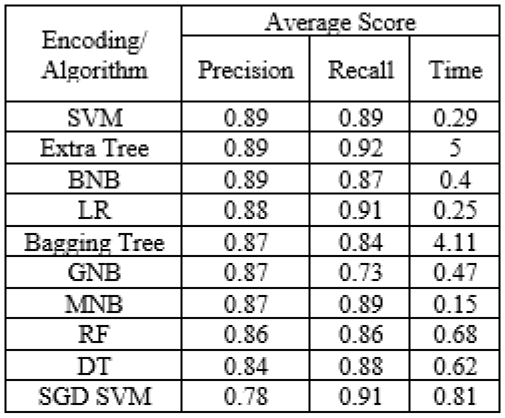
Encoding/ Algorithm	POS			BOW			TF-IDF			Average Score		
	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time
SVM	0.88	0.87	0.30	0.90	0.88	0.29	0.89	0.91	0.25	0.89	0.89	0.29
SGD SVM	0.80	0.93	0.75	0.78	0.92	0.82	0.78	0.9	0.86	0.78	0.91	0.81
LR	0.88	0.9	0.23	0.90	0.91	0.26	0.85	0.93	0.26	0.88	0.91	0.25
DT	0.83	0.86	0.55	0.87	0.96	0.6	0.82	0.84	0.73	0.84	0.88	0.62
Extra Tree	0.89	0.93	4.4	0.9	0.91	4.84	0.87	0.91	5.93	0.89	0.92	5
Bagging Tree	0.88	0.84	3.85	0.88	0.85	3.98	0.85	0.83	4.5	0.87	0.84	4.11
RF	0.85	0.86	0.56	0.87	0.86	0.65	0.86	0.87	0.55	0.86	0.86	0.68
GNB	0.87	0.70	0.41	0.88	0.75	0.52	0.87	0.74	0.5	0.87	0.73	0.47
MNB	0.86	0.88	0.14	0.89	0.87	0.16	0.85	0.92	0.16	0.87	0.89	0.15
BNB	0.90	0.86	0.36	0.89	0.88	0.44	0.89	0.87	0.42	0.89	0.87	0.4

Table 17: FR-NFR Classification-Sorted Recall Results



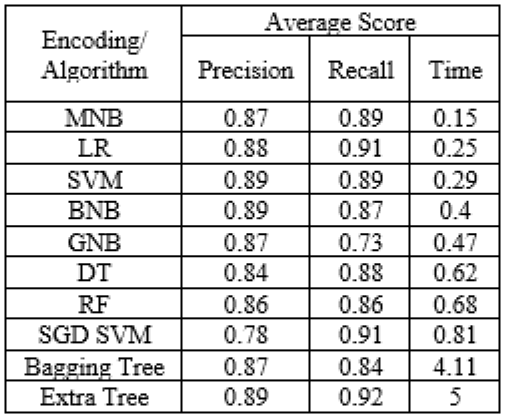
Encoding/ Algorithm	Average Score		
	Precision	Recall	Time
Extra Tree	0.89	0.92	5
SGD SVM	0.78	0.91	0.81
LR	0.88	0.91	0.25
SVM	0.89	0.89	0.29
MNB	0.87	0.89	0.15
DT	0.84	0.88	0.62
BNB	0.89	0.87	0.4
RF	0.86	0.86	0.68
Bagging Tree	0.87	0.84	4.11
GNB	0.87	0.73	0.47

Table 18: FR-NFR classification-Sorted Precision Results



Encoding/ Algorithm	Average Score		
	Precision	Recall	Time
SVM	0.89	0.89	0.29
Extra Tree	0.89	0.92	5
BNB	0.89	0.87	0.4
LR	0.88	0.91	0.25
Bagging Tree	0.87	0.84	4.11
GNB	0.87	0.73	0.47
MNB	0.87	0.89	0.15
RF	0.86	0.86	0.68
DT	0.84	0.88	0.62
SGD SVM	0.78	0.91	0.81

Table 19: FR-NFR classification-Sorted Execution Time Results



Encoding/ Algorithm	Average Score		
	Precision	Recall	Time
MNB	0.87	0.89	0.15
LR	0.88	0.91	0.25
SVM	0.89	0.89	0.29
BNB	0.89	0.87	0.4
GNB	0.87	0.73	0.47
DT	0.84	0.88	0.62
RF	0.86	0.86	0.68
SGD SVM	0.78	0.91	0.81
Bagging Tree	0.87	0.84	4.11
Extra Tree	0.89	0.92	5

Considering the average results, the highest recall values are achieved by Extra Tree, SGD SVM, and LR which attained 0.92, 0.91, and 0.91, respectively. Considering precision values, SGD SVM achieved the least precision (0.78) among these three, and Extra Tree and LR with precision values of 0.89 and 0.88 performed better. Considering execution time, Extra Tree took 5s, LR took 0.25s, and SGD SVM took 0.81s. Taking all three-evaluation metrics into consideration, LR outperformed other classifiers in this classification task.

It should also be noted that SVM with average recall 0.89, precision 0.89, and execution time 0.29, and MNB classifier with average recall 0.89, average precision 0.87, and execution time 0.15s performed good as well.

Considering all combinations and comparing their performance, decision Tree with BoW achieved the highest recall value (0.96) with execution time of 0.6 seconds. Extra Tree with POS-tagging achieved recall of 0.93 but its execution time was the worst among all of the combinations (4.4 seconds). Considering all of the three-evaluation metrics, three of the classifiers outperformed others in this set of classifiers, SVM, LR, and MNB. Although they performed well with all three feature extraction techniques, the results show that they achieved higher values when combined with TF-IDF feature extraction technique.

5.2 Classification of NFRs

In this classification task, we are using our training dataset which contains 530 NFRs overall. The number of instances for some types are low as shown in Figure 16. Thus, we chose the top four NFR types which contain the greatest number of requirement instances, usability with 121, security with 82, performance with 70, and operational with 62 requirements. Same with the previous classification task, we perform the classification of NFRs with three different feature sets. Syntactic POS-tagging based features, BoW, and TF-IDF. We try 10 classifiers including SVM, SGD-SVM, LR, Decision Tree, Ensemble Extra Tree, Bagging Tree, Random Forest, Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes with each of these feature sets. The results are shown in Table 19.

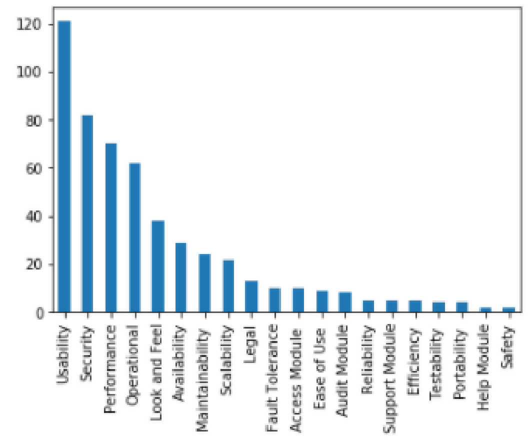
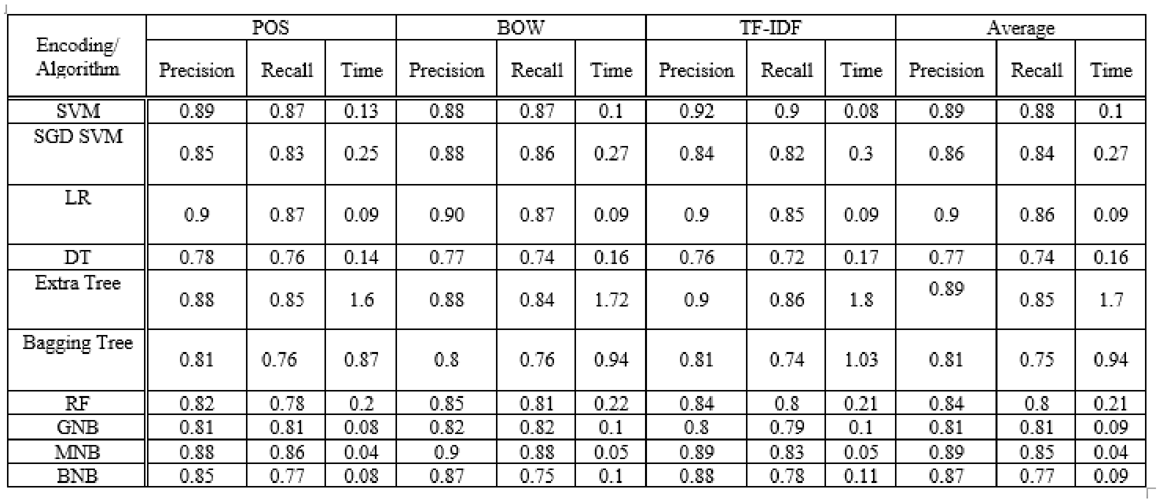


Figure 12: Distribution of NFRs in the training dataset

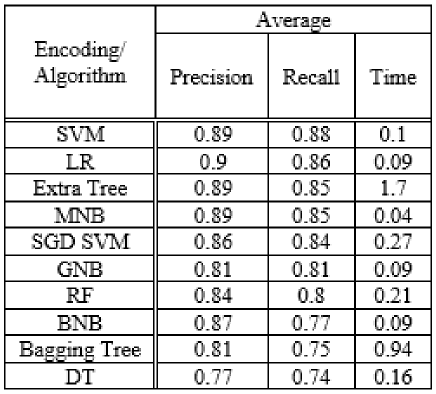
Table 20: Results of the classification of four NFR types: usability, security, performance, and operational



Encoding/ Algorithm	POS			BOW			TF-IDF			Average		
	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time
SVM	0.89	0.87	0.13	0.88	0.87	0.1	0.92	0.9	0.08	0.89	0.88	0.1
SGD SVM	0.85	0.83	0.25	0.88	0.86	0.27	0.84	0.82	0.3	0.86	0.84	0.27
LR	0.9	0.87	0.09	0.90	0.87	0.09	0.9	0.85	0.09	0.9	0.86	0.09
DT	0.78	0.76	0.14	0.77	0.74	0.16	0.76	0.72	0.17	0.77	0.74	0.16
Extra Tree	0.88	0.85	1.6	0.88	0.84	1.72	0.9	0.86	1.8	0.89	0.85	1.7
Bagging Tree	0.81	0.76	0.87	0.8	0.76	0.94	0.81	0.74	1.03	0.81	0.75	0.94
RF	0.82	0.78	0.2	0.85	0.81	0.22	0.84	0.8	0.21	0.84	0.8	0.21
GNB	0.81	0.81	0.08	0.82	0.82	0.1	0.8	0.79	0.1	0.81	0.81	0.09
MNB	0.88	0.86	0.04	0.9	0.88	0.05	0.89	0.83	0.05	0.89	0.85	0.04
BNB	0.85	0.77	0.08	0.87	0.75	0.1	0.88	0.78	0.11	0.87	0.77	0.09

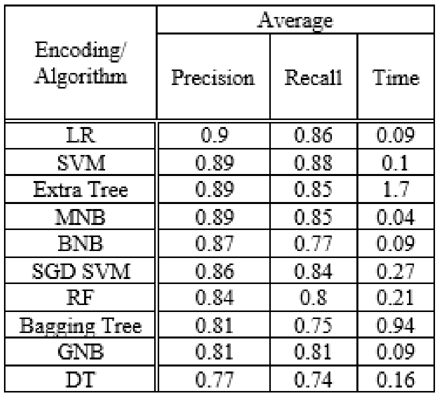
Generally, all applied classifiers, got recall values above 0.7. The highest recall value was achieved by SVM with Linear kernel and the worst recall was achieved by Decision Tree as indicated in Table 20. Regarding precision values, all classifiers achieved precision values above 0.8, except for DT as it is shown in Table 21. Also, classification results sorted by the classifiers' execution time is shown in Table 22. As we see, MNB is the fastest with 0.04 s and Extra Tree classifiers achieved the worst execution time with 1.7s.

Table 21: NFR classification-Sorted Recall Results



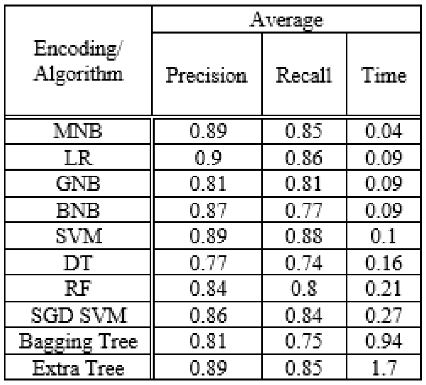
Encoding/ Algorithm	Average		
	Precision	Recall	Time
SVM	0.89	0.88	0.1
LR	0.9	0.86	0.09
Extra Tree	0.89	0.85	1.7
MNB	0.89	0.85	0.04
SGD SVM	0.86	0.84	0.27
GNB	0.81	0.81	0.09
RF	0.84	0.8	0.21
BNB	0.87	0.77	0.09
Bagging Tree	0.81	0.75	0.94
DT	0.77	0.74	0.16

Table 22: NFR classification-Sorted Precision Results



Encoding/ Algorithm	Average		
	Precision	Recall	Time
LR	0.9	0.86	0.09
SVM	0.89	0.88	0.1
Extra Tree	0.89	0.85	1.7
MNB	0.89	0.85	0.04
BNB	0.87	0.77	0.09
SGD SVM	0.86	0.84	0.27
RF	0.84	0.8	0.21
Bagging Tree	0.81	0.75	0.94
GNB	0.81	0.81	0.09
DT	0.77	0.74	0.16

23: NFR classification-Sorted Execution Time Results



Encoding/ Algorithm	Average		
	Precision	Recall	Time
MNB	0.89	0.85	0.04
LR	0.9	0.86	0.09
GNB	0.81	0.81	0.09
BNB	0.87	0.77	0.09
SVM	0.89	0.88	0.1
DT	0.77	0.74	0.16
RF	0.84	0.8	0.21
SGD SVM	0.86	0.84	0.27
Bagging Tree	0.81	0.75	0.94
Extra Tree	0.89	0.85	1.7

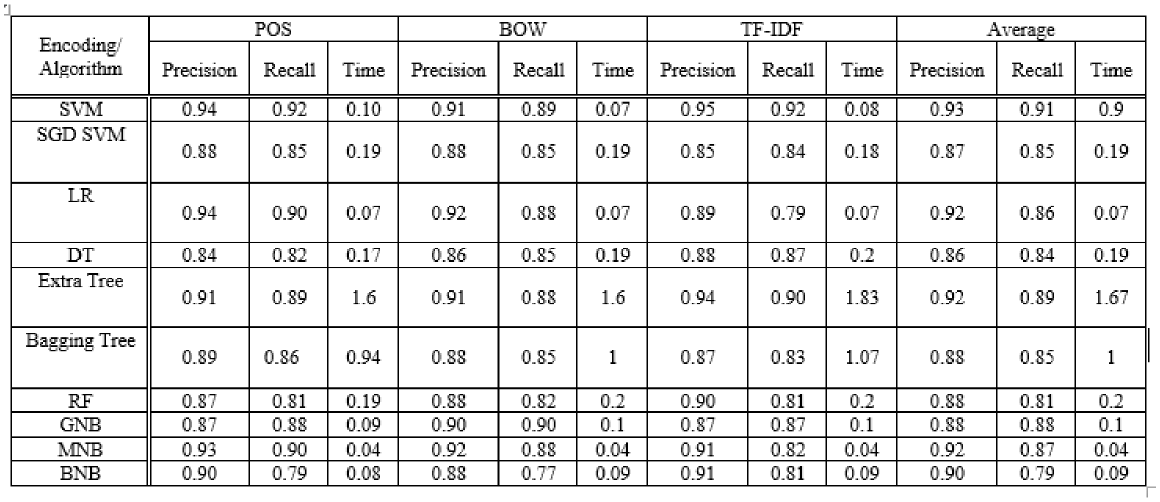
Considering the average results for NFR classification, SVM achieved the highest recall value of 0.88. Between other classifiers, LR with recall of 0.86 and Extra Tree classifier and MNB both with recall value of 0.85, performed good as well. Considering precision values, LR achieved the highest precision value among the mentioned four classifiers with the precision value of 0.9. SVM, Extra Tree and MNB all achieved precision 0.89. Based on the execution time, Extra Tree classifier achieved the worst execution time of 1.7s. MNB is the fastest classifier between all 10 classifiers with execution time of 0.04s. Regarding all evaluation metrics (precision, recall, execution time), LR, SVM with linear kernel, and MNB outperformed other classifiers in this classification task.

Considering all the combinations of feature extraction techniques and SML algorithms, the highest recall value is 0.9, and it is achieved by SVM classifier and TF-IDF feature extraction technique. The precision value of this combination is 0.92, and its execution time is 0.08s. It should be noted that SVM classifiers performed well with all three feature extraction techniques. The other combinations are LR with BoW and POS tagging (recall: 0.87, precision: 0.9, and execution time: 0.09s) and MNB with BoW (recall: 0.88, precision: 0.9, execution time: 0.05s) that performed good as well. Between all of the combinations, DT achieved the worst results with all three feature extraction techniques.

5.3 Binary Classifiers for NFR classification (Usability, Security, Performance, Operational)

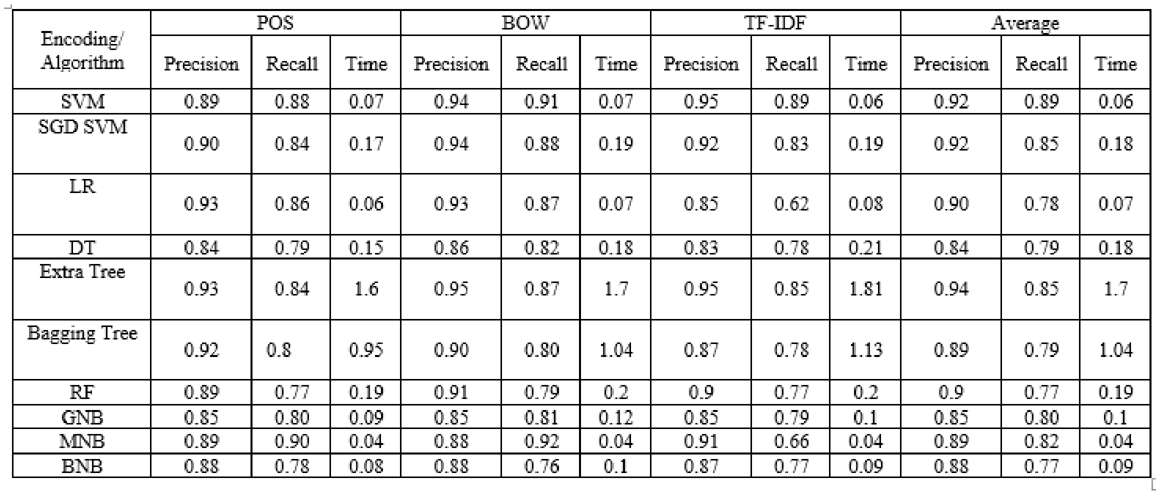
As NFR classification task involves multiple classes, we evaluate two approaches. The first one was shown in section 6.2, in which we trained a single multi-class classifier. In this section, in order to see whether we can possibly improve our results, we take a different evaluation approach and we train four separate binary classifiers, one for each of the four NFR types that we want to classify: usability, security, performance, and operational. The results of these four binary classifiers are shown in Table 24, 25, 26, and 27 accordingly.

Table 24: Results of the binary classification for usability requirements



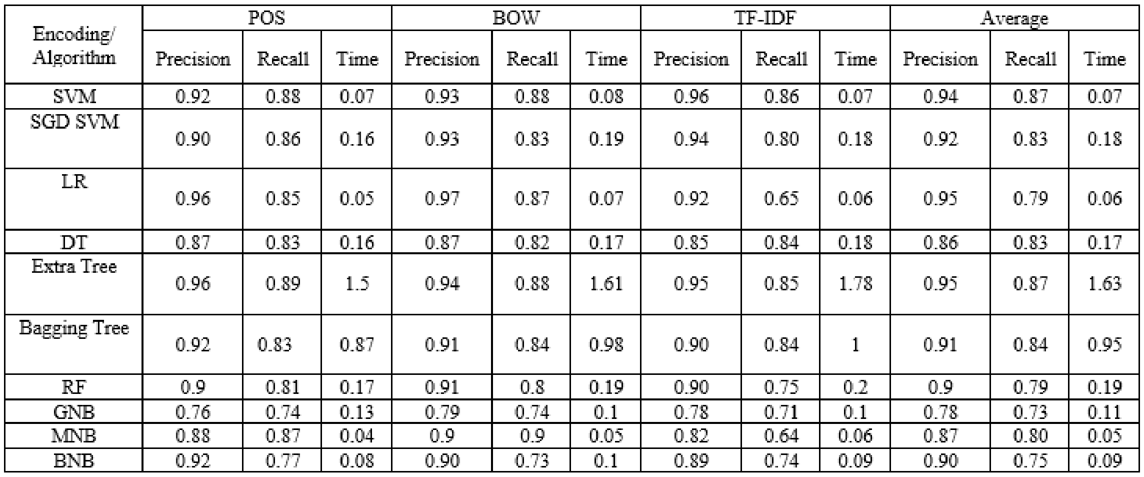
Encoding/ Algorithm	POS			BOW			TF-IDF			Average		
	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time
SVM	0.94	0.92	0.10	0.91	0.89	0.07	0.95	0.92	0.08	0.93	0.91	0.9
SGD SVM	0.88	0.85	0.19	0.88	0.85	0.19	0.85	0.84	0.18	0.87	0.85	0.19
LR	0.94	0.90	0.07	0.92	0.88	0.07	0.89	0.79	0.07	0.92	0.86	0.07
DT	0.84	0.82	0.17	0.86	0.85	0.19	0.88	0.87	0.2	0.86	0.84	0.19
Extra Tree	0.91	0.89	1.6	0.91	0.88	1.6	0.94	0.90	1.83	0.92	0.89	1.67
Bagging Tree	0.89	0.86	0.94	0.88	0.85	1	0.87	0.83	1.07	0.88	0.85	1
RF	0.87	0.81	0.19	0.88	0.82	0.2	0.90	0.81	0.2	0.88	0.81	0.2
GNB	0.87	0.88	0.09	0.90	0.90	0.1	0.87	0.87	0.1	0.88	0.88	0.1
MNB	0.93	0.90	0.04	0.92	0.88	0.04	0.91	0.82	0.04	0.92	0.87	0.04
BNB	0.90	0.79	0.08	0.88	0.77	0.09	0.91	0.81	0.09	0.90	0.79	0.09

Table 25: Results of the binary classification for security requirements



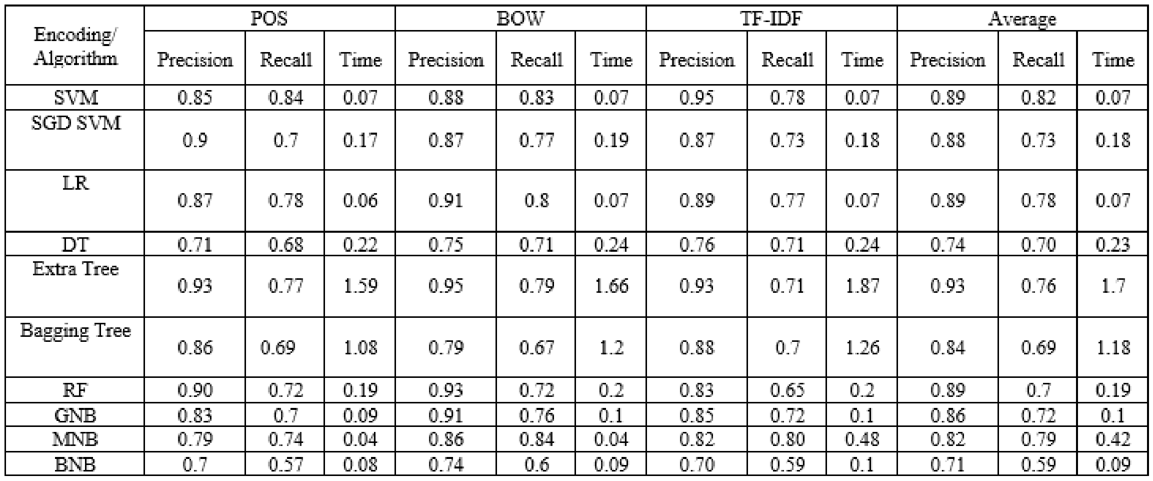
Encoding/ Algorithm	POS			BOW			TF-IDF			Average		
	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time
SVM	0.89	0.88	0.07	0.94	0.91	0.07	0.95	0.89	0.06	0.92	0.89	0.06
SGD SVM	0.90	0.84	0.17	0.94	0.88	0.19	0.92	0.83	0.19	0.92	0.85	0.18
LR	0.93	0.86	0.06	0.93	0.87	0.07	0.85	0.62	0.08	0.90	0.78	0.07
DT	0.84	0.79	0.15	0.86	0.82	0.18	0.83	0.78	0.21	0.84	0.79	0.18
Extra Tree	0.93	0.84	1.6	0.95	0.87	1.7	0.95	0.85	1.81	0.94	0.85	1.7
Bagging Tree	0.92	0.8	0.95	0.90	0.80	1.04	0.87	0.78	1.13	0.89	0.79	1.04
RF	0.89	0.77	0.19	0.91	0.79	0.2	0.9	0.77	0.2	0.9	0.77	0.19
GNB	0.85	0.80	0.09	0.85	0.81	0.12	0.85	0.79	0.1	0.85	0.80	0.1
MNB	0.89	0.90	0.04	0.88	0.92	0.04	0.91	0.66	0.04	0.89	0.82	0.04
BNB	0.88	0.78	0.08	0.88	0.76	0.1	0.87	0.77	0.09	0.88	0.77	0.09

Table 26: Results of the binary classification for performance requirements



Encoding/ Algorithm	POS			BOW			TF-IDF			Average		
	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time
SVM	0.92	0.88	0.07	0.93	0.88	0.08	0.96	0.86	0.07	0.94	0.87	0.07
SGD SVM	0.90	0.86	0.16	0.93	0.83	0.19	0.94	0.80	0.18	0.92	0.83	0.18
LR	0.96	0.85	0.05	0.97	0.87	0.07	0.92	0.65	0.06	0.95	0.79	0.06
DT	0.87	0.83	0.16	0.87	0.82	0.17	0.85	0.84	0.18	0.86	0.83	0.17
Extra Tree	0.96	0.89	1.5	0.94	0.88	1.61	0.95	0.85	1.78	0.95	0.87	1.63
Bagging Tree	0.92	0.83	0.87	0.91	0.84	0.98	0.90	0.84	1	0.91	0.84	0.95
RF	0.9	0.81	0.17	0.91	0.8	0.19	0.90	0.75	0.2	0.9	0.79	0.19
GNB	0.76	0.74	0.13	0.79	0.74	0.1	0.78	0.71	0.1	0.78	0.73	0.11
MNB	0.88	0.87	0.04	0.9	0.9	0.05	0.82	0.64	0.06	0.87	0.80	0.05
BNB	0.92	0.77	0.08	0.90	0.73	0.1	0.89	0.74	0.09	0.90	0.75	0.09

Table 27: Results of the binary classification for operational requirements



Encoding/ Algorithm	POS			BOW			TF-IDF			Average		
	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time
SVM	0.85	0.84	0.07	0.88	0.83	0.07	0.95	0.78	0.07	0.89	0.82	0.07
SGD SVM	0.9	0.7	0.17	0.87	0.77	0.19	0.87	0.73	0.18	0.88	0.73	0.18
LR	0.87	0.78	0.06	0.91	0.8	0.07	0.89	0.77	0.07	0.89	0.78	0.07
DT	0.71	0.68	0.22	0.75	0.71	0.24	0.76	0.71	0.24	0.74	0.70	0.23
Extra Tree	0.93	0.77	1.59	0.95	0.79	1.66	0.93	0.71	1.87	0.93	0.76	1.7
Bagging Tree	0.86	0.69	1.08	0.79	0.67	1.2	0.88	0.7	1.26	0.84	0.69	1.18
RF	0.90	0.72	0.19	0.93	0.72	0.2	0.83	0.65	0.2	0.89	0.7	0.19
GNB	0.83	0.7	0.09	0.91	0.76	0.1	0.85	0.72	0.1	0.86	0.72	0.1
MNB	0.79	0.74	0.04	0.86	0.84	0.04	0.82	0.80	0.48	0.82	0.79	0.42
BNB	0.7	0.57	0.08	0.74	0.6	0.09	0.70	0.59	0.1	0.71	0.59	0.09

As shown in Table 24, for the binary classification of usability non-functional requirements, according to average results, SVM achieved the best results with precision, recall, and execution time of 0.95, 0.91, and 0.09s. Compared with the SVM classifier trained for multi class classification of NFRs, this binary SVM achieved both higher recall and higher precision.

Considering all of the combinations, the three top classifiers are same as the multi-class classifier: SVM, LR, and MNB. To be accurate, SVM with TF-IDF achieved precision, recall, and execution time of 0.95, 0.92, and 0.08s; LR with POS achieved precision, recall, and execution time of 0.94, 0.9, and 0.07s; and MNB with POS achieved precision, recall, and execution time of 0.93, 0.9, and 0.04. Generally, binary classifier for usability performed better than the multi-class NFR classifier; however, it should be noted that BNB, RF and DT achieved the worst results.

As shown in Table 25, for the binary classification of security non-functional requirements, according to average results, SVM achieved the best results with precision, recall, and execution time of 0.92, 0.89, and 0.06s. Compared to the SVM classifier trained for multi class classification of NFRs, this binary SVM performed almost the same.

Considering all of the combinations, the top classifiers are SVM and MNB. To be accurate, SVM with BoW achieved precision, recall, and execution time of 0.94, 0.91, and 0.07s; and MNB with BoW achieved precision, recall, and execution time of 0.88, 0.92, and 0.04. Generally, binary classifier for security performed slightly better than the multi-class NFR classifier; however, it should be stressed that BNB, LR and DT did not perform well.

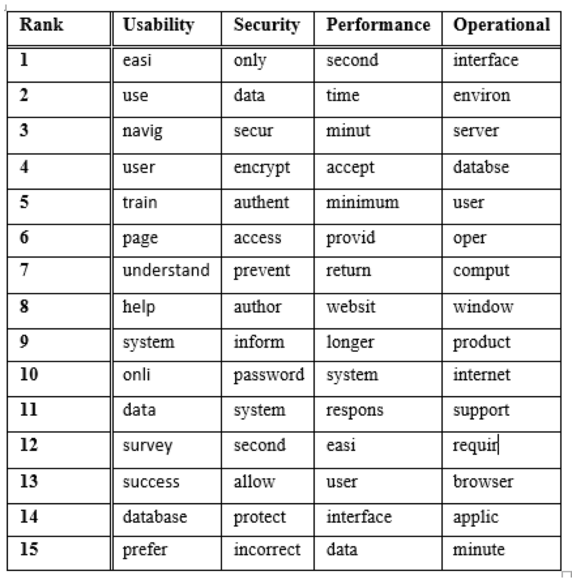
As shown in Table 26, for the binary classification of performance requirements, SVM achieved the best average results with precision, recall, and execution time of 0.94, 0.87, and 0.07s. Compared to the SVM classifier trained for multi class classification of NFRs, this binary SVM achieved higher precision but less recall values.

Considering all of the combinations, the top classifiers are Extra Tree, MNB, SVM and LR. To be accurate, Extra Tree with POS achieved precision, recall, and execution time of 0.96, 0.89, and 1.5s (worst execution time); MNB with BoW achieved precision, recall, and execution time of 0.9, 0.9, and 0.05s, SVM with BoW achieved precision, recall, and execution time of 0.93, 0.88, and 0.08s; and LR with BoW achieved precision, recall, and execution time of 0.97, 0.87, 0.07. Generally, binary classifier of performance performed better than the multi-class classifier according to precision values; nevertheless, according to recall values, although it still performed well, the multi-class NFR classifier could achieve higher recall values. Another interesting observation is that LR with TF-IDF did not perform very well, although it achieved good results in combination with BoW feature extraction. In this binary classification task, BNB and GNB did not perform well but DT performed better than usability and security binary classifiers.

As shown in Table 27, for the binary classification of operational non-functional requirements, according to average results, SVM achieved the best results with precision, recall, and execution time of 0.89, 0.82, and 0.07s. As we can see, multi-class NFR classifier performed better than this binary classifier. We are going to discuss the reasons why some classifiers outperform the others and why some are not performing well. We should point out that the main reason that the operational binary classifier did not perform well is because of the number of instances available in the training dataset. The less the samples are available, the worse the classifier preformed during our investigation of NFR classification.

Considering all the combinations, the top classifiers are SVM with POS, which achieved precision, recall, and execution time of 0.85, 0.84, and 0.07s; MNB with BoW, which achieved precision, recall, and execution time of 0.86, 0.84, and 0.04s; LR with BoW, that achieved a precision, recall, and execution time of 0.91, 0.80, and 0.07s. Same as average results, we can see that this binary classifier performed worse than the other three binary classifiers (usability, security, performance). It should be mentioned that same as other three classifiers, BNB and DT did not perform well with operational binary classification.

Table 28: Top 15 indicator terms learned from the training set



Rank	Usability	Security	Performance	Operational
1	easi	only	second	interface
2	use	data	time	environ
3	navig	secur	minut	server
4	user	encrypt	accept	databse
5	train	authent	minimum	user
6	page	access	provid	oper
7	understand	prevent	return	comput
8	help	author	websit	window
9	system	inform	longer	product
10	onli	password	system	internet
11	data	system	respons	support
12	survey	second	easi	requir
13	success	allow	user	browser
14	database	protect	interface	applic
15	prefer	incorrect	data	minute

5.4 Feature Selection

As mentioned above, in order to assess the feature importance, we use binary classifiers for each of the four NFR types (usability, security, performance, operation) to introduce the top 15 informative keywords for each of the four mentioned NFRs. As it was mentioned in Chapter 4, Section 4.3, our scoring classifier is an ensemble of tree classifiers which consists of Adaptive Boost, Extra Tree, Gradient Boost, and Random Forest. For each feature, we average the sum over all feature importance scores. And then, we rank the features according to their importance and select the top 15. Table 21 indicates these ranked keywords.

5.5 Discussion

The results show that usability binary classifier achieved the best results. Then security, performance, and operation classifiers with this order achieved their results. The comparison of results indicates that the trend of the classification results of the four NFR types we did the binary classification for, largely follows the trend of the number of the available requirement samples in an unbalanced dataset. Automatic classification performs worse when the size of certain types is smaller. This finding is interesting because it suggests that to some extent, increasing the size of certain types in the training dataset can improve the classification results of the types. This hypothesis should be further investigated with more datasets.

Comparing the results of the binary classification of the four chosen NFRs with the multi-class classification of NFRs shows that generally both recall and precision got better using binary classifiers, but precision values increased more than recall values. As explained before, precision ensures that all retrieved requirements are truly relevant, whereas recall focuses on retrieving all relevant requirements. That said, based on the results, using binary classifiers decreased the number of false positives more than it decreased the number of false negatives. Although the disadvantage of using binary classifiers is that in large scale projects with a huge number of requirements, building a binary classifier for each of the NFR types might not be easy, the results show that it can improve the performance of the classification.

The results show that SVM with linear kernel, Linear Regression, and Multinomial Naïve Bayes classifiers performed well with both multi-class NFR classifier and binary classifiers with MNB being the fastest. These classifiers were also found in former research projects as the best classifier for classification of requirement sentences comparing it to other classifiers [52, 53].

To choose the best classifier in practice, we considered execution time as well as recall and precision. Table 16 and Figure 19 represent the execution time of classifiers used in our classification. These values represent the whole execution time of the measurement using the same environment. According to our experiments, the process based on Multinomial Naïve Bayes has produced the best execution time. Based on the precision, recall, and the execution time, the Multinomial Naïve Bayes classifier is the best choice in practice for classification of requirement sentences. This classifier was identified as the best based on its performance by some former research [53, 54]. Our research complemented the comparison with other popular classifiers as well. However, classifiers like Logistic Regression or Support Vector Machine have performed a bit better regarding to their precision and recall, but considering the execution time, Multinomial Naïve Bayes classifier can be denoted as the best choice for practice.

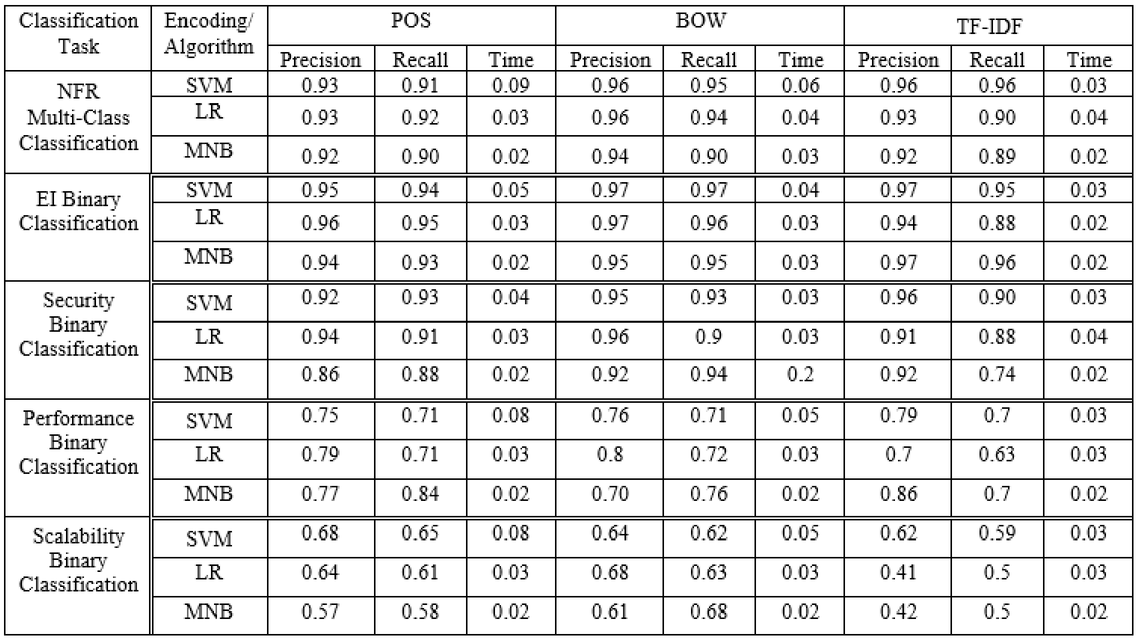
Observing the results of binary classifiers in Table 24, 25, 26, and 27, one of the classifiers that produced weak precision and recall results is the Bernoulli Naïve Bayes classifier. This classifier binarizes the feature vector before it performs the classification process because its algorithm works on feature vector containing Boolean values. Using BoW and TF-IDF representation, this approach has been proved wrong for classification of requirement sentences.

5.6 Case Study

Based on our results, three of the classifiers (SVM, LR, and MNB) performed well with the non-functional requirement classification task. To validate these results even more, we tried the experiments with the three top classifiers for the multi-class classification task of NFRs. We also performed binary classification for the top four NFR types in the dataset. This industry dataset includes 262 NFRs including 68 external interface, 45 security, 44 scalability, and 31 performance requirements, etc. Table 28 shows the credible results of these three classifiers. As we can see, in multi-class classification, all three classifiers achieved recall values above 0.91 and precision values above 0.92 with execution time less than 0.09s.

As discussed before, the number of available requirement instances in the dataset affect the classifier’s performance when we use binary classifiers. The binary classifier for external interface and security achieved recall values above 0.74 and precision values above 0.86. The binary classifier of performance achieved moderate results. Nonetheless, the binary classifier of scalability did not achieve good results (recall values and precision values less than 0.68s) compared to the other three binary classifiers. This can be due to the smaller number of requirement instances of this type available in the dataset.

Table 29: NFR classification of the industry dataset



Classification Task	Encoding/ Algorithm	POS			BOW			TF-IDF		
		Precision	Recall	Time	Precision	Recall	Time	Precision	Recall	Time
NFR Multi-Class Classification	SVM	0.93	0.91	0.09	0.96	0.95	0.06	0.96	0.96	0.03
	LR	0.93	0.92	0.03	0.96	0.94	0.04	0.93	0.90	0.04
	MNB	0.92	0.90	0.02	0.94	0.90	0.03	0.92	0.89	0.02
EI Binary Classification	SVM	0.95	0.94	0.05	0.97	0.97	0.04	0.97	0.95	0.03
	LR	0.96	0.95	0.03	0.97	0.96	0.03	0.94	0.88	0.02
	MNB	0.94	0.93	0.02	0.95	0.95	0.03	0.97	0.96	0.02
Security Binary Classification	SVM	0.92	0.93	0.04	0.95	0.93	0.03	0.96	0.90	0.03
	LR	0.94	0.91	0.03	0.96	0.9	0.03	0.91	0.88	0.04
	MNB	0.86	0.88	0.02	0.92	0.94	0.2	0.92	0.74	0.02
Performance Binary Classification	SVM	0.75	0.71	0.08	0.76	0.71	0.05	0.79	0.7	0.03
	LR	0.79	0.71	0.03	0.8	0.72	0.03	0.7	0.63	0.03
	MNB	0.77	0.84	0.02	0.70	0.76	0.02	0.86	0.7	0.02
Scalability Binary Classification	SVM	0.68	0.65	0.08	0.64	0.62	0.05	0.62	0.59	0.03
	LR	0.64	0.61	0.03	0.68	0.63	0.03	0.41	0.5	0.03
	MNB	0.57	0.58	0.02	0.61	0.68	0.02	0.42	0.5	0.02

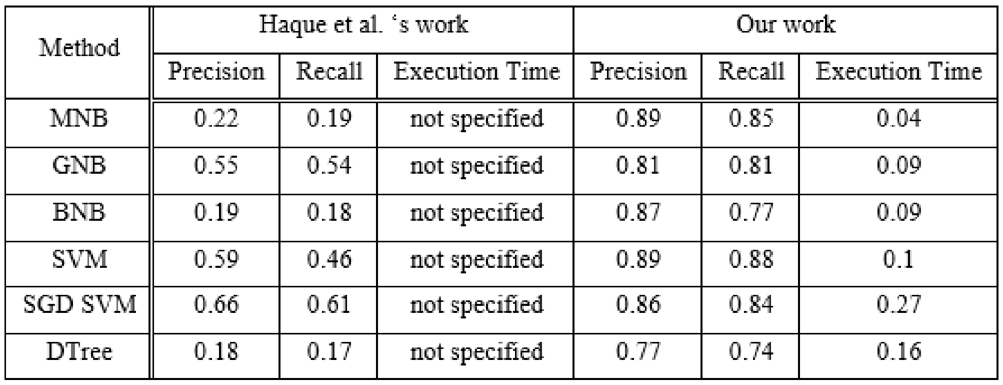
It should be noted that there is more possibility to validate our solution by using datasets from different domains and large-scale industry projects. This is outside the scope of this thesis.

5.7 Comparison with Related Work

In this section, we will compare our results with two similar studies that have been done in the classification of NFRs using supervised machine learning algorithms to put our results into context.

1. In a similar study [55], the authors used the NFR PROMISE dataset to automatically classify NFRs to find out the best combination of four feature extraction techniques including BOW, TF-IDF (character level), TF-IDF (word level), and N-gram with seven machine learning techniques consisting of Multinomial-, Gaussian- and Bernoulli Naïve Bayes, K-Nearest Neighbor, Support Vector Machine, SGD SVM, and Decision Tree algorithms. They found that, SGD SVM classifier achieved the best results with precision and recall of 0.66 and 0.61. Table 23 indicates the comparison of their results with ours (we only included the mutual classifiers used in both researches works).

Table 30: Comparison with Haque et al. 's work



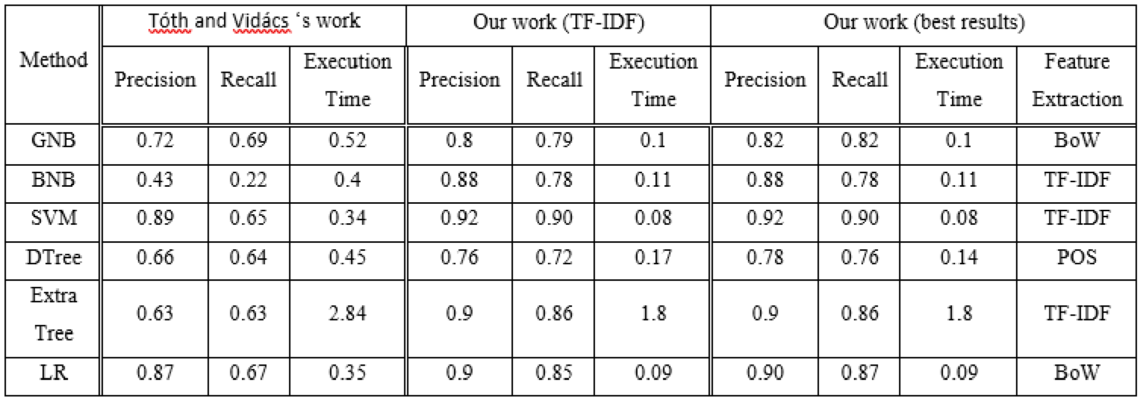
Method	Haque et al. ‘s work			Our work		
	Precision	Recall	Execution Time	Precision	Recall	Execution Time
MNB	0.22	0.19	not specified	0.89	0.85	0.04
GNB	0.55	0.54	not specified	0.81	0.81	0.09
BNB	0.19	0.18	not specified	0.87	0.77	0.09
SVM	0.59	0.46	not specified	0.89	0.88	0.1
SGD SVM	0.66	0.61	not specified	0.86	0.84	0.27
DTree	0.18	0.17	not specified	0.77	0.74	0.16

We can clearly see from Table 29 that our classifiers achieved higher recall and precision values. Some noteworthy points: (a) the MNB classifier underperformed in Haque et al.’s work although it has been claimed in several other research works that MNB achieved good results (e.g. precision and recall of 0.84 and 0.68 in [54]). Their best results are achieved by SGD SVM. In our investigation, SGD SVM also performed well with precision and recall values of 0.86 and 0.84 but SVM with Linear kernel performed better as this was also indicated in former research works that SVM with linear kernel performs well with requirement classification task [29, 69]

In Haque et al.’s research work, it is not specifically mentioned whether they trained a multi-class classifier or a binary classifier; however, based on their indicated results (they have not showed recall and precision values for each of the NFR types), it seems that they have used a multi-class classifier. Considering that they have used only the PROMISE dataset, which is small in size with 625 requirements overall (255 functional and 370 non-functional), we can conclude that the performance of these classifiers was highly affected by the small size of their dataset. Another important issue is that there are 11 different types of NFR in the PROMISE dataset some of which only have 1 requirement instance. Including these types with an insufficient number of requirement instances makes the training dataset highly imbalanced, which consequently affects the classifiers’ performance [55]. Another factor could be the parameter settings of their classifiers. They have not given any details of the parameter values that they have used for these classifiers, but that could be another possible reason for the difference in their classifiers’ performance. It should be mentioned that investigation is required to define the exact reasons behind the difference in the performance of same classifiers in different studies.

2. In another similar study, Tóth and Vidács conducted a comparative study of the performance of various classifiers in labeling non-functional requirements [54]. They used the NFR PROMISE dataset for their experiments and converted the dataset to TF-IDF representation. Different classifiers including Multinomial-, Gaussian- and Bernoulli Naïve Bayes, Support Vector Machine with linear kernel, Linear Logistic Regression, Decision Tree, and Extra Tree were applied on the dataset. Table 24 indicates the comparison of their results with those of ours (we only included the mutual classifiers used in both researches works). It should be noted that we included two results of our work for each of the classifiers. First results are achieved using TF-IDF feature extraction. This enables us to compare the results of two works based on factors other than feature extraction techniques. We also included the highest results achieved. That enables us to observe the probable effect of using a different feature extraction technique.

Table 31: Comparison with Tóth et al.'s work



Method	Tóth and Vidács 's work			Our work (TF-IDF)			Our work (best results)			
	Precision	Recall	Execution Time	Precision	Recall	Execution Time	Precision	Recall	Execution Time	Feature Extraction
GNB	0.72	0.69	0.52	0.8	0.79	0.1	0.82	0.82	0.1	BoW
BNB	0.43	0.22	0.4	0.88	0.78	0.11	0.88	0.78	0.11	TF-IDF
SVM	0.89	0.65	0.34	0.92	0.90	0.08	0.92	0.90	0.08	TF-IDF
DTree	0.66	0.64	0.45	0.76	0.72	0.17	0.78	0.76	0.14	POS
Extra Tree	0.63	0.63	2.84	0.9	0.86	1.8	0.9	0.86	1.8	TF-IDF
LR	0.87	0.67	0.35	0.9	0.85	0.09	0.90	0.87	0.09	BoW

As indicated in the above table, the results are similar to our results in a sense that in both works, three of the classifiers performed better than the other classifiers in this set of algorithms. MNB, LR and SVM with linear kernel have produced the best results. Comparing their results with our results in which we used TF-IDF as the feature extraction technique, we could see that we achieved better results (e.g. with SVM classifier they achieved recall of 0.65 and we achieved recall of 0.90). Considering that both feature extraction techniques and classifiers used are the same, this could be because of the difference in the training dataset. Tóth et al. stated in their work that the size of the training dataset is small and very unbalanced, and they concluded that based on their achieved results, more validation is required using a bigger labeled dataset. Our research work indicates that using a larger and more balanced dataset will potentially improve the performance of these classifiers (as shown in Table 19). That said, we also included the best results achieved by these classifiers in our work. As is shown in Table 24, best results of BNB, SVM and Extra Tree were achieved by TF-IDF, but the best results achieved by GNB, DTree, and LR were achieved by BoW, POS, and BoW. The difference between the precision and recall values are not much but it indicates that using different feature extraction technique could potentially improve the classifiers' performance. This can be further investigated as a future step in this research.

Chapter 6

6 Conclusion and Future work

Identification of non-functional requirements (NFRs) from a given set of mixed functional requirements (FRs) and NFRs in a requirements specification document and classification of NFRs according to the stakeholders' concerns is an important aspect of large, complex software project [11]. In this thesis research, we studied how accurately we can automatically classify requirements as functional and non-functional in the dataset, using feature extraction techniques and supervised machine learning algorithms. Furthermore, we assessed how accurately we can identify various types of NFRs, in particular, usability, security, performance, and operational requirements. In this project, we applied three feature extraction techniques including POS-tagging bases, BoW, and TF-IDF, and 10 supervised machine learning algorithms including SVM, SGD-SVM, LR, Decision Tree, Bagging Tree, Extra Tree, Random Forest, Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes.

FR-NFR classification results shown in Table 16, indicates that generally, all applied classifiers, except GNB and SGD-SVM, achieved a minimum recall of 0.83 and minimum precision of 0.82 with MNB being the fastest. With reference to the average results, LR with recall, precision, and execution time of 0.91, 0.88, and 0.25s outperformed other classifiers. Considering all combinations of feature extraction techniques and SML algorithms, SVM, LR, and MNB with TF-IDF feature extraction achieved the best results.

Average of the results shown in Table 20, indicates that in the classification of non-functional requirements, SVM outperformed the other classifiers with recall, precision and execution time of 0.88, 0.89, and 0.1. Considering all combinations of feature extraction techniques and SML algorithms, SVM with TF-IDF achieved the best results with recall, precision and execution time of 0.9, 0.92, and 0.08. It should be mentioned that SVM performed achieved high recall and precision values with all three feature extraction techniques. Other combinations that performed well are LR with BoW and POS with recall, precision, and execution time of 0.87, 0.90, and 0.09; and MNB with BoW with recall, precision, and execution time of 0.88, 0.9, and 0.05. Based on these results, MNB classifier is the fastest, and although SVM and LR achieved a bit higher results, considering all evaluation metrics (recall, precision, and execution time), MNB is the best solution.

We also investigated whether using binary classifiers for each of NFR types could possibly improve performance of the NFR classification or not. Thus, for each of the four NFR types (usability, security, performance, and operational), we trained binary classifiers using the above mentioned SML algorithms. The results shown in Table 23, 24, 25, and 26 indicates that binary classifiers achieve higher recall and precision values compared to the multi-class NFR classifier. The three top classifiers are SVM, LR, and MNB with MNB being the fastest classifier among all 10 SML algorithms we used.

To validate our results, we applied the three top classifiers including SVM, LR, and MNB on a separate industry dataset to classify non-functional requirements. The results in Table 27, indicated that these classifiers performed well with the case study dataset with average recall values above 0.89 and average precision values above 0.92. We also applied binary classifications on the top four NFRs available in the dataset including external interface, security, performance, and scalability NFRs. All these classifiers achieved good results with recall and precision mostly above 0.9, except scalability classifier with recall and precision less than 0.7. One explanation for the bad performance of scalability binary classifier is the smaller number of requirement instances in the dataset [59].

Developing a tool based on the proposed strategy for the identification and classification of NFRs, can support different stakeholders, such as: (i) architects to whom architecturally significant requirements (e.g., performance, efficiency, and interface) are important in choosing software architectural decisions [3]; (ii) business Analysts to whom business-related NFRs (e.g., security, availability, and usability) are important in the business [4]; and (iii) developers with specific expertise (e.g., user interface, security, and database) [7]. These stakeholders can use such a tool in several phases of the project to ensure the completeness of their work.

6.1 Limitation and Threats to Validity

Limitation: In our training dataset, each requirement was labeled with one type, but in reality, one requirement sentence may belong to more than one type of NFRs or FRs. During the process of extracting requirement sentences from software requirement specification documents, there were requirements with multiple types. We manually separated these sentences into more fine-grained sentences which can be labelled in one type before conducting the classification investigation. For instance, the sentence "all fields of an edited asset can be modified except Ids and the system shall employ consistent colors and symbols for editable fields." should be classified as usability NFR and FR, we separated this requirement sentence into two sentences, one is "all fields of an edited asset can be modified except Ids" which was classified as usability NFR and the other is "the system shall employ consistent colors and symbols for editable fields." which was classified as FR. This is currently a limitation of our approach which cannot attach multiple labels to one requirement sentence. Future work is needed to overcome this limitation.

Internal validity focuses on the design of a study, especially whether the results follow from the data. One threat to the internal validity in this study is that the tests overfit the machine learning. Overfitting happens when a model learns the datil I the training data to the extent that it negatively impacts the performance of the model on new data [37]. To mitigate the influence of this threat, we used our initial training data to generate multiple mini train-test splits and we used these splits to tune our model by applying a 10-fold cross-validation in our experiments.

External validity refers to the degree to which our findings from this study can be generalized in other projects. To mitigate possible threat, we conducted the experiments on two different datasets: the NFR PROMISE dataset which contained 15 different projects and the PURE dataset which contained 6 different projects. We also used a separate case study dataset which belongs to the transportation domain. The diversity of the chosen project domains increases the generalizability of the experiment results and decreases any sampling bias. But it is still unclear at this stage whether this experiment can attain similar results when being applied to other project domains. Another threat to external validity is that we only performed our classification on four NFR types because there were not enough number of other NFR types in the dataset. Ona way to improve this is to collect larger datasets with a greater number of available NFRs and conduct the classification tasks on more types.

Reliability refers to whether the study yields the same results if other researchers replicate this study, which in this work is related to the processes of automatic NFR classification (the process of the investigation). By making explicit these processes and providing a pseudo code of the main function of the classification task (as detailed in Chapter 4), we consider this threat to be contained.

6.2 Future Work

The work performed in this project considers product quality attributes such as usability, performance, security, scalability, etc. A further step in providing a concern-based NFR classification tool for stakeholders is to consider process quality aspects such as risk, effort, cost as well. This would enable stakeholders to prioritize specific NFRs based on complementary aspects and they can use it as a base line in their decision making.

The results showed that our proposed classifiers generally perform well independent of the feature extraction technique they have been combined with. However, in some cases that we discussed in chapter 6, specific feature extraction technique produced better results. For example, in NFR multi-class classification, although SVM performed well with all three feature extraction techniques, it produced higher recall and precision values with TF-IDF. Investigating other techniques such as AUR-BoW proposed by [1] could potentially improve our results.

As mentioned earlier, using more industry datasets with a greater number of requirement instances and more NFR types available can be used to investigate the validity of this work even more. It will also enable the investigation of classifiers for more NFR types such as safety, reliability, etc.

References

[1] Davis, Gordon B. "Strategies for information requirements determination." *IBM systems journal* 21, no. 1 (1982): 4-30.

[2] Kaur, Ranjeet, and Tajinder Singh. "Analysis and need of requirements engineering." *International Journal of Computer Applications* 7, no. 14 (2010): 27-32.

[3] Sommerville, Ian, and Pete Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997.

[4] Lauesen, Soren. *Software requirements: styles and techniques*. Pearson Education, 2002.

[5] Automatic Classification of Non-Functional Requirements from Augmented App User Reviews, John, 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)*, pp. 9-16. IEEE, 2013.

[6] Bajpai, Vikas, and Ravi Prakash Gorthi. "On non-functional requirements: A survey." In *2012 IEEE Students' Conference on Electrical, Electronics and Computer Science*, pp. 1-4. IEEE, 2012.

[7] Finkelstein, Anthony, 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*, pp. 2-4. IEEE, 1996.

[8] Dalcher, Darren. "Disaster in London. The LAS case study." In *Proceedings ECBS'99. IEEE Conference and Workshop on Engineering of Computer-Based Systems*, pp. 41-52. IEEE, 1999.

[9] Jung, Hyo Taeg, and Gil-Haeng Lee. "A systematic software development process for non-functional requirements." In *2010 International conference on information and communication technology convergence (ICTC)*, pp. 431-436. IEEE, 2010.

[10] Maiti, Richard R., and Frank J. Mitropoulos. "Capturing, eliciting, predicting and prioritizing (CEPP) non-functional requirements metadata during the early stages of agile software development." In *SoutheastCon 2015*, pp. 1-8. IEEE, 2015.

[11] Casamayor, Agustin, Daniela Godoy, and Marcelo Campo. "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach." *Information and Software Technology* 52, no. 4 (2010): 436-445.

[12] Cleland-Huang, Jane, Raffaella Settimi, Xuchang Zou, and Peter Solc. "Automated classification of non-functional requirements." *Requirements Engineering* 12, no. 2 (2007): 103-120.

[13] Kazman, Rick, and Len Bass. *Toward deriving software architectures from quality attributes*. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1994.

[14] Ameller, David, Claudia Ayala, Jordi Cabot, and Xavier Franch. "Non-functional requirements in architectural decision making." *IEEE software* 30, no. 2 (2012): 61-67.

[15] Xu, Lihua, Hadar Ziv, Thomas A. Alspaugh, and Debra J. Richardson. "An architectural pattern for non-functional dependability requirements." *Journal of Systems and Software* 79, no. 10 (2006): 1370-1378.

[16] Pavlovski, Christopher J., and Joe Zou. "Non-Functional Requirements in Business Process Modeling." In *APCCM*, vol. 8, pp. 103-112. 2008.

[17] Reifer, Donald J. *Making the software business case: Improvement by the numbers*. Pearson Education, 2001.

[18] Ferreira, Susan, James Collofello, Dan Shunk, and Gerald Mackulak. "Understanding the effects of requirements volatility in software engineering by using analytical modeling and software process simulation." *Journal of Systems and Software* 82, no. 10 (2009): 1568-1577.

[19] Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1, no. 2. Cambridge: MIT press, 2016.

[20] Nuseibeh, Bashar, and Steve Easterbrook. "Requirements engineering: a roadmap." In *Proceedings of the Conference on the Future of Software Engineering*, pp. 35-46. 2000.

[21] Mitchell, Tom. "Machine Learning. MacGraw-Hill Companies." *Inc., Boston* (1997).

[22] Glinz, Martin. "Rethinking the notion of non-functional requirements." In *Proc. Third World Congress for Software Quality*, vol. 2, pp. 55-64. 2005.

[23] Kannan, Subbu, and Vairaprakash Gurusamy. "Preprocessing techniques for text mining." *International Journal of Computer Science & Communication Networks* 5, no. 1 (2014): 7-16.

[24] Chung, Lawrence, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*. Vol. 5. Springer Science & Business Media, 2012.

[25] Afreen, Nida, Asma Khatoon, and Mohd Sadiq. "A taxonomy of software's non-functional requirements." In *Proceedings of the second international conference on computer and communication technologies*, pp. 47-53. Springer, New Delhi, 2016.

[26] Van Lamsweerde, Axel. *Requirements engineering: From system goals to UML models to software*. Vol. 10. Chichester, UK: John Wiley & Sons, 2009.

[27] Mairiza, Dewi, Didar Zowghi, and Nurie Nurmuliani. "An investigation into the notion of non-functional requirements." In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 311-317. 2010.

[28] Mairiza, Dewi, and Didar Zowghi. "Constructing a catalogue of conflicts among non-functional requirements." In *International Conference on Evaluation of Novel Approaches to Software Engineering*, pp. 31-44. Springer, Berlin, Heidelberg, 2010.

[29] Rashwan, Abderahman. "Automated quality assurance of non-functional requirements for testability." PhD diss., Concordia University, 2015.

[30] Sabriye, Ali Olow Jim'ale, and Wan Mohd Nazmee Wan Zainon. "A framework for detecting ambiguity in software requirement specification." In *2017 8th International Conference on Information Technology (ICIT)*, pp. 209-213. IEEE, 2017.

[31] Savage, Grant T., Timothy W. Nix, Carlton J. Whitehead, and John D. Blair. "Strategies for assessing and managing organizational stakeholders." *Academy of management perspectives* 5, no. 2 (1991): 61-75.

[32] Wateridge, John. "How can IS/IT projects be measured for success?." *International journal of project management* 16, no. 1 (1998): 59-63.

[33] Guide to the Systems Engineering Body of Knowledge (SEBoK). (n.d.). Retrieved November 27, 2020, from ttps://sebokwiki.org/wiki/Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK)

[34] Kurtanović, Zijad, and Walid Maalej. "Automatically classifying functional and non-functional requirements using supervised machine learning." In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pp. 490-495. Ieee, 2017.

[35] Chung, Lawrence, and Brian A. Nixon. "Dealing with non-functional requirements: three experimental studies of a process-oriented approach." In *1995 17th International Conference on Software Engineering*, pp. 25-25. IEEE, 1995.

[36] Eckhardt, Jonas, 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*, pp. 832-842. 2016.

[37] Mitchell, Tom. "Machine Learning. MacGraw-Hill Companies." *Inc., Boston* (1997).

[38] Kotsiantis, Sotiris B., I. Zaharakis, and P. Pintelas. "Supervised machine learning: A review of classification techniques." *Emerging artificial intelligence applications in computer engineering* 160, no. 1 (2007): 3-24.

[39] Zhang, Wen, 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)*, pp. 190-195. 2011.

[40] Wright, Raymond E. "Logistic regression." (1995).

[41] Vijayan, Vikas K., K. R. Bindu, and Latha Parameswaran. "A comprehensive study of text classification algorithms." In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1109-1113. IEEE, 2017.

[42] Zhou, Zhi-Hua. "Ensemble Learning." *Encyclopedia of biometrics* 1 (2009): 270-273.

[43] Breiman, Leo. "Bagging predictors." *Machine learning* 24, no. 2 (1996): 123-140.

[44] Chan, Jonathan Cheung-Wai, and Desiré Paelinckx. "Evaluation of Random Forest and Adaboost tree-based ensemble classification and spectral band selection for ecotope mapping using airborne hyperspectral imagery." *Remote Sensing of Environment* 112, no. 6 (2008): 2999-3011.

[45] Schapire, Robert E. "The boosting approach to machine learning: An overview." In *Nonlinear estimation and classification*, pp. 149-171. Springer, New York, NY, 2003.

[46] Ferrari, Alessio, Giorgio Oronzo Spagnolo, and Stefania Gnesi. "Pure: A dataset of public requirements documents." In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pp. 502-505. IEEE, 2017.

[47] Hussain, Ishrar, 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*, pp. 287-298. Springer, Berlin, Heidelberg, 2008.

[48] Nigam, K., A. McCallum, and S. Thrun. "& Mitchell, T.(2000).“Text classification from labeled and unlabeled documents using EM.”." *Machine Learning* 39, no. 2/3: 103-134.

[49] De Marneffe, Marie-Catherine, Bill MacCartney, and Christopher D. Manning. "Generating typed dependency parses from phrase structure parses." In *Lrec*, vol. 6, pp. 449-454. 2006.

[50] Riaz, Maria, Jason King, John Slankas, and Laurie Williams. "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts." In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pp. 183-192. IEEE, 2014.

[51] Jindal, Rajni, Ruchika Malhotra, and Abha Jain. "Automated classification of security requirements." In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2027-2033. IEEE, 2016.

[52] Lu, Mengmeng, 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*, pp. 344-353. 2017.

[53] Abad, Zahra Shakeri Hossein, 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)*, pp. 496-501. IEEE, 2017.

[54] Tóth, László, and László Vidács. "Study of various classifiers for identification and classification of non-functional requirements." In *International Conference on Computational Science and Its Applications*, pp. 492-503. Springer, Cham, 2018.

[55] Haque, Md Ariful, Md Abdur Rahman, and Md Saeed Siddik. "Non-Functional Requirements Classification with Feature Extraction and Machine Learning: An Empirical Study." In *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, pp. 1-5. IEEE, 2019.

[56] Kobilica, Armin, Mohammed Ayub, and Jameleddine Hassine. "Automated Identification of Security Requirements: A Machine Learning Approach." In *Proceedings of the Evaluation and Assessment in Software Engineering*, pp. 475-480. 2020.

[57] Binkhonain, Manal, and Liping Zhao. "A review of machine learning algorithms for identification and classification of non-functional requirements." *Expert Systems with Applications: X* 1 (2019): 100001.

[58] Khelifa, Amani, Mariem Haoues, and Asma Sellami. "Towards a Software Requirements Change Classification using Support Vector Machine." In *LPKM*. 2018.

[59] Kobilica, Armin, Mohammed Ayub, and Jameleddine Hassine. "Automated Identification of Security Requirements: A Machine Learning Approach." In *Proceedings of the Evaluation and Assessment in Software Engineering*, pp. 475-480. 2020.

[60] Caruana, Rich. "Learning from imbalanced data: Rank metrics and extra tasks." In *Proc. Am. Assoc. for Artificial Intelligence (AAAI) Conf*, pp. 51-57. 2000.

[61] Deocadez, Roger, Rachel Harrison, and Daniel Rodriguez. "Preliminary study on applying semi-supervised learning to app store analysis." In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pp. 320-323. 2017.

[62] Knauss, Eric, and Daniel Ott. "(Semi-) automatic categorization of natural language requirements." *In International Working Conference on Requirements Engineering: Foundation for Software Quality*, pp. 39-54. Springer, Cham, 2014.

Curriculum Vitae

Name:	Mahtab EzzatiKarami
post-secondary Education and Degrees:	Amirkabir University of Technology Tehran, Iran 2014-2018 B.Sc.
	The University of Western Ontario London, Ontario, Canada 2019-1995 M.Sc.
Related Work Experience	Teaching Assistant The University of Western Ontario 2019-2020