Cross-applicability of ML classification methods intended for (non-)functional requirements

Final Project (192199978)

Nguyen Nhu Thuy

Graduation Committee:
Supervisor & Committee Chair: Dr. Maya Daneva
Co-Supervisor & Examiner: Dr. Faiza A. Bukhsh
Examiner: Dr. Faizan Ahmed

Software Technology

Faculty of Electrical Engineering,
Mathematics and Computer Science
(EEMCS)

University of Twente.

August, 2021

A report submitted in partial fulfilment of the requirements for the degree of M.Sc. in Computer Science.

Abstract

Machine Learning (ML) has been applied to a wide variety of feeds and achieved significantly promising results. Its power arises in the ability to learn from data and make decisions based on its learning. Recognizing the impact of this cutting edge technology and how it can benefit Requirements Engineering (RE), researchers have tried applying different ML approaches onto RE tasks. The literature review "The Landscape of Machine Learning in Requirements Engineering" [1] shows that in recent years, a plethora of ML techniques have been proposed to solve the problem of classifying requirements, targeted specifically to functional or nonfunctional requirements.

In this study, we experiment the cross-applicability of these ML methods, that are intended for either functional or nonfunctional requirements, when being used to classify the other. The ML techniques found in [1] will be re-evaluated on a common dataset of (non-)functional requirements, and then be used to classify the other to compare their effectiveness. With this study, we hope to put a conclusion to our hypothesis that although not designed to, ML methods intended for classifying functional requirements can be effectively used for non-functional requirements, and vice versa.

Keywords — Machine Learning, ML, Requirement Engineering, RE, Requirement Classification

Contents

List of Abbreviations

ANN	Artificial Neural Network
BoW	Bag of Words
CNN	Convolutional Neural Network
FE	Feature Extraction
FR	Functional Requirement
GD	Gradient Decent
ML	Machine Learning
NLTK	Natural Language Toolkit
NR	Non-functional Requirement
POS	Part of Speech
Req.	Requirement
Tf-Idf	Term frequency — Inverse document frequency

List of Figures

List of Tables

1 Introduction

1.1 | Motivation

Machine Learning (ML) has become more and more pervasive in recent years because of its proven ability to automatically solve complicated problems by exploring the hidden pattern inside the given data. As it shows its ability to benefit Requirement Engineering, researchers have applied Machine Learning to solve different Requirement Engineering problems, especially the requirements classification task. However, it is observed from our literature review "The Landscape of Machine Learning in Requirements Engineering" that research on the requirements classification task targets only classifying functional requirements or non-functional requirements. In theory, the functional requirements classification problem and the non-functional requirements classification are two parts of the same problem. However, there is no practical research that shows this is the actual case. Secondly, as mentioned before, researchers have only focused on classifying either non-functional or functional requirements exclusively but not both. In this thesis, multiple experiments are conducted to investigate if it is true that they are identical in nature or there are some hidden nuances that make these two classification problems - functional requirements classification and non-functional requirements classification different. Additionally, we aim to check whether there were some unrevealed barriers that prevent classifying both types of requirements at one time.

1.2 | Goal & Scope

As stated in the Section 1.1, the main goals of this research are to investigate the cross-applicability of established methods and find out if these methods can extend its ability to classify both types of requirements.

With those goals defined, the scope of this study is as follows:

- Replicate established methods if the implementations are not publicly available
- Train and test these methods with requirements of the opposite type
- Train and test these methods with both functional requirements and non-functional requirements

1.3 | Research Questions

The following research questions are formulated to drive the project to follow the critical analysis of the cross-applicability of the models:

RQ1 How effective is it to use ML methods that were intended for functional requirements classification to classify non-functional requirements? And vice versa, how effective is it to use ML methods that were intended for non-functional requirements classification to classify functional requirements?

RQ2 How effective is it to use ML methods that were intended for either functional or non-functional requirements classification to classify both types of requirements?

RQ3 *[Extra]* What other ML methods can also be used for the requirements classification problem?

1.4 | Document Structure

This final report is structured as follows. Chapter 2 presents the background knowledge and the related work. Chapter 3 explains the methodology used to structure our study. Chapter 5 discusses experiments conducted in this project while 6 reports results obtained from experiments and discusses findings from results. Chapter 7 concludes.

2 Background

2.1 | Requirements

According to Sommerville [1], the specifications of what a system is expected to offer and constraints/criteria on its functionalities are called requirements. Requirements are of such importance that they have their own field (i.e. Requirements Engineering) because they give an orientation of what and how a system should become, show needed goals for developing teams to achieve a complete system and help validate and verify the deliverable.

Normally, requirements are categorized into two categories: functional and non-functional requirements. Functional requirements are those that describe the functionalities that a system should offer, its intended behaviors and reactions to different inputs while non-functional ones are those that describe properties, constraints or quality criteria of a system or its functionalities for example security, scalability, performance, reliability [2]. "The system shall allow a user to define the time segments" or "The system shall locate the preferred repair facility with the highest ratings for the input criteria" are examples for functional requirements [3]. Examples for non-functional requirements are "Product shall be able to process 100 payment transactions per second in peak load." or "When repairing a defect, related non-repaired defects shall be less than 0.5 on average." [2]

2.2 | Machine Learning

2.2.1 | What is Machine Learning?

Machine Learning is a field that tries to make machines be able to learn based on provided data without human instruction. There are four types of machine learning including supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.

A method that predicts outcomes of new inputs based on provided labeled data is called supervised learning. On the other hand, unsupervised learning is a method where unlabeled data is given instead of labeled data. This method will depend on the structure of data to perform some tasks (e.g. clustering, reducing dimension). Semi supervised learning is a method that is used when given data comprises both labeled and unlabeled data. Reinforcement learning helps a machine be able to automatically make decisions based on the context to retrieve the maximum total cumulative reward.

2.2.2 | Loss Function

To be able to learn, a machine learning model first has to calculate a value (i.e. loss) using loss function. Loss function is a function that calculates the difference between the predicted values and the true values and shows how well the performance of a model is. Currently, there are many optimization techniques and Gradient Descent (GD) is one of the algorithms that is commonly used in both academic and industrial environments.

2.2.2.1 | Gradient Descent

Gradient descent is a common technique that is used for optimization problems to increase the efficiency of a ML method by minimizing loss function.

A rate of change of a one-variable function is described using a derivative. However, for a multiple-variables function, gradient, which is a synthesis of all partial derivatives of a function, is used to show the direction to the fastest increase of a function. So to find the minimum value of a function, we have to go in the opposite direction (i.e. in the descent direction) of a gradient which explains why this technique is called Gradient Descent.

2.2.2.2 | Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a technique that picks only one sample of data to compute the gradient of the lost function and update θ values - parameters of a model.

This calculation is performed on every sample of data. If a dataset has n samples, theta values will be updated n times for one epoch. After each epoch, the dataset has to be shuffled (i.e. data will be reordered) to ensure the randomness which is why it is called stochastic gradient descent.

2.2.3 | Machine Learning Methods

2.2.3.1 | Support Vector Machine

Support Vector Machine (SVM) is a method that creates an optimal hyperplane to separate data into different classes.

2.2.3.2 | Stochastic Gradient Descent SVM

A Stochastic Gradient Descent Support Vector Machine (SGD SVM) is a version of SVM which uses Stochastic Gradient Descent to solve the SVM optimization problem by stochastically and iteratively minimizing the hinge loss function based on the direction of the gradient vector.

2.2.3.3 | K-Nearest Neighbors

K-Nearest Neighbors (KNN) is an algorithm that predicts a label of a new sample of data based on K nearest data samples in its training set because, normally, similar samples of data are near to each other. To be more specific, this method will first calculate the distance between a new data sample and all data it has, then K nearest points will be picked and decide the label of a new sample.

The advantage of this method lies in the simplicity of calculation in the training process and of prediction of new data. However, this algorithm is sensitive to noise when K is small and when the training set is huge, the prediction could take time because it has to calculate the distance from a new point to every single point in the training set.

2.2.3.4 | Naive Bayes

Naive Bayesian is a method that calculates the probability of the input's label based on Bayes' theorem with the assumption that each feature of the input is independent of each other.

2.2.3.5 | Multi-nominal Naive Bayes

Multi-nominal Naive Bayes is one of the variants of Naive Bayes. It considers features that represent the frequency, the number of times appear (e.g. word counts).

2.2.3.6 | Bernoulli Naive Bayes

Another type of Naive Bayes, namely Bernoulli Naive Bayes is a machine learning algorithm that considers binary features such as 0-1, yes-no, true-false, success-failure.

2.2.3.7 | Gaussian Naive Bayes

Gaussian Naive Bayes is a version of Naive Bayes often used for continuous data under the assumption that the data within each label follow a Gaussian (normal) distribution.

2.2.3.8 | Decision Tree

Decision tree is a supervised learning method that creates a tree structure which can predict the label or the value of the given input by learning decision rules inferred from training data. The decision rules (i.e. attributes) and their order will be arranged based on criteria such as entropy, information gain, or Gini, etc. For entropy or information gain criteria, the attribute with the lowest entropy/highest information gain value will be selected as the current non-leaf node of a tree whose attribute's values are also treated as child nodes of the tree. The process is continued on these child nodes until all child nodes are leaf nodes.

Building a tree with a lot of nodes could lead to overfitting which makes the prediction less precise in real life. To avoid this problem, a few solutions can be used including providing stopping criteria when building a tree such as tree depth, the maximum number of leaf nodes or pruning - a technique that after finishing building a complete tree, it trims leaf nodes which does not affect the overall accuracy of a tree.

2.2.3.9 | Neural Network

The Artificial Neural Network (ANN) [4] operates similarly to the way our brain's biological neural networks work. There are numerous neurons existing in human's brain and they connect to each other to receive and transmit information.

ANN do the same things. The network consists of a lot of artificial neurons, also known as perceptions. Each perceptron connects to others via weighted edges. Information between neurons will be transmitted through these edges. The weights of edges affect the importance of the input to the current neuron. If the weight is high, the input's information is very important, otherwise the information is insignificant to the current perceptron. Furthermore, these weights are not fixed numbers instead they are learnable parameters. They can be adjusted when ANN is learning based on the given training data so that the network can achieve the highest performance. To combine all the inputs received from other perceptrons of the preceding layer, a summation function is used which will collectively add each input proportional to its weight. Additionally, bias is added to the summation function so that it can shift the activation function to the left or right to better fit to the given data.

Activation Function

Activation function is a function that transforms a given value to a desired-range output. This transformation is important because it controls if an output should be activated i.e whether the output should be passed to the next neuron. Hence, choosing an activation function is important which directly influences the outcome of the network.

Currently, there are many activation functions being used including linear and non-linear ones. In the machine learning field, non-linear activation functions are more often used because they add the non-linear property to a neural network which helps the network be able to perform complicated tasks. Sigmoid function and rectified linear unit (ReLU) function are two of the most commonly used non-linear activation function where sigmoid squashes an input to a range from 0 to 1 and ReLU retains inputs whose value is bigger than 0 and squashes inputs that are smaller than 0 to 0.

Neural Network Architecture A neural network architecture comprises three main parts: an input layer, a set of multiple hidden layers and an output layer. Neurons are arranged into these layers.

A neural network that has only one hidden layer is called a shallow neural network. Meanwhile, a deep neural network is a network that has two or more hidden layers.

One of the common issues with training models that have many hidden layers is over-fitting. This issue refers to models that excessively fit only to their training data but do not fit well to unseen data. Hence, these models would yield extremely good results in the training phase but perform poorly in the testing phase. To avoid this case, dropout is used. This technique will randomly omit some neurons of a model while training so that neurons are forced to not be too dependent on others.

2.2.3.10 | CNN

Convolutional Neural Network (CNN) [5] is one of the deep learning neural networks that is used for object detection and natural language processing. The CNN architecture has three main types of layer: convolutional layer, pooling layer and fully-connected layer.

Convolutional layers extract features from images, pooling layers picks up features from inputs of the preceding convolutional layer while fully-connected layers are used to classify data.

In CNN, convolutional layers are the most important layers. To extract features, a convolutional layer uses a set of filters (i.e. kernels) sliding over a given input to create a stack of feature maps with depths equal to the number of filters.

A pooling layer, also known as a sub-sampling layer, receives the output of the previous convolutional layer, tries to retain important information from the given input while reducing the spatial size of it in order to decrease the number of parameters used in a network. The reduction in the number of parameters will help prevent the overfitting scenario. A pooling layer works similarly as a convolutional layer by applying a matrix on an input, but the difference is that a matrix is only applied on a distinct region. There are pooling operations such as max pooling and average pooling. Max pooling operation is more widely used than other operations. This operation only takes the maximum value of the region that a matrix is applied to, thus, helps retain key features.

2.3 | Requirements Classification

Requirement classification is one of the four phases of the requirements elicitation process [1]. This phase is crucial because it groups relevant requirements into coherent clusters, provides a guideline to other activities in the requirements elicitation phase and the requirement validation phase. Such a guideline can be used to check the requirements coverage and requirement relevance on different aspects of a system (i.e. check if necessary requirements are missing or inconsistent to others). Functional requirements categorization and non-functional requirements categorization will be introduced Section 2.3.1 and Section 2.3.2 while Section 2.3.3 will discuss about the recent applications of machine learning methods on the requirement classification problem according to our literature review "The Landscape of Machine Learning in Requirements Engineering".

2.3.1 | Functional Requirements Classification

Currently, there are different ways to categorize functional requirements. In the paper [6], functional requirements were categorized into five categories including data input, data output, data validation, business logic, data persistence, communication, event trigger, user interface, user interface navigation, user interface logic, event rigger, external call and external behavior.

Koelsch [7] listed possible types of functional requirements in his book *Requirements Writing for System Engineering* which are business rules, transaction corrections adjustments and cancellations, administrative functions, authentication, authorization levels, audit tracking, external interfaces, certification requirements, searching reporting requirements, historical data, archiving, compliance legal or regulatory requirements, structural, algorithms, database, power, network, infrastructure, backup and recovery. Meanwhile, solutions, enablement, action constraints, attribute constraints, definitions, or policy requirements were categories to classify functional requirements listed in the paper of Jain, Verma, Kass, *et al.* [8] based on the sentence structure of functional requirements.

2.3.2 | Non-functional Requirements Classification

Non-functional requirements are categorized based on quality attributes [2]. There are various quality attributes, some of the main types of non-functional requirements based on quality attributes are scalability, reliability, availability, maintainability, security, etc. In this study, we choose 11 types of non-functional requirements based on the selected types of non-functional requirements of the PROMISE dataset [3]. These types include availability, fault tolerance, legal, look and feel, maintainability, operational, performance, portability, scalability, security and usability.

2.3.3 | ML Requirements Classification

In real life, requirement classification is usually done manually. However, this step can be tedious, time-consuming and inaccurate due to different reasons. Hence, recently, researchers have tried to apply machine learning on this task to automate and increase the efficiency of it. Four selected papers listed below are papers found from our literature review which identifies studies on the application of machine learning in the area of Requirements Engineering in the last five years from 2015 to 2020 published in three digital libraries: IEEE Xplore, ScienceDirect and Web of Science.

Rahimi, Eassa, and Elrefaei [9] proposed an ensemble machine learning method for functional requirements classification into six classes consisting of solution, enablement, action constraints, attribute constraints, definitions and policy. This method combined 5 different models including Naive Bayes, SVM, Decision Tree, Logistic Regression, Support Vector Classification to form the proposed ensemble model. After conducting different experiments, it was found that CountVectorizer achieved a better performance than TF-IDF in all aspects and for all classifiers in the classification task. The proposed ensemble approach with the three best classifiers even though gained the same accuracy as the proposed one with all 5 models, achieved a slight improvement in time. It was also concluded that the proposed approach obtained the highest results (99.45%) compared with other ensemble methods for classifying functional requirements.

Kurtanović and Maalej [10] presented a study for automatically classifying non-functional requirements into categories including usability, security, operational and performance using SVM. Employing only word features, non-functional requirements binary classifiers achieved the precision and recall ranging between 72% and 93% while selecting 200 most informative features using automatic feature selection, the binary classifiers obtained more than 70% precision and recall on four different categories.

Haque, Abdur Rahman, and Siddik [11] presented an empirical study that combined 7 machine learning techniques including Multinomial Naive Bayes (MNB), Gaussian Naive Bayes (GNB), Bernoulli Naive Bayes (BNB), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Stochastic Gradient Descent SVM and Decision Tree with 4 feature extraction approaches namely Bag of Words (BoW), Term Frequency Inverse Document Frequency (TF-IDF) (character level), TF-IDF (word level), TF-IDF (n-gram) techniques and examined which pair performed the best in classifying non-functional requirements. It was found that Stochastic Gradient Descent SVM obtained the highest precision, recall and F1 score regardless of feature extraction methods and the pair Stochastic Gradient Descent SVM and TF-IDF was the best combination for non-functional requirements classification among other pairs in the study.

Baker, Deng, Chakraborty, *et al.* [12] compared two different neural network models namely an artificial neural network and a convolutional neural network in a classifying non-functional requirements into five categories including maintainability, operability, performance, security and usability task. The ANN model was trained on 4 classes including operability, performance, security, usability with the number of training samples cut to half. On the other hand, the authors trained the CNN with an entire dataset with a full number of security requirements training samples. It was found that both CNN and ANN achieved high results on precision, recall and F-score, but the CNN model performed better than the ANN in Performance and Security classes (10% and 9% respectively).

3 Research Methodology

This chapter explains the structure of this project based on the framework CRISP-DM. Some introduction about CRISP-DM is first given. The rest of this chapter will show how the phases and steps of CRISP-DM are adapted.

Introduced in 1999 by Chapman, Clinton, Kerber, *et al.* [13], Cross Industry Standard Process for Data Mining (CRISP-DM) - a well-proven data-mining model is used to structure this study. This framework comprises six different phases including business understanding, data understanding, data preparation, modelling, evaluation, deployment as shown in the Figure 3.1. Each phase will have corresponding tasks to help explore more about the phase.

These phases represent the life cycle of a data mining project. The previous phase will base on its results and decide which phase or a task of a phase will perform next. Dependency relationships between these phases are shown in the inner arrows.

The cyclical nature of a data mining project is shown in the outer circle. The accomplishment of a data-mining project is not determined by finishing these phases from business understanding to deployment at one time since a new process can be triggered after new business questions are formed based on lessons and solutions learned from previous phases' results.

3.1 | Business Understanding

Objectives of the study, seen from a business perspective, are presented in this phase through smaller steps including:

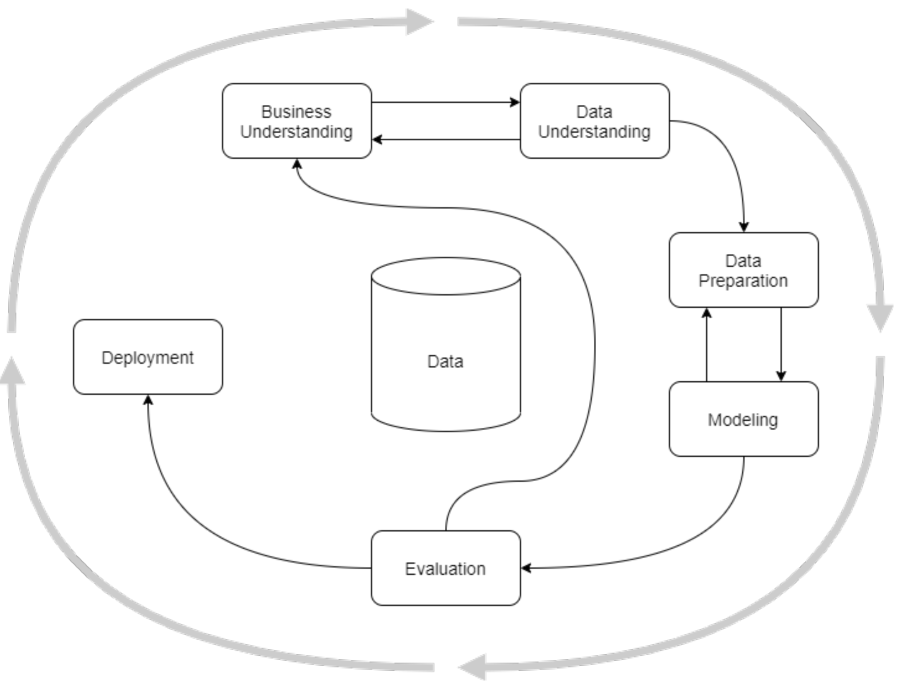

Figure 3.1: Overview of the CRISP-DM framework

- Determine Business Objectives

  As the name implies, this task's main purpose is to decide the business objectives for the study/project and establish criteria for what a successful outcome of a study/project is.

- Assess Situation

  This task requires to give more details about factors (e.g. resources, requirements, constraints, assumptions, etc.) that could determine goals and plans of a study.

- Determine Data Mining Goals

  Instead of identifying business objectives, this task points out objectives and criteria that are seen from a technical perspective.

- Produce Project Plan

  A detailed intended plan for how to attain the study's goals will be proposed in this step.

3.2 | Data Understanding

This phase is to get familiar with data used in the study through carrying tasks:

- Collect Initial Data

  This task is to gather data from source(s) for the project.

- Describe Data

  Collected data from the previous task is examined and described in this task.

- Explore Data

  The main purpose of this task is to deepen understanding on collected data through data mining questions.

- Verify data quality

  After exploring the data, the quality of data will be verified and reported in this task.

3.3 | Data Preparation

As its name suggests, after the data is explored, necessary steps will be taken to process initial raw data into those that could be used in the study. These steps are:

- Select Data

  Data will be filtered based on criteria such as goals, quality, constraints for the next phase.

- Clean Data

  In this task, selected data will be processed to improve the quality of the data.

- Construct Data

  Attributes that can be derived from the data will be extracted in this task.

- Integrate Data

  This task's main purpose is to combine data from different forms into an unified form so that it can be handled conveniently.

- Format Data

  In this task, necessary modifications, mostly syntactic ones, are made to the data so that it can be used by modelling tools.

3.4 | Modelling

In this phase, different models are explored, selected and tuned to achieve the optimal results. Several tasks should be done in this phase including

- Select Modeling Techniques

  In this task, modeling techniques will be specifically chosen to achieve goals defined in the business understanding phase.

- Generate Test Design

  The concern of this task is to design tests that can evaluate the performance and the validity of models.

- Build Model

  As its name suggests, this task is to create models on prepared data.

- Assess Model

  Models created in the previous task are interpreted in this task depending on criteria and test design, then, are evaluated to check their quality and their generality.

3.5 | Evaluation

This phase is to evaluate and compare the performance of the developed models to see if goals that are introduced in the business understanding phase are achieved. The evaluation will include

- Evaluate Results

  Results produced by models will be evaluated in this task to see if they meet the business objectives or discover the reasons leading to the deficiency of models.

- Review Process

  At this point, the whole process will be reviewed to ensure that no activities are overlooked and there are no quality assurance issues.

- Determine Next Steps

  In this task, based on the outcome of the two previous tasks, some decisions will be made to the current project (e.g. pass on to the deployment phase, setup iterations, establish new projects/studies).

3.6 | Deployment

Models after being developed and evaluated will be deployed to be used by customers in this phase. This phase comprises multiple steps including:

- Plan Deployment

  Deployment strategies will be planned in this task depending on the results in previous phase - evaluation.

- Plan Monitoring and Maintenance

  Detailed plan of actions on how to maintain and monitor if the project is deployed is prepared and designed in this task.

- Produce Final Report

  All information related to the project will be documented in a final report.

- Review Project

  Aspects of the project and acquired experience when conducting the project will be reviewed and discussed at this stage.

3.7 | The Structure Mapping

As mentioned above, the CRISP-DM framework is used to guide this study. This thesis report is structured similarly to the structure of the framework. Business understanding will be presented in the Chapter 1 while data understanding and data preparation phases will be discussed in Chapter 4. Chapter 5 will explain the Modelling phase through experiments conducted in the project while the results produced by models modeled in Chapter 5 will be used in the Evaluation phase - described in Chapter 6. The deployment phase is out of the scope and hence it will not be discussed in this report.

4

Dataset

This chapter describes the data used in our project as well as the process of transforming raw data into data that were used as inputs of machine learning methods. The insight of the used data is discussed in Section 4.1, while the transformation process is presented in Section 4.2.

4.1 | Data Understanding

The final dataset contains 1838 requirements from 13 different datasets collected from different sources [3], [14]–[18] and in different formats. 1034 out of 1838 requirements are functional requirements while the remaining (803 requirements) are non-functional requirements. Table 4.1 shows an overview of the number of requirements originated from different datasets that are used in this project. PROMISE, SecReq, Dronology, Leeds, ReqView, Wasp datasets are available in CSV format while the rest are in the form of requirement specification documents intended for human reading. Collecting data from different sources ensures that our dataset is context-free and diverse, which will help increasing the performance of models when they encounter new data.

Table 4.1: Dataset Summary

Source	Dataset Id	Project Id	Req. Set Name	# Req.	# FR	# NR
[3]	1	1 to 15	PROMISE	625	253	372
[19]	2	1 to 3	SecReq	483	211	271

(continued on the next page)

Table 4.1: *(continued)* Dataset Summary

Source	Dataset Id	Project Id	Req. Set Name	# Req.	# FR	# NR
[20]	3	1	Dronology	97	94	3
		2	Leeds	85	47	38
		3	ReqView	87	77	10
		4	WASP	62	58	4
[18]	4	1	Inspire	28	16	12
[15]	5	1	CCTNS	63	3	66
		2	Gamma	38	12	26
		3	Inventory	12	6	6
		4	Themas	24	24	0
		5	Multi-mahjong	30	30	0
		6	TCS	204	203	1
Total requirements				**1838**	1034	803

As shown in the Table 4.1, the PROMISE [3] dataset accounted for the most requirements of the final dataset, consisting of 625 requirements (255 functional requirements and 255 non-functional requirements) collected from 15 different projects. The CSV file of this dataset comprises three columns, namely ProjectID, RequirementText, Class. ProjectID is an identification number for each project belonging to a dataset. RequirementText contains the textual content of a requirement. Class presents the category that a requirement belongs to. There are 12 classes in the PROMISE dataset (11 classes for non-functional requirements denoted similarly in the Table 4.2 and one class for functional requirements denoted as F). Some data samples of the PROMISE dataset are shown in the Listing 4.1.

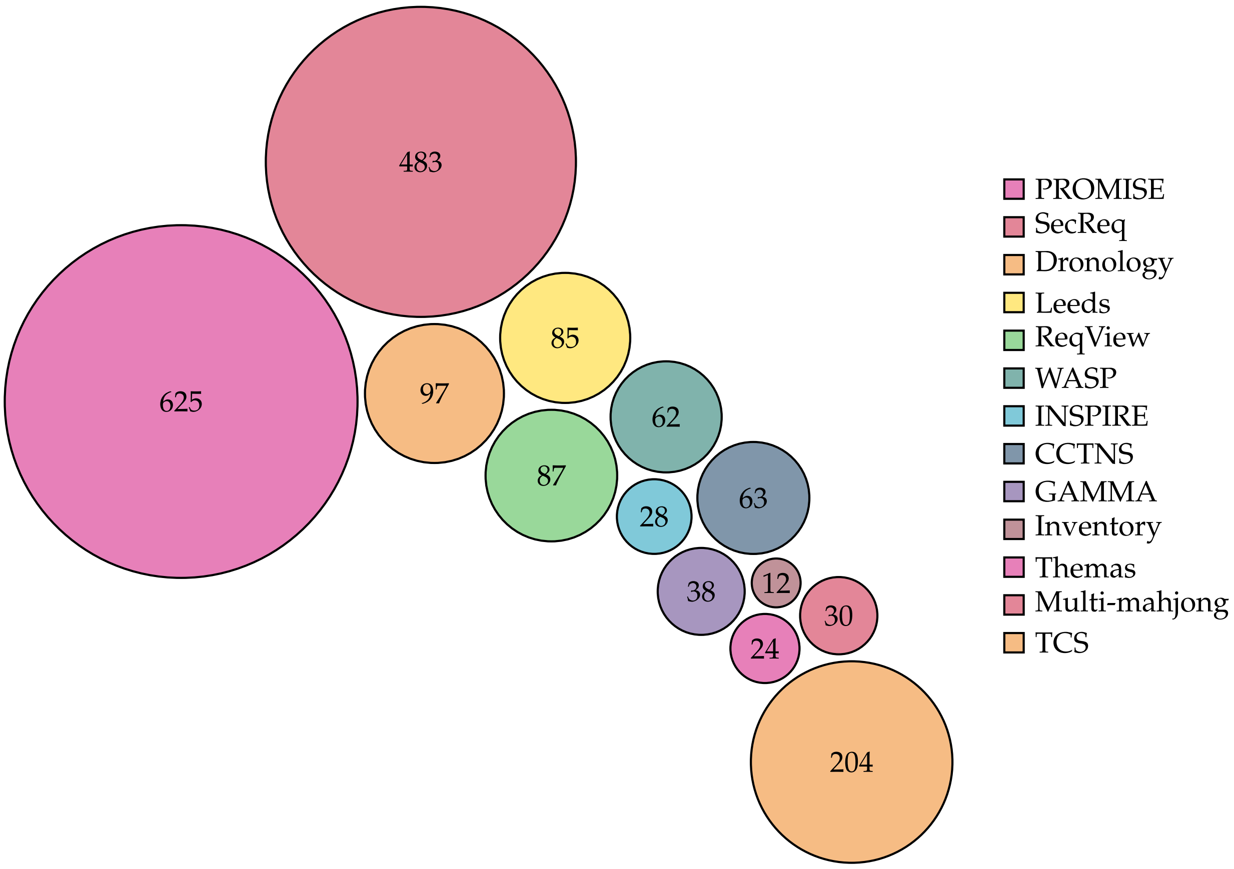

Figure 4.1: Proportion of different datasets

Listing 4.1: PROMISE CSV Data Samples

```
ProjectID,RequirementText,class
1,'The system shall filter data by: Venues and Key Events.',F
2,'The product shall be easy for a realtor to learn.',US
12,'The product shall continue to operate during upgrade  change or new resource
↪  addition.The product shall be able to continue to operate with no interruption
↪  in service due to new resource additions.',MN
```

The SecReq [19] dataset has 483 requirements with 185 requirements labeled as security requirements and the rest labeled as non-security requirements, making SecReq the second-highest requirement contributor to our final dataset. The collected CSV file has three columns namely ProjectID, RequirementText, IsSecurity. The IsSecurity column value can either be 1 or 0, denoting if a requirement is a security requirement or not respectively. Some data samples of the SecReq dataset are shown in the Listing 4.2.

Listing 4.2: SeqReq CSV Data Sample

```
ProjectID,RequirementText,IsSecurity
1,"The CNG shall implement an authorization management handling policy.",1
1,"The CNG shall support mechanisms to authenticate itself to the NGN for
↪  connectivity purposes.",1
2,"Payment for a transaction is only required when a detail transaction is submitted
↪  for payment. A merchant acquirer may choose to pay the merchant using the batch
↪  total",0
3,"The run time environment shall responsible to establish communication services
↪  between card and off-card entities",0
3,"Security domain shall ensure complete seperation of keys among the card issuers
↪  and other application providers",1
```

For convenience, the datasets Dronology, Leeds, ReqView, WASP [14], [16], [17], [21] are taken from the work of Dalpiaz, Dell'Anna, Aydemir, *et al.* [20], instead of the original sources. They processed and formatted each dataset into a unified format (CSV) which is arguably more convenient than manually extracting requirements from textual specification documents. These datasets contain four columns of data namely ProjectID, RequirementText, IsFunctional, and IsQuality. The paper [20] adopted the requirement categorization of Li, Horkoff, Mylopoulos, *et al.* [22] which allows a requirement to be functional or quality (non-functional) or both at the same time. The IsFunctional and IsQuality columns are intended for the labelling requirements based on the adopted categorization. Some data samples of these datasets are shown in the Listing 4.3.

Listing 4.3: Dalpiaz, Dell'Anna, Aydemir, *et al.* [20] CSV Data Samples
This dataset contains multiple files from different projects. Each of these fragments shows some samples of the data contained in each file in this dataset. From top to bottom, the names of these project are: Dronology, Leeds, ReqView, and WASP.

```
ProjectID,RequirementText,IsFunctional,IsQuality
1,The system must offer customisable metadata schema.,0,1
1,The system must offer customisable workflow to import or create metadata and
↪  upload associated files and support multiple ingest protocols e.g. SWORD2,1,1
```

```
1,"The MapComponent shall support different types of map layers (e.g.  terrain
↪  satellite)",1,1
1,"The MissionPlanner shall execute flight plans for multiple UAVs concurrently.",1,1
1,"The GCS shall transmit the UAV's properties to the GCSMiddleware",1,0
```

```
1,User shall be able to import a table from MS Excel,1,1
1,User shall be able to use the application without installation of any additional
↪  SW except the web browser,0,1
```

```
1,The WASP platform must provide services that may be used by a WASP application to
↪  charge the user for using one of his services.,1,0
1,The WASP platform must allow end-users to provide profile and context information
↪  explicitly to applications or the platform.,1,0
```

The Inspire, CCTNS, Gamma, Inventory, Themas, Multi-mahjong, and TCS [15], [18] datasets are presented as human readable specification documents in the form of PDF files. Since these files are intended for human viewing, some extra steps are required to transform them into a computer-friendly format. This is done by extracting the requirement text and their corresponding classes to a CSV file.

The reason for using the CSV format is that it is widely used in both industry and scientific research. There are a lot of extensions and libraries supporting extracting and inserting data from/to CSV files. Furthermore, some of the datasets are already processed and stored in CSV files as mentioned above hence it will be more convenient to use CSV as the file format to store data.

For any case (i.e. datasets that are already in CSV format or datasets that have to be transformed to a more computer-friendly format), all of the datasets listed above need to be partially or fully relabeled so that they can be used for this study. Hence, after collecting all necessary data, we proceeded with the labelling task.

There are different requirement categorizations that are discussed in the Section 2.3. In this study, we adopted the functional requirements categorization introduced by Jain, Verma, Kass, *et al.* [8]. Figure 4.2 presents an overview of this categorization. Actions that a system should perform will be described through solution requirements. Enablement requirements are those that report abilities of a system offering to its users.

Action constraint requirements describe constraints on a system's actions/behaviours while attribute constraint requirements show constraints on attributes or attribute values. A system has two kind of entities: agent and non-agent. Agents are entities that can perform actions while non-agents cannot. Non-agents are defined in definition requirements. Lastly, policy requirements are those that name policies a system or a solution of that system must adhere to.

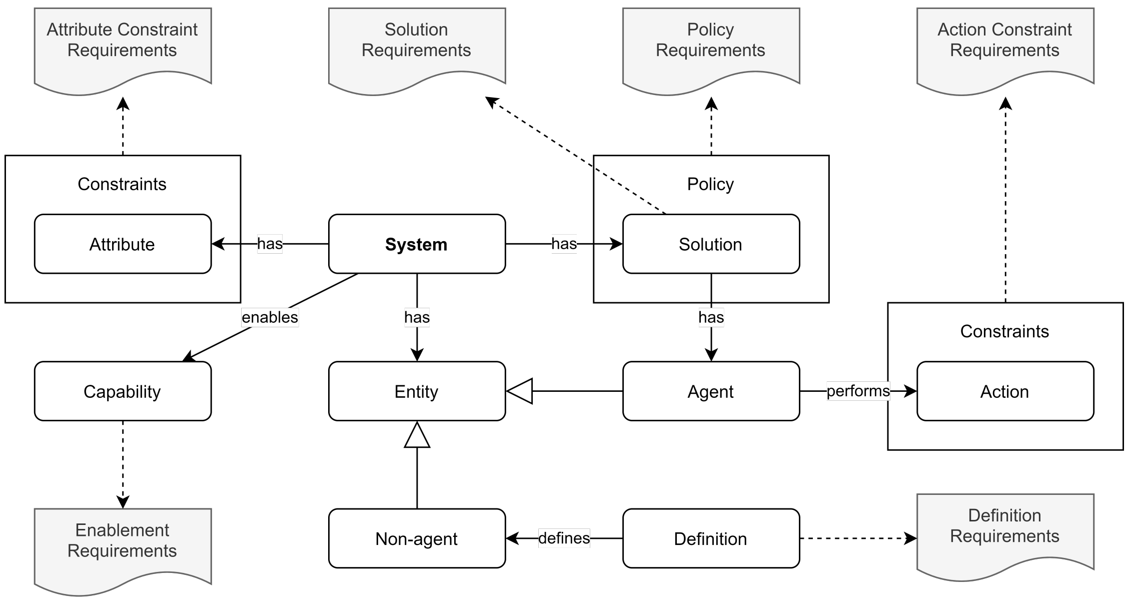

Figure 4.2: An overview of the categories of functional requirements

For non-functional requirements classification, we adopt the types of non-functional requirements of the PROMISE dataset [3] as discussed in the Section 2.3.2

In the total of 1838 requirements, we manually labeled 1181 requirements into 17 classes (11 sub-classes of non-functional requirement and six sub-classes of functional requirement). These classes are denoted by a distinct capital letter or a pair of capital letters as shown in Table 4.2. Thus, in the final file, a column to determine whether a requirement is functional or non-functional is unnecessary and will be intentionally left out since this information can be derived from the Class column.

As shown in the Table 4.2, our dataset is imbalanced. Even though the difference in the number of functional and non-functional requirements is insignificant, the distribution between their sub-classes is heavily unequal as the number of Definition (DE), Policy (PL), Fault Tolerance (FT), and Portability (PO) requirements is much less than other classes. This distribution also reflects the chance of encountering these requirements in practice. Security, operation, and usability are usually considered important quality attributes of a system, hence security requirements, operational requirements, and usability requirements appear more often in software requirements specifications. Similarly, for functional requirements, solution requirements and enablement requirements are more commonly encountered than others because they help specify the functionalities of a system as well as what a system can offer to its user. This imbalance can negatively affect the performance of ML methods because it is hard for a ML method to learn to differentiate the characteristics of the classes whose information are limited by the low number of requirements.

Table 4.2: Requirement Types and Classes Distribution Overview

Type	ID	Class	Count
Non-functional	1	[A] Availability	29
	2	[FT] Fault Tolerance	12
	3	[L] Legal	20
	4	[LF] Look & Feel	44
	5	[MN] Maintenance	26
	6	[O] Operational	170
	7	[PE] Performance	74
	8	[PO] Portability	12
	9	[SC] Scalability	24
	10	[SE] Security	262
	11	[US] Usability	131
Functional	12	[AC] Action Constraint	178
	13	[AT] Attribute Constraint	64
	14	[EN] Enablement	348
	15	[DE] Definition	9

(continued on the next page)

Table 4.2: *(continued)* Requirement Types and Classes Distribution Overview

Type	ID	Class	Count
	16	[PL] Policy	15
	17	[SO] Solution	420

4.2 | Data Preparation

4.2.1 | Data Integration

In this integration step, all of the collected data are transformed into a unified format and then are combined in the final file. The final file has a similar structure to the CSV file of the PROMISE dataset with four columns including ProjectID, RequirementText, Class, and an extra column called DatasetID. DatasetID is a unique ID number that identifies each dataset which are numbered from 1 to 5 as shown in the Table 4.1. A dataset obtained from one source will be denoted with the same DatasetID. CCTNS, Gamma, Inventory, Themas, Multi-mahjong, and TCS are denoted with the same DatasetID because they are requirement specifications collected from the work of researchers from Formal Methods and Tools Group (FMT) of Institute of Information Science and Technologies "Alessandro Faedo" under the name Natural Language Requirements Dataset, they are denoted with the same DatasetID. Dronology, Leeds, ReqView, and Wasp are also considered to be one dataset since we use datasets' CSV files taken from the work of Dalpiaz, Dell'Anna, Aydemir, *et al.* [20]. ProjectID, RequirementText, and Class have the same meaning as explained in the Section 4.1.

Since data were collected from different sources, reformatting data was necessary and hence was proceeded so that when being combined, they all had the same style and can be easily processed in the next step. Steps taken to reformat data include escaping special characters with backslashes, using single quotation mark to delimit a string, removing unrelated or unnecessary columns to the study from datasets' CSV files (e.g. isSecurity, isFunctional, isQuality), adding extra desired columns according to the final data file structure (e.g. DatasetID, Class).

Listing 4.4: Final CSV Data Samples

```
DatasetID,ProjectID,RequirementText,Class
1,1,'The product shall be available during normal business hours. As long as the
↪  user has access to the client PC the system will be available 99\% of the time
↪  during the first six months of operation.',A
1,2,'The look and feel of the system shall conform to the user interface standards
↪  of the smart device.',LF
2,3,'The OPEN shall responsible to load application code; card content and memory
↪  management',SO
5,6,'The TCS shall be capable of operating continuously in functional Operation Mode
↪  for a minimum of 72 hours.',PE
```

4.2.2 | Data Pre-processing

Some feature extraction techniques require the data to be processed. In total, there are four pre-processing steps performed on raw data (i.e. requirement text). These steps include lower casing, punctuation removal, stop-words removal, lemmatization, utilizing the Natural Language Toolkit (NLTK) Python library [23] With this process, first, all the words of a requirement are converted into lowercase. Then, punctuation and stop-words (i.e. common words in a language, in this case, English) are removed since they have low semantic values. Words of these text are transformed into their based form to normalize the data. This transformation process is called lemmatization.

Besides pre-processing requirement texts, categorical data (i.e. Class of requirements) is encoded to number in order to be used by machine learning models. The numerical value corresponding to each class is shown in the Table 4.2.

4.2.3 | Feature Extraction

Feature extraction is the process of transforming raw data into features. Features are information of interest that can be used for further analysis. In this section, we discuss some of the feature extraction techniques that are used in the experiments carried out in our project which are presented in the recent related works [9]–[12]. In these works, one or multiple feature extraction techniques are used in combination with a machine learning technique, which will be discussed more in detail in Chapter 5.

4.2.3.1 | BoW

Bag-of-Words is a common feature extraction technique that counts the number of times a word appears in a text (in this case, requirement text). For this technique, ngram_range is an important parameter that decides the type of n-grams to be extracted.

Besides words, word n-grams - a sequence of n adjacent words is often used to identify multi-word expressions (e.g. nice—weather, I—like—rain, rain—cat—dog). 1-gram (unigram) is a sequence of 1 word (i.e. word), 2-gram (bi-gram) is a sequence of 2 adjacent words, etc. For BoW, ngram_range is set to (1,1) which means only words are extracted. To change from BoW to Bag-of-ngrams, ngram_range value is adjusted. For Bag of Trigrams, ngram_range is set to (3, 3). For Bag of Unigram, Bigram and Trigram, ngram_range equals (1, 3). As n-grams increases, the information that a model receives increases and the vocabulary increases, too.

The BoW implementation is obtained by the CountVectorizer provided by Scikit-learn [24]. To extract features, the training data is first fitted and transformed by the constructed CountVectorizer. This fitting helps the defined instance learn the vocabulary dictionary from the training data. Then, this CountVectorizer is used to transform the testing data into a document-term matrix so that they can be understood by machine learning methods. The testing data is not fitted to the CountVectorizer because it can leak the information of the testing data to the machine learning methods which affects the validity of the model testing step.

4.2.3.2 | TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) [25] is an alternative technique to BoW. This technique calculates two things: the frequency of a term in a document and the importance of that term in the document, then multiplies them for every single term in a document. Words that have higher scores are less common, and more relevant to the document, thus will be kept in output vectors. Commonly used words such as a, an, the usually have lower scores (i.e. less important) hence can be removed out of output vectors.

The Tf-Idf implementation is obtained by the TfidfVectorizer provided by Scikit-learn. For TfidfVectorizer, ngram_range and analyzer are two important parameters that affect the output feature. *analyzer* determines the output feature are made of words or characters while ngram_range controls the values of n-grams (e.g. (1,1) - only unigrams, (1,3) - unigrams, bigrams and trigrams).

The normal process to extract features using TfidfVectorizer is to fit and transform a training set into something that ML methods are capable of processing using the learned TfidfVectorizer to transform a testing set. As mentioned above, the reason for only fitting to training data, not to both training and testing set is to avoid data leakage which could lead to a wrong performance assessment of a model.

4.2.3.3 | Part-of-Speech N-Grams

Part of Speech (POS) is a grammatical category for words (e.g. noun, verb, adjective, adverb, modal verb). POS N-Grams is a feature extraction technique that gets a sequence of n adjacent tags. For example: POS 1-gram (unigram) is a sequence of 1 POS tag (e.g. MD (i.e. modal), NN (i.e. noun), RB (i.e. adverb)), POS 2-gram (bi-gram) is a sequence of 2 adjacent POS tags (e.g. VB-VBN (i.e. verb base form-verb past particle) (e.g. from a bi-gram "be reviewed")), POS 3-gram (tri-gram) is a sequence of three POS tags (e.g. VB-VBN-IN (i.e. verb base form-verb past participle-preposition) (e.g. from a tri-gram "be reviewed of")). According to the Penn Treebank corpus [26], there are 36 POS tags excluding punctuation.

Similar to BoW and Tf-Idf, ngram_range is an important and adjustable parameter. This feature extraction technique is performed before the stopwords removal and lemmatization steps because removing stop words alters the structure of sentences which also affects the information retrieved from the extracted feature.

Because this technique has not been provided by Python libraries as well as the implementation of this technique is not publicly available, we re-implemented it based on the explanation of the authors Kurtanović and Maalej [10]. Our re-implementation of this technique is shown in the Appendix A.

4.2.3.4 | Textual Features

Besides the above features, the paper [10] uses additional textual features to train their models. These textual features include the fraction of nouns, verbs, adjectives, adverbs and modal verbs, text length, the height of the syntax tree and the number of subtrees of the syntax tree. Those features are also performed before the stopwords removal and lemmatization steps because these steps change the number of part-of-speech which could affect the values of these features. The height of the syntax tree and the number of subtrees of the syntax tree are derived using NLTK [23].

Besides the listed features above, CP unigrams - one of the features used in the paper [10] is not used in our study. The description of the paper's authors about this feature - "unigrams of part of speech (POS) tags on the clause and phrase level (CP)" is brief and incomplete so that we cannot fully recreate these features.

5

Experiments

This chapter describes the experiments that have been conducted during the project to answer the research questions defined in Section 1.3.

5.1 | Experiment Setup

Setups which were used throughout experiments conducted in this project are discussed in this section. Hardware and software specifications will be covered in the Section 5.1.1 and Section 5.1.2. Dataset used in the experiments will be reported in the Section 5.1.3. The Section 5.1.4 and Section 5.1.5 describe feature extraction techniques and machine learning techniques setups while the combination of feature extraction techniques and ML techniques are listed in the Section 5.1.6 to create ML methods following the processes of selected papers.

5.1.1 | Hardware Specification

The experimentation is carried out on a system with the configurations as below:

- **CPU:** Intel(R) Core(TM) i7-7820HQ CPU @ 2.90GHz
- **GPU:** Intel(R) HD Graphic 630
- **RAM:** 32GB
- **OS:** Windows 10 Pro x64

5.1.2 | Software Specification

All experiments in this chapter are implemented using Python 3.6.13. Python libraries namely Scikit-learn, Pytorch are used to build machine learning models. Scikit-learn provides various tools for machine learning modeling ranging from data pre-processing, data extraction and selection to machine learning model building and model evaluation. Pytorch is a framework developed by Facebook which helps create deep learning networks. NLTK and Pandas are two Python packages that are used to process data where Pandas supports structuring data and NLTK helps with pre-processing data in the domain of natural language processing. All of the experiments are programmed and built using PyCharm IDE.

5.1.3 | Data Setup

The dataset used in the experiment is the same as mentioned in the Chapter 4. Each method is trained with three different types of requirements taken from the dataset including functional requirements, non-functional requirements, mixed-type of requirements (functional and non-functional requirements).

Some of the selected papers only show the results of ML methods in classifying requirements of top X classes in term of requirement counts, while others trained their ML algorithms to classify requirements of all classes. Hence, in our experiments, our ML methods are trained on two different sets of inputs containing requirements of either significant classes or all classes to see the differences in the results of two different sets.

The definition of a significant class varies from paper to paper. Thus, in this project, a class is considered to be significant if it contains a number of requirements greater or equal the average number of requirements per class, which is calculated as:

$$\text{Average number of req. per class} = \frac{\text{Total number of req.}}{\text{Number of classes}} \tag{5.1}$$

Applying this formula to the requirement counts of the classes shown in Table 4.2, the average number of requirements for each functional requirement class is 172 requirements and the average number of requirements for each non-functional requirement class is 73 requirements. Hence, based on our dataset, functional requirement classes that are considered significant are Action Constrain, Enablement and Solution. Meanwhile, Non-functional requirement classes that are considered significant are Look & Feel, Operational, Performance, Security and Usability.

5.1.4 | Feature Extraction Technique Setups

This section lists and describes all of the feature extraction techniques setups that are used in our project. One feature extraction technique setup could be used with multiple machine learning methods, hence, each setup is set with an id for better organization.

5.1.4.1 | FE1: Term frequency – Inverse document frequency (Tf-Idf) Word Level

For the feature extraction technique, since the authors [9], [11] did not bring up the configuration of Tf-Idf, the values of the parameters of Tf-Idf are set as default (Tf-Idf Word Level with ngram_range (1, 1)).

The defined TfidfVectorizer then fits to the training data (i.e. learns vocabulary of the training data and calculates the idf) and transforms them into a document-term matrix. For the testing set, TfIdfVectorizer transforms it based on its knowledge of the training data.

5.1.4.2 | FE2: Term frequency – Inverse document frequency (Tf-Idf) N-Grams

Tf-Idf N-Grams is one of the feature used in the paper [11]. Because the author did not mention the used values of parameters of Tf-Idf, parameters are left as default. However, for Tf-Idf N-Grams, letting the Tf-Idf technique with ngram_range value as default (i.e. 1, 1) will generate the same result as the Tf-Idf word level - which is mentioned in Section 5.1.4.1, hence this parameter is assigned a new value (1, 4). The defined TfidfVectorizer is fit with the training data and transforms it to a document-term matrix. Then, the testing set is transformed by the learned TfidfVectorizer.

5.1.4.3 | FE3: Bag of Words (BoW)

The BoW technique is set up with parameters are set default since [9], [11] did not report the used parameters' values. The defined CountVectorizer fits to the training set (i.e. learn a vocabulary dictionary of the training set) and transforms the training set into a document-term matrix. The CountVectorizer also transforms the testing set into a document-term matrix based on the learned vocabulary dictionary of the training data.

5.1.4.4 | FE4: Bag of Words (BoW)

According to the paper [12], some parameters of CountVectorizer are set including:

- 'analyzer' : 'word'

- ngram_range : (1, 1)

The defined CountVectorizer fits to the training set (i.e. learn a vocabulary dictionary of the training set) and transforms the training set into a document-term matrix. The CountVectorizer then transforms the testing set into a document-term matrix based on the learned vocabulary dictionary of the training data.

5.1.4.5 | FE5: Multiple Techniques

As discussed in the Section 4.2.3.4 and Section 4.2.3.3, some of features used to train the SVM in the original paper [10] retrieved before the stopwords removal and lemmatization steps including POS ngrams, %noun, %verb, %adjective, %adverb, %modal verb, text length, the syntax tree height and the subtrees count. Only N-Grams feature is drawn out after the pre-processing is finished.

Based on the original paper, for N-Grams and POS ngrams, their ngram_range was set at (1, 3). As mentioned in the Section 4.2.3.1, these two techniques are also followed by the same process which first defines an instance, uses this instance to fit and transform the training data, and last applies its knowledge of the training data to transform the testing data.

5.1.5 | ML Technique Setups

The implementation of machine learning techniques that are used in the experiments as well as their parameters will be described in this section.

5.1.5.1 | ML1: Ensemble Method

Based on the explanation of Rahimi, Eassa, and Elrefaei [9] in their paper, the proposed ensemble method combines three different base models, in particular, SVM, Linear SVC and Logistic Regression. The authors did not explicitly mention the parameters used for these techniques but did mention the default value of these parameters set by the used library [24].

As described in the Section 2.2, a weighted ensemble method evaluates the performance of each model and calculates a weight to each model. The final prediction of this ensemble method is based on the contribution of each base model which is decided by the weight. In this paper, Rahimi, Eassa, and Elrefaei [9] proposed a new method to calculate the weights for base models. Since the implementation of this proposed method is not publicly available, we re-implement the method based on the explanation of the algorithm in the original paper [9]. The Python code of our re-implementation as well as the setting for the ensemble classifier can be found in the Appendix A.

5.1.5.2 | ML2: Multinomial Naive Bayesian

Because Haque, Abdur Rahman, and Siddik [11] did not describe parameters' values that are set for this algorithm, we assume that parameters used in their algorithm are left default. Hence, the initial setup of this method uses the default values which are set by Scikit-learn [24]that we use to build this algorithm.

5.1.5.3 | ML3: Gaussian Naive Bayesian

Because Haque, Abdur Rahman, and Siddik [11] did not describe parameters' values that are set for this algorithm, we assume that parameters used in their algorithm are left default. Hence the initial setup of this method uses the default values which are set by Scikit-learn [24]that we use to build this algorithm.

5.1.5.4 | ML4:Bernoulli Naive Bayesian

Because Haque, Abdur Rahman, and Siddik [11] did not describe parameters' values that are set for this algorithm, we assume that parameters used in their algorithm are left default. Hence the initial setup of this method uses the default values which are set by Scikit-learn [24]that we use to build this algorithm.

5.1.5.5 | ML5: K-Nearest Neighbors

Because Haque, Abdur Rahman, and Siddik [11] did not describe parameters' values that are set for this algorithm, we assume that parameters used in their algorithm are left default. Hence the initial setup of this method uses the default values which are set by Scikit-learn [24]that we use to build this algorithm.

5.1.5.6 | ML6: SVM

Because both papers [10], [11] did not describe parameters' values that are set for this algorithm, we assume that parameters used in their algorithm are left default. Hence the initial setup of this method uses the default values which are set by Scikit-learn [24]that we use to build this algorithm.

5.1.5.7 | ML7: Stochastic Gradient Descent SVM

Because Haque, Abdur Rahman, and Siddik [11] did not describe parameters' values that are set for this algorithm, we assume that parameters used in their algorithm are left default. Hence the initial setup of this method uses the default values which are set by Scikit-learn [24]that we use to build this algorithm.

5.1.5.8 | ML8: Decision Tree

Because Haque, Abdur Rahman, and Siddik [11] did not describe parameters' values that are set for this algorithm, we assume that parameters used in their algorithm are left default. Hence the initial setup of this method uses the default values which are set by Scikit-learn [24]that we use to build this algorithm.

5.1.5.9 | ML9: One vs Rest Classifier

Based on the paper [10], this strategy is set with the estimator Support Vector Machine. Other parameters of the estimator (i.e. SVM) are set default as in the setup of the used Python library since Kurtanović and Maalej [10] did not explicitly mention the values of these parameters.

5.1.5.10 | ML10: ANN

The architecture of this network [12] has one input layer, one hidden layer and one output layer. The number of neurons in the hidden layer is a tuning parameter. Meanwhile, the number of neurons of the input layers will depend on the size of the unique words in the training data and the number of neurons of the output layers will depend on the number of classes to classify. The Table 5.1 shows parameters and their values that are set in the original paper [12].

Table 5.1: Parameters set and explicitly mentioned by Baker, Deng, Chakraborty, *et al.* [12] for their ANN

Aspect	Parameter	Value
Architecture	Number of Hidden Layers	1
	Hidden Neurons	5

(continued on the next page)

Table 5.1: *(continued)* Parameters of ANN for the initial setup

Aspect	Parameter	Value
Training	Number of Epochs	50000
	Alpha	0.25

The code of the architecture of the ANN can be found in the Section A.3.

5.1.5.11 | ML11: CNN

The architecture of this network [12] has a word embedding layer followed by convolutional layers, max-pooling layers, a dropout layer and lastly a fully connected layer. The Table 5.2 shows parameters that are brought up and valued in the original paper [12].

Table 5.2: Parameters set and explicitly mentioned by Baker, Deng, Chakraborty, *et al.* [12] for their CNN

Aspect	Parameter	Value
Architecture	Number of Filters	32
	Embedding Dimension	150
Training	Optimizer	Adam GD
	Number of Epochs	250
	Batch Size	20

Besides the parameters that are mentioned in the paper, there are parameters that influence the performance of the network but are not brought up in the paper including filter sizes, learning rates, loss function. Due to this, these parameters will be set by us.

The chosen loss function for the network is Cross Entropy which is normally used in the multi-class classification. For filter sizes and learning rate, a lot of tests were performed to find the best values. Filter sizes were set to [1, 3, 4, 5] and learning rate was set to 0.01.

The code of the architecture of the CNN can be found in the Section A.4.

5.1.5.12 | ML12: XGBoost

The parameters of this algorithm are set default as in the setup of the used Python library.

5.1.5.13 | ML13: Random Forest

The parameters of this algorithm are set default as in the setup of the used Python library.

5.1.6 | Method Setups Overview

As mentioned above, this section presents combinations of a feature extraction technique and a machine learning technique used in our experiments which are shown in the Table 5.3. These combinations are similar to those in selected papers introduced in the Section 2.3.3 in order to create base ML methods for our research.

Table 5.3: An overview of experimented ML methods and their input features, where
FE techniques stands for Feature Extraction techniques
ML Methods stands for Machine Learning techniques

Method ID	FE Technique	ML Technique	Source
M1	FE1	ML1	[9]
M2	FE3	ML1	[9]
M3	FE2	ML2	[11]
M4	FE3	ML2	[11]
M5	FE2	ML3	[11]
M6	FE3	ML3	[11]
M7	FE1	ML4	[11]
M8	FE3	ML4	[11]

(continued on the next page)

Table 5.3: *(continued)* An overview of experimented ML methods and their input features

Method ID	FE Technique	ML Technique	Source
M9	FE2	ML5	[11]
M10	FE3	ML5	[11]
M11	FE2	ML6	[11]
M12	FE3	ML6	[11]
M13	FE2	ML7	[11]
M14	FE3	ML7	[11]
M15	FE2	ML8	[11]
M16	FE3	ML8	[11]
M17	FE5	ML6	[10]
M18	FE5	ML9	[11]
M19	FE4	ML10	[12]
M20	—	ML11	[12]
M21	FE5	ML12	[27]
M22	FE5	ML13	[28]

5.2 | Experiment 1: Method Replication

This section describes the process of replicating ML methods based on the descriptions of adopted papers [9]–[12] discussed in the section Section 2.3. The Table 5.3 lists the feature extraction techniques followed by the ML methods that are used to replicate the models in the original papers.

5.2.1 | Experiment Goals

Since the ML methods' implementation of the papers [9]–[12] are not publicly available as well as our dataset is different from the datasets used in these methods, model replication is necessary which gives us a common ground (i.e. base models) to carry out the experiments and compare results. In this experiment, we attempt to recreate these methods based on the descriptions in the papers. The setup for each method is described in the subsequent sections below.

5.2.2 | Experiment Process

5.2.2.1 | Method M1

The whole process to replicate the model starts with splitting up the data using StratifiedKFold of Scikit-learn. This technique helps splitting data into training and testing sets and preserves the percentage of requirements for each class in each set. The dataset is split into a training set and a testing set. The training set takes 70% of the data while the testing data occupies 30%, as specified in the original paper, the chosen ratio that the authors use to split data into training and test set is 7:3.

Then, features are extracted from the training set using TfidfVectorizer of Scikit-learn as explained in Section 5.1.4.1, followed by inputting these extracted features into the ML technique (ML1). This trained model will be tested using the prepared testing data to evaluate its performance.

5.2.2.2 | Method M2

The dataset is first split with the ratio that the authors use to split data is 7:3 using StratifiedKFold of Scikit-learn. Then, converting raw text data to meaningful features is proceeded using BoW technique as explained in Section 5.1.4.3, followed by inputting these extracted features into the ML2 technique. Then this trained model will be tested using the prepared testing data to evaluate its performance.

5.2.2.3 | Method M3, M5, M9, M11, M13, M15

The whole process to replicate the model starts with splitting data using StratifiedKFold of Scikit-learn. This technique helps splitting data into training and testing sets and preserves the percentage of requirements for each class in each set. Since the training and testing split ratio is also not mentioned in the original paper [11], the chosen ratio to split data into training and test set is 8:2.

Then, Tf-Idf N-Grams features are extracted using TfidfVectorizer of Scikit-learn as explained in Section 5.1.4.2. Features are input to different ML algorithms. These trained models will be tested separately using the prepared test data.

5.2.2.4 | Method M7

The whole process to replicate the model starts with splitting data using StratifiedKFold of Scikit-learn. This technique helps splitting data into training and testing sets and preserves the percentage of requirements for each class in each set. Since the training and testing split ratio is also not mentioned in the original paper [11], the chosen ratio to split data into training and test set is 8:2.

Then, Tf-Idf Word Level features are extracted using TfidfVectorizer of Scikit-learn as explained in Section 5.1.4.1. Features are input to the ML4. The trained model will be tested using the prepared test data.

5.2.2.5 | Method M4, M6, M8, M10, M12, M14, M16

Since the training and testing split ratio is also not mentioned in the original paper [11], the chosen ratio to split data into training and test set is 8:2 using StratifiedKFold of Scikit-learn. This technique helps splitting data into training and testing sets and preserves the percentage of requirements for each class in each set. Then, CountVectorizer is constructed as explained in Section 5.1.4.3 and fits to the training data. The training data and testing data are then transformed based on the learned vocabulary dictionary to retrieve BoW feature. These features are inserted into ML algorithms to train them. Lastly, the test set is input to the trained models to evaluate their performance.

5.2.2.6 | Method M17, M18

Lowercasing is carried out first. As mentioned above, features such as POS N-Grams, %noun, %verb etc will be drawn out. Then, the preprocessing phase continues with stopwords removal and lemmatization. At this point, the dataset is split into training and testing sets and the CountVectorizer is defined as discussed in Section 5.1.4.5. The constructed CountVectorizer fits to the training set and transforms it into a document-term matrix. This vectorizer then applies the vocabulary dictionary obtained from the training set to transform the testing set into a document-term matrix. The matrix of the training set will be inputted into the machine learning method, in this case, SVM. The trained model then performs the testing to evaluate the performance of it.

5.2.2.7 | Method M19

A training dataset and a testing dataset are split from the initial dataset using StratifiedKFold of Scikit-learn. This technique helps splitting data into training and testing sets and preserves the percentage of requirements for each class in each set. Then, CountVectorizer as described in Section 5.1.4.4 fits and transforms the training set before transforming the testing set based on the vocabulary dictionary of the training set. At this point, transformed training data is fed to the neural network to train. The trained network then performs the testing to evaluate its performance.

5.2.2.8 | Method M20

Firstly, the dataset is split into a training dataset and a testing dataset using StratifiedKFold of Scikit-learn. This technique helps splitting data into training and testing sets and preserves the percentage of requirements for each class in each set. A vocab dictionary which maps words into indices is built on the training set using build_vocab_from_iterator function provided by torchtext - a Python library which is a part of Pytorch framework. Based on this dictionary, we transform all the training and testing data into indices. All transformed data are then padded so they all have the same length.

Training data at this point is input into the network. After learning, the network carries out some tests to evaluate its performance.

5.3 | Experiment 2: Cross Requirement Types Input

After the base models have been established, in this experiment, requirements of the opposite type are fed to the same ML methods to evaluate their cross-applicability. The Table 5.3 lists the feature extraction techniques followed by the ML methods that are used to conduct the experiment 2.

5.3.1 | Experiment Goal

As stated in the Section 1.2, this experiment's purpose is to investigate the cross-applicability of ML methods which are intended for either functional or nonfunctional requirements, when being used to classify the other.

5.3.2 | Experiment Process

All of the setup for the method ID from 1 to 17 in this experiment is kept the same as in the Experiment 1 (Section 5.2), with only the input data is swapped. The methods that are used to classify functional requirements will be fed with non-functional requirements and vice versa. For the method M18 and M19 (i.e. ANN and CNN), their architectures have a slight change in the number of neurons in the output layer due to the change of the number of classes (from 11 classes of non-functional requirements to 6 classes of functional requirements).

5.4 | Experiment 3: Mixed Requirement Types Input

Instead of training ML methods with input data that only belong to functional or non-functional requirements, in this experiment, we will put in both types of requirements and see the difference between the performance of these methods and other methods of the above experiments. The Table 5.3 lists the feature extraction techniques followed by the ML methods that are used to conduct the experiment 3.

5.4.1 | Experiment Goal

The goal of this experiment is to see if ML methods' learning capabilities still can be expanded to classify both types of requirements (functional and non-functional requirements) or they already reach their learning limit. This experiment is necessary and beneficial because if succeeded, these methods can reduce an extra step in the categorization process.

5.4.2 | Experiment Process

Since we want to test out the limit of leaning capabilities of these methods, all of the classes should be included, therefore in this experiment, significant classes will not be used to train these methods.

Except for the input data, all other setup for the method ID from 1 to 17 in this experiment is similar as in the Experiment 1 (Section 5.2).

As mentioned above, the data input in these models are both functional and non-functional requirements. For the method 18 and 19 (i.e. ANN and CNN), the architecture has a slight change in the number of neurons of the output layer due to the change of the number of classes (i.e. from 11 classes of non-functional requirements to 17 classes of both functional and non-functional requirements).

5.5 | Experiment 4: Other ML Methods

This experiment explores ML methods that are not mentioned in any of the selected work discussed above.

5.5.1 | Experiment Goal

Besides methods that are discussed above, we try to solve requirement classification problems using other new Machine Learning methods.

5.5.2 | Experiment Process

5.5.2.1 | Random Forest Classifier

Random Forest is an ensemble method which consists of decision trees. When classifying any data sample, the forest collects its trees' predictions for the sample and gives the final prediction based on the prediction that most of its trees vote for. The reason for choosing this method for this experiment is that according to our literature review "The Landscape of Machine Learning in Requirements Engineering", Random Forest is the fourth most used machine learning algorithm. Hence, we want to investigate the effectiveness of this method on the requirements classification task given our dataset.

The experiment process is similar to other experiments. First, the used dataset is split then we proceed to extract features on the training set and the test set. A combination of this machine learning method and feature techniques as introduced in the Section 5.1.4 are explored to see which combination yields the best results. After experimenting with different combinations, the combination of multiple features Section 5.1.4.5 and Random Forest produces the best result which will be used as the main method in this experiment. The method at this point will be trained on the training set. The trained model then performs the testing to evaluate its performance on the test set. The best result will be presented in the Chapter 6.

5.5.2.2 | XGBoost

Extreme Gradient Boosting (XGBoost) is a popular algorithm in the last couple of years because of the extensive usage of winning competitors for their solutions in machine learning challenges. Boosting is an approach that builds models sequentially, each successor focusing on solving instances that its predecessor incorrectly classifies (i.e. difficult instances). The more difficult instances are, the more weight they get. Gradient Boosting is quite similar but instead of giving weight for difficult instances, it minimizes its loss when adding new models by using the gradient descent. Normally, gradient boosting approaches are quite slow and unscalable because it takes time to build models sequentially. XGBoost is built to overcome that disadvantage which focuses on speed and performance. With all reasons mentioned above, we want to assess its performance on the requirements classification task given our dataset.

The experiment process is similar to other experiments. First, the used dataset is split then we proceed to extract features on the training set and the test set. A combination of this machine learning method and feature techniques as introduced in the Section 5.1.4 are explored to see which combination yields the best results. After experimenting with different combinations, the combination of multiple features Section 5.1.4.5 and XGBoost produces the best result which will be used as the main method in this experiment. The method at this point will be trained on the training set. The trained model then performs the testing to evaluate its performance on the test set. The best result will be presented in the Chapter 6.

6 Evaluation & Discussion

6.1 | Evaluation Metrics

This section is to explain the metrics used in the project to evaluate the performance of ML methods. These metrics include accuracy, precision, recall and F1 which are known for their simplicity and their popularity in classification problems.

Accuracy is a ratio of correctly predicted instances over the total number of instances. Accuracy works best only if each class contains the same number of instances (i.e. balanced dataset). However, with an imbalanced dataset, accuracy is not a proper measure to assess the performance of models. Hence, instead of relying only accuracy, precision, recall and F1 are often utilized in combination with accuracy if the imbalanced dataset is used.

Assume that we have a binary classification problem where a ML method can be utilized to classify whether a requirement is functional or not. In this case, true positives (TP) are instances that are classified correctly as functional requirements while true negatives (TN) are instances that are classified correctly as not functional requirements. Instances that are incorrectly classified as functional requirements are false positives (FP) while instances that are incorrectly classified as not functional requirements are false negatives (FN). Together, the number of these instances are the building blocks that define the precision, recall and F1 score.

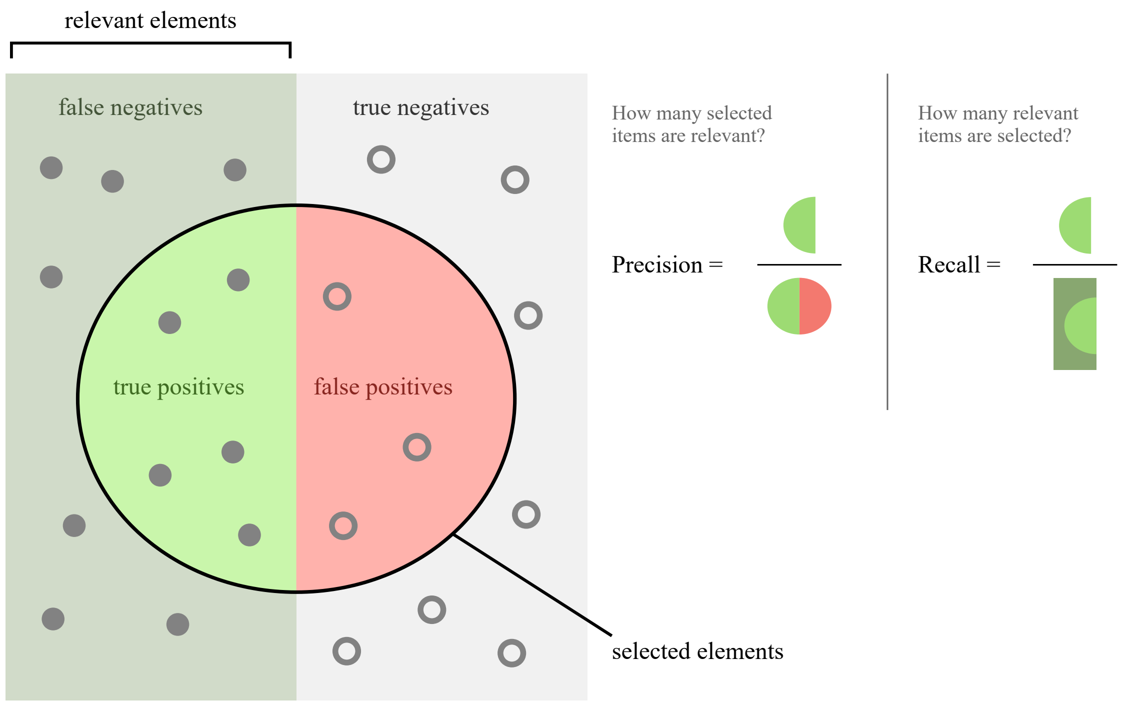

Figure 6.1: Precision and recall [29]

Precision is a metric used to calculate the proportion of instances that are correctly predicted as positive (TP) over the total number of instances that are predicted positive (TP and FP).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.1}$$

Recall is a metric used to calculate the proportion of instances that are correctly predicted as positive (i.e. true positives) over the total number of actually positive instances (TP and FN).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.2}$$

Back to the assumption made above, in that scenario, high recall but low precision indicates that the method classifies a lot of requirements as functional requirements but only a few requirements are predicted correctly (i.e. only a few of them are functional requirements) while low recall but high precision indicates that the method does not think a lot of requirements are functional requirements, but when it does, it predicts correctly (i.e. requirements are correctly classified as functional requirements). High precision and high recall is the best scenario where the model correctly picks functional requirements out of all instances.

The harmonic mean of precision and recall is called F_1 score. This metric is used when precision and recall are valued equally (i.e. they are both important).

$$F_1 \text{ score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{6.3}$$

The Figure 6.2 shows the spread and the distribution of best results of established methods collected from selected papers [9]–[12] using box plots. The maximum and minimum values among precision values of these established methods are 0.878 and 0.54 respectively while the median value of these precision values equals 0.67. 0.85 and 0.48 are the maximum and minimum values among recall values of the ML methods of the selected papers. The median value of these recall values is 0.62. For F1 score, the minimum value and the maximum value are 0.52 and 0.86 with the median equals 0.61.

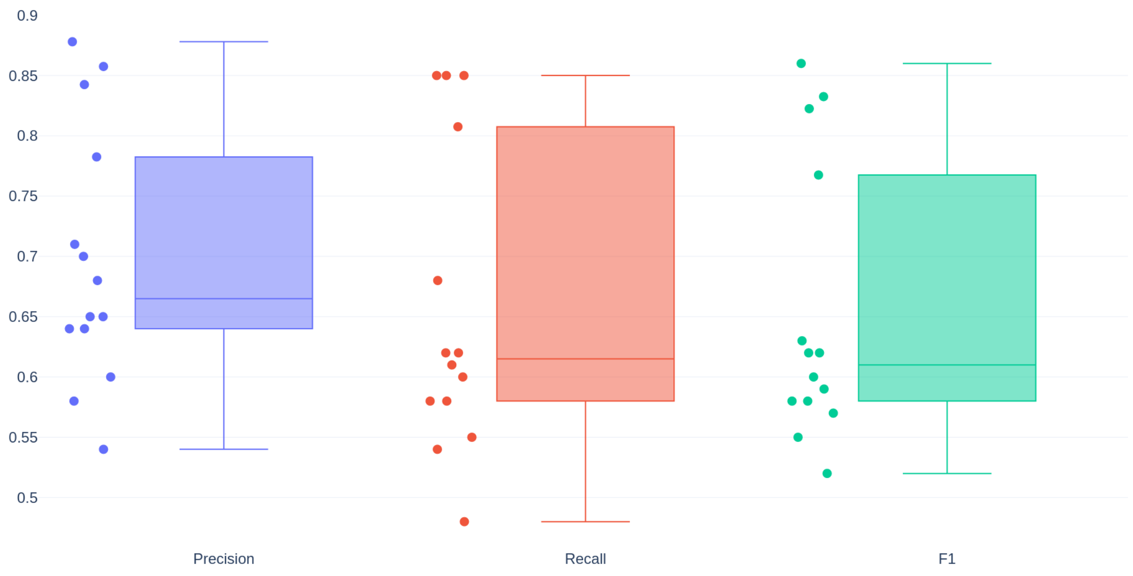

Figure 6.2: Metrics Values Distribution Box Plots

6.2 | Evaluation Results

This section presents the results of the experiments set up in Section 5.1.6 and discusses their results to answer the research questions in the Section 1.3. Due to the sheer amount of quantitative data produced, this section will only provide an overview of the evaluation results, whereas the full result data is available in Appendix B.

The content of this section is divided based on the established methods from Table 5.3, which are combinations of feature extraction techniques and machine learning techniques, so that it is more convenient to follow the results produced by them in each experiment.

6.2.1 | Results Overview

This section provides an overview of the results of the experiments. The full results are available in Appendix B. To better understand the results, graphs are used to illustrate and visually compare the performance of each method when being used for classifying different types of requirements. Mixed-type requirements should include all classes while significant classes experiment only include some types of requirements as discussed in the Section 5.1.3. Therefore, there are two types of graphs that differ in the number of axes for comparing the methods' performances in non-functional, functional, and mixed-type requirements classification. Combined with the use of only significant classes or all classes for training, there are in total six graphs, each showing the methods' performances in a certain metric: precision, recall, or F1 score. For any kind of graph, each method is presented as a point. The style of the point is determined by the original intention of that method as shown in the legend of each graph.

The first kind of graph illustrates the methods' performances in classifying non-functional and functional requirements when trained on only significant classes. These graphs use a scatter plot as a base where the horizontal axis indicates a method performance in an evaluation metric when being used to classify non-functional requirements, whereas the vertical axis indicate the same metric for classifying functional requirements. There is a dotted line bisecting each graph. The closer a method is to this line, the more similar its performances are for classifying the two types of requirements, i.e. higher cross-applicability. In the plane of these graphs, there are two special regions color coded in red and green. These regions are created from the median of each metric, where the red zone indicates that a method, if fallen in, performs worse than average in both classification tasks of the two types of requirements.. In contrast, if a method is placed inside the green zone, that method performs better on average in both tasks. In short, the closer to the dotted bisector and the closer to the far corner of the green zone, the more desirable a method is.

The second kind of graph has similar meaning to the first kind while extending up one more axis of comparison. These graphs illustrate the methods' performances in classifying non-functional, functional, and mixed-type requirements when trained on all classes. An instance of this kind of graph has three planes, each is a sheared image of the first kind of graph. The bottom plane compares between classifying non-functional and functional requirements, while the left plane compares between classifying non-functional and mixed-type requirements, and the right plane compares between classifying functional and mixed-type requirements.

It is observed that when being trained with requirements of significant classes, the performance of ML methods are improved as the number of ML methods lying in the green zone is increased and the number of ML methods falling in the red zone is decreased compared to when being trained on requirements of all classes.

It can be seen from these graphs that most of the methods (75% of these methods) can be cross-applicably used (i.e. can be used for both classifying functional requirements and classifying non-functional requirements). However, two methods that are originally intended for functional requirements classification do not perform well in both functional requirements and non-functional requirements classification compared to non-functional methods. Near to none methods perform well in mixed-type requirements classification, which may hint at the lack of learning capability of these methods. New methods, specifically, XGBoost and Random Forest classify functional requirements better than they classify non-functional requirements or mixed-type requirements. However, their performance is only in the white zone at best and not close to the bisectors of the graphs which shows that the performance of these methods are not very decent and they are not entirely cross-applicable.

The Table 6.1 shows the training time of each method of our research on the mixed-type requirements dataset. Overall, given the hardware specification shown the Section 5.1.1, the training time of methods in our study is fast. The training time of Random Forest is highest while variants of Naive Bayes have the fastest training time. For Random Forest, building various decision trees to create a forest probably takes a lot of time which slows down the training phase. For Naive Bayes variants in this study, namely, Gaussian Naive Bayes, Bernoulli Naive Bayes, Multinomial Naive Bayes, the incredible speed is gained because these methods only need to calculate the probability of each class and the probability of each class given different input values based on the Bayes theorem. CNN takes the second longest time for training the network. The complexity of the architecture of CNN could be the main reason behind the slow training time

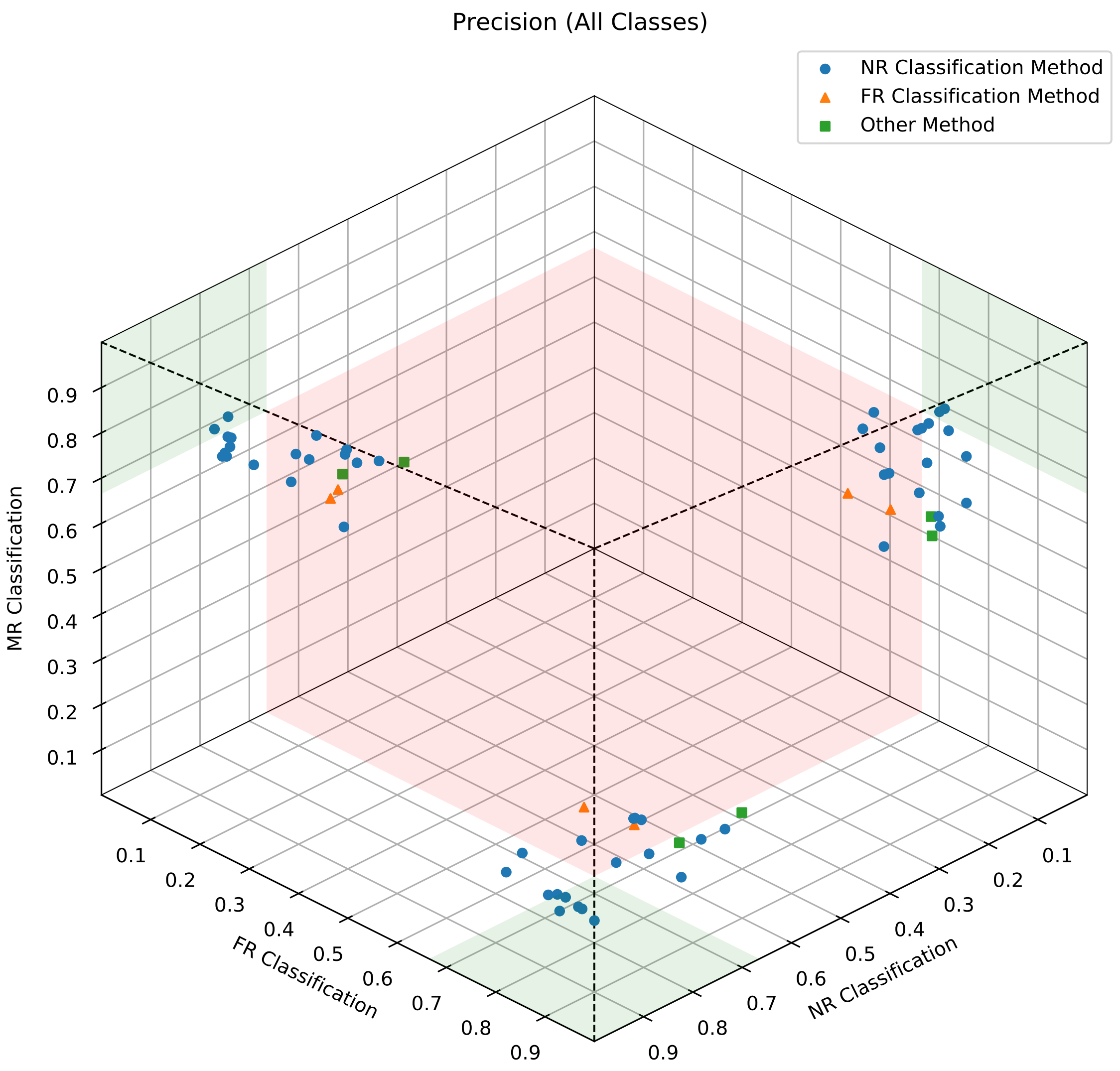

Figure 6.3: Overview on the precision values trained on all classes

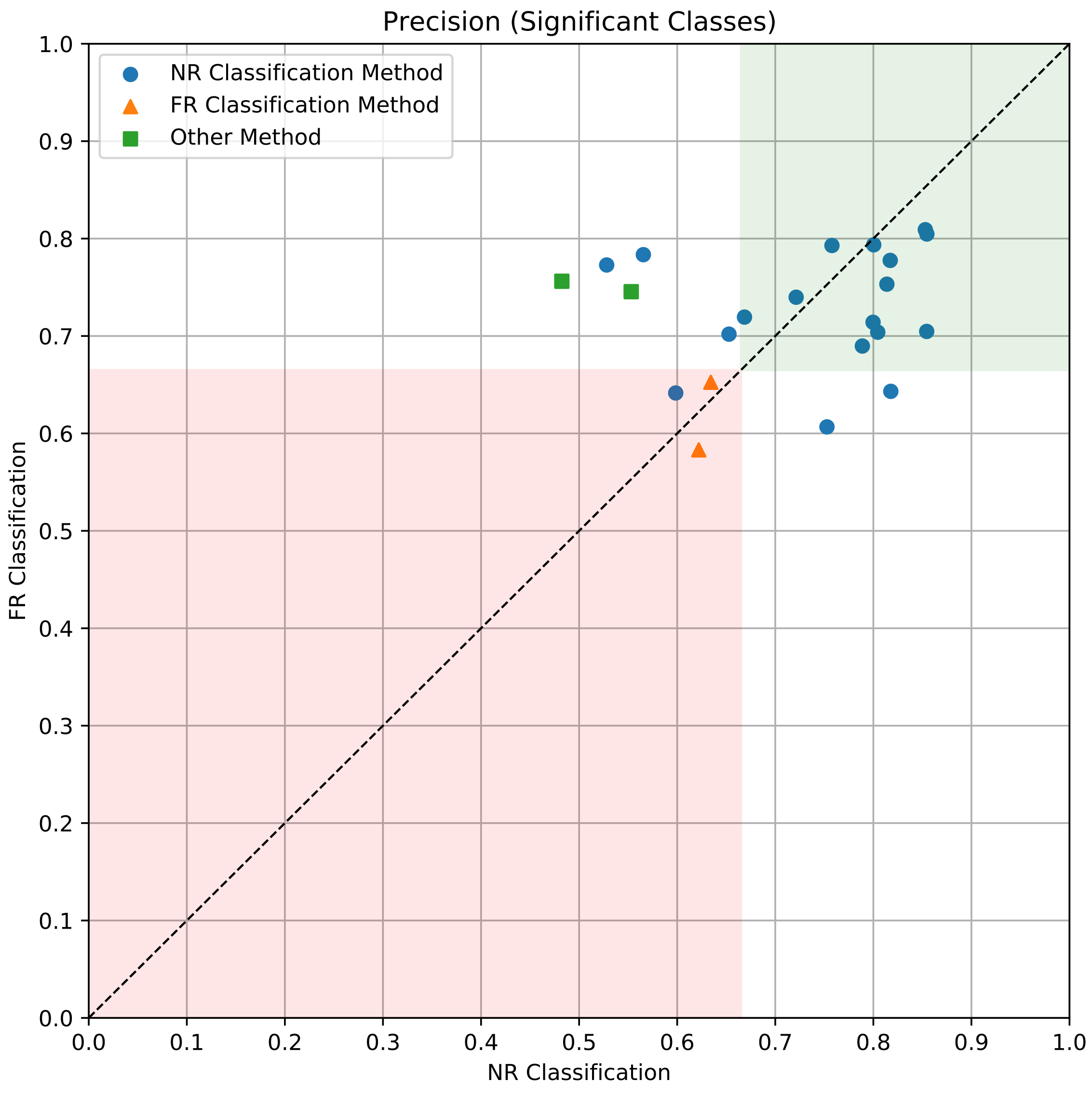

Figure 6.4: Overview on the precision values trained on only significant classes

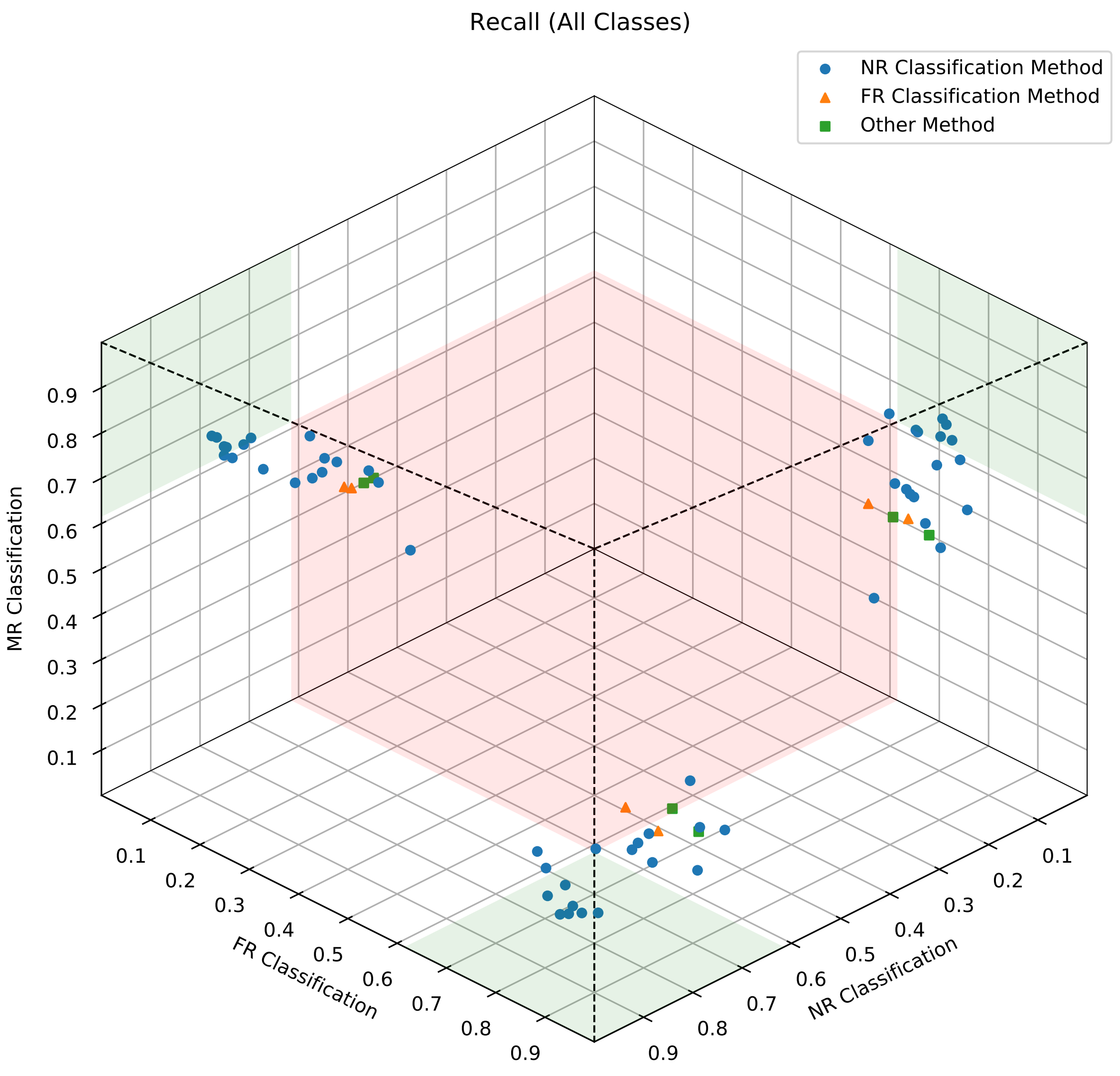

Figure 6.5: Overview on the recall values trained on all classes

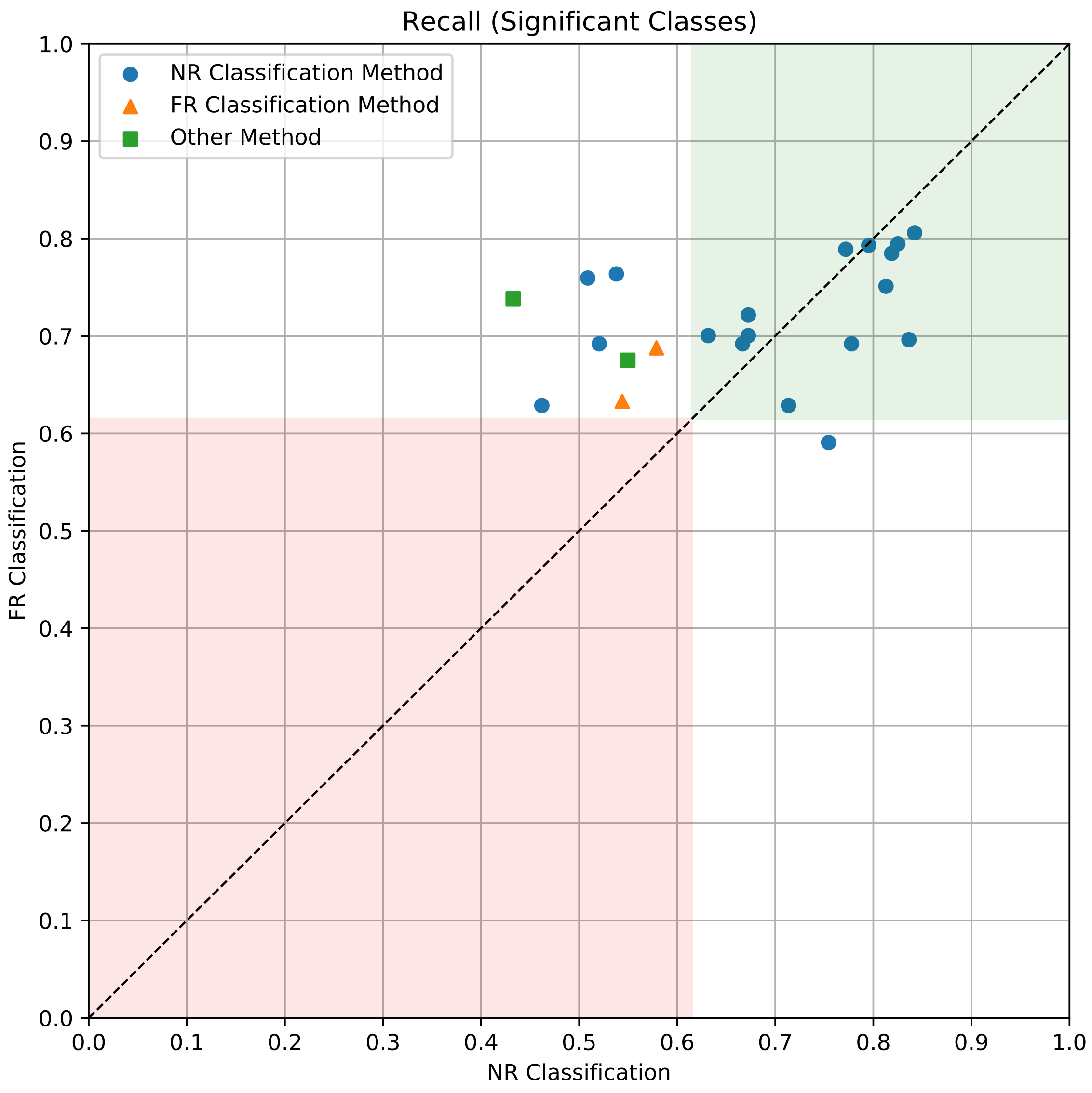

Figure 6.6: Overview on the recall values trained on only significant classes

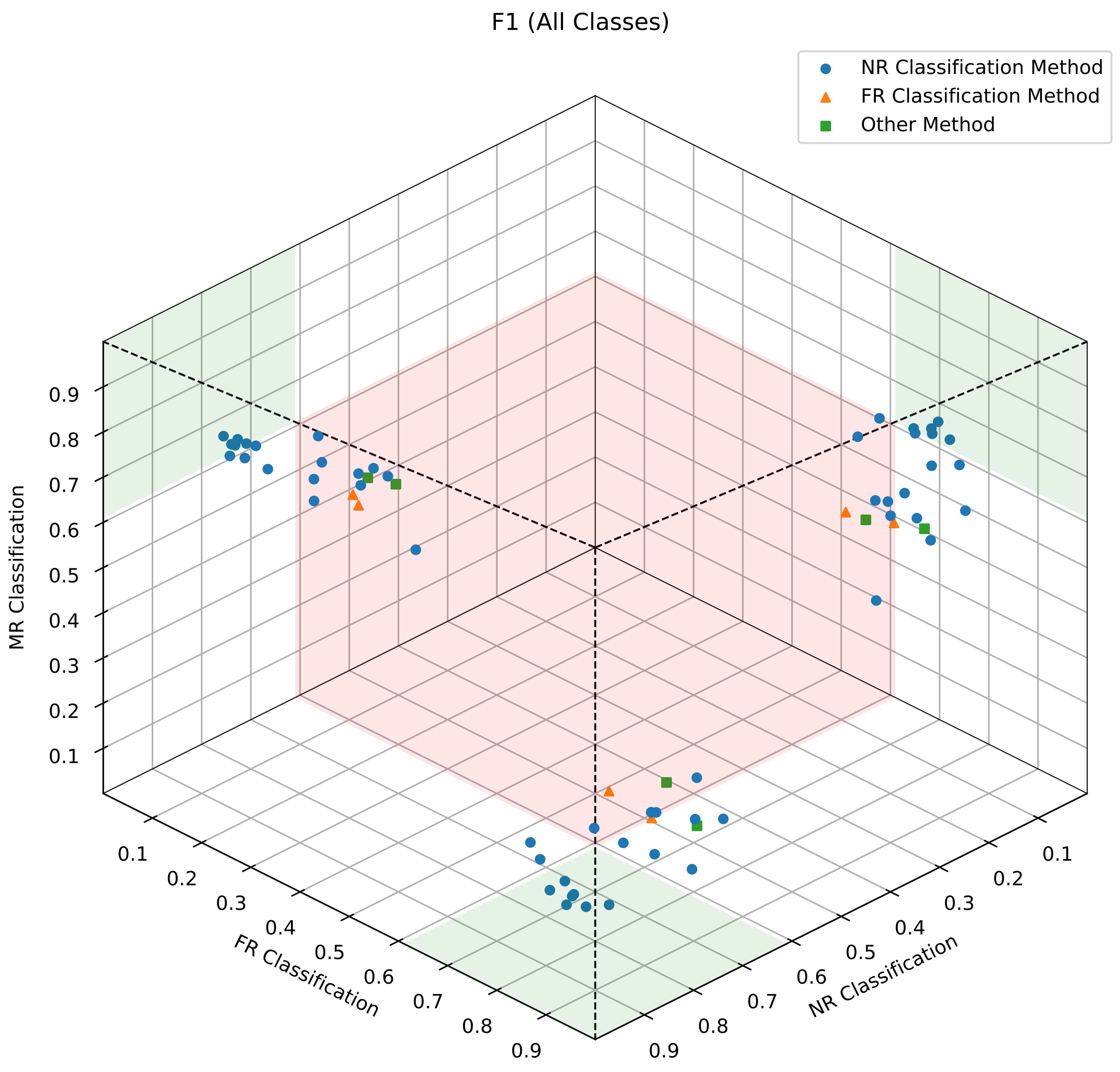

Figure 6.7: Overview on the F1 scores trained on all classes

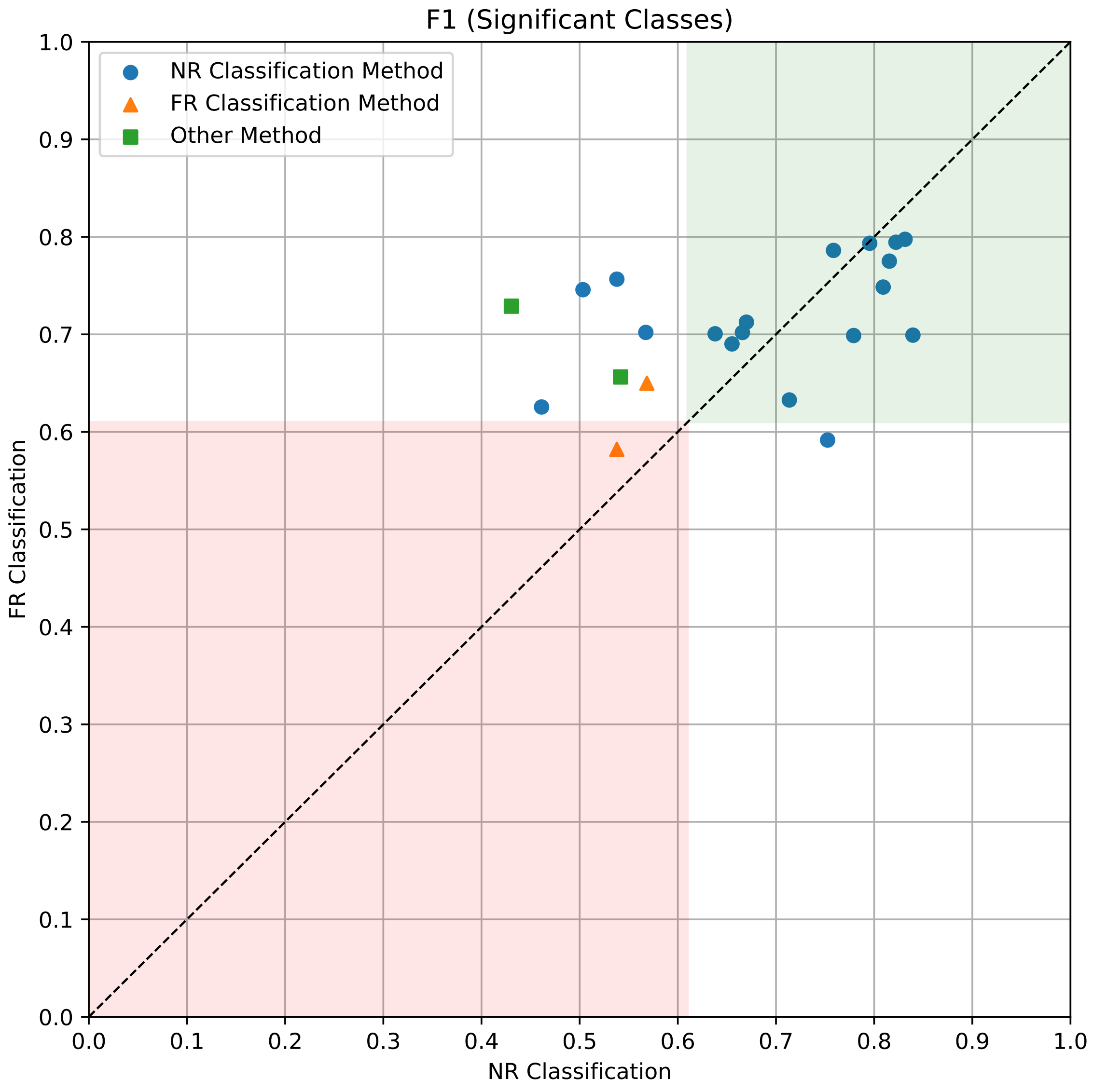

Figure 6.8: Overview on the F1 scores trained on only significant classes

compared to other methods.

Table 6.1: Methods Training Time on Different Types of Requirements

Method	FR	NR	MR
M1	0:00:03.066243	0:00:02.636284	0:00:32.466857
M2	0:00:02.840371	0:00:02.316606	0:00:32.466857
M3	0:00:00.046819	0:00:00.032004	0:00:00.030088
M4	0:00:00.010460	0:00:00.015334	0:00:00.012609
M5	0:00:00.168756	0:00:00.097504	0:00:00.235937
M6	0:00:00.014530	0:00:00.011936	0:00:00.018035
M7	0:00:00.008041	0:00:00.004660	0:00:00.007502
M8	0:00:00.014084	0:00:00.023092	0:00:00.016087
M9	0:00:00.015862	0:00:00.009071	0:00:00.022023
M10	0:00:00.000523	0:00:00.000709	0:00:00.000359
M11	0:00:03.614631	0:00:02.449048	0:00:07.292294
M12	0:00:00.170945	0:00:00.151414	0:00:00.234395
M13	0:00:00.863041	0:00:00.890506	0:00:01.062518
M14	0:00:00.104966	0:00:00.110774	0:00:00.152510
M15	0:00:01.007318	0:00:00.645720	0:00:04.032186
M16	0:00:00.083207	0:00:00.051328	0:00:00.176058
M17	0:00:05.272762	0:00:03.242775	0:00:39.129029
M18	0:00:11.047466	0:00:10.187083	0:01:44.576935
M19	0:00:00.783336	0:00:00.482951	0:00:01.239276
M20	0:02:19.578522	0:01:27.252679	0:08:42.463433

(continued on the next page)

Table 6.1: *(continued)* Methods Training Time

Method	FR	NR	MR
M21	0:00:52.311821	0:00:52.888177	0:08:39.630521
M22	0:02:04.974625	0:01:19.235434	0:11:49.468390

6.2.2 | Method M1: Ensemble Method + TF-IDF Word Level Feature

The result of the method M1 as shown in Table B.1 is not similar to the result of the original paper [9] even though efforts are made to reconstruct the same method based on the description of the paper. This could be because the dataset used in our experiment is different from the dataset used in the paper. The quality of the dataset could positively or negatively affect the performance of the model. The dataset used in the paper is balanced (i.e. the number of requirements of classes are quite equally distributed) while our dataset is imbalanced which negatively affects the result. The detailed configurations of the feature extraction techniques and of the machine learning algorithms are not provided. This unrevealing could lead to the difference between the paper's model and our model which results in the difference in the result.

The result of classifying only functional requirements from three significant classes of the method M1 shows the effect of a slight imbalanced dataset. As explained in the method setup, since we split the dataset into the training set and the testing set using the StratifiedKfold, the training data and the testing data had the same percentage of requirements with a given class. In the three significant functional requirements classes, Action Constraint is the class that has the least requirements with only 45 requirements in the test set. Because of the difference in the number of requirements in each class, the performance of classifying each class in the test set is different too. While Enablement requirements and Solution requirements are likely to be classified correctly by the method M1, Action Constraint requirements are not since the small number of Action Constraint requirements compared to other classes makes it difficult for the ML method to learn possible characteristics of it, thus, also difficult to differentiate it with others. However, because the dataset is less imbalanced, overall, it is observed that the performance of classifying requirements from only significant functional classes of the method M1 is better than its performance of classifying instances from all functional classes.

It can be seen from the Table B.1 that the result when using the method M1 to classify non-functional requirements is quite close to the result when classifying functional ones, approximately 0.02 in the F1 score. From this observation, the method M1 seems to be able to classify both non-functional requirements and functional requirements.

Similarly to the experiment of classifying only requirements from significant functional classes, the result obtained from the non-functional requirements classification from significant classes is better than the original result (i.e. the result of classifying requirements from all non-functional classes). This better performance could be caused by the change in the dataset from a severe imbalanced dataset to a slight imbalanced dataset. In more detail about the obtained result, for classes that have a small amount of requirements compared to others such as Look & Feel class, the method M1 finds it difficult to classify these requirements correctly. For Operational and Security requirements, the result is high in precision but low in recall indicating that if the method M1 classifies any requirement as Operational or Security requirement, it is likely correct but the method M1 does not usually classify requirements as Operational or Security requirement.

Compared to other results of the method M1 obtained in the experiments 1 and 2, the result of the method M1 when classifying the mixed type requirements is lower with 0.3554 in F1 score.

6.2.3 | Method M2: Ensemble Method + BoW Feature

Similarly to the method M1, the result of the method M2 as shown in Table B.2 is different from the results provided by the paper [9]. The difference in the used dataset between our study and the paper and the possible dissimilarity between the two methods regardless of the effort in replicating the same method as described in the paper could be the reasons for the inequality of the two results.

In general, the overall performance of classifying only requirements from significant functional classes of the method M2 is better than the performance of classifying requirements from all functional classes. As explained in the Section 6.2.2 above, the improved performance could be because the training set is more balanced compared to the training set for classifying all functional requirements and the number of classes that the method M2 has to predict is reduced by half. Similarly to the method M1, the method M2 tends to correctly classify requirements from classes that have a significant amount of requirements such as Enablement and Solution. For the Action Constraint class, the number of requirements of this class is not as many as the two other classes hence it is difficult for the method M2 to differentiate requirements of this class with other requirements.

The non-functional requirements classification ability of the method M2 is not as good as its functional requirements classification ability with the F1 score difference being more than 0.1, as seen from the Table B.2.

In the experiment which let the method M2 classify only non-functional requirements from 5 significant classes, its learning ability is improved with 0.07 higher than the non-functional requirements classification from all classes in precision. This achievement, as explained in the Section 6.2.2 above, could be due to the less severe balance dataset. One of the interesting details in the obtained result is that the Look & Feel class has a 1.0 point in precision but only 0.09 in recall. As explained above, the method M2 even though it does not usually classify requirements as Look & Feel, when it does, the predictions are correct.

It is shown that the method M2's performance when classifying mixed type requirements is not as good as its own performance when classifying functional requirements or non-functional requirements which is significantly lower with 0.22 less than the result of classifying functional requirements and 0.11 lower than the result of classifying non-functional requirements.

6.2.4 | Method M3: MNB + TF-IDF Word Level Feature

Unlike the other two above mentioned methods, the method M3 outperformed the original paper's method [11]. This difference in the results of the two methods could be explained by the fact that the dataset used in our study is different than the paper's dataset. Even though, as mentioned above, the difference in the used datasets between our study and the paper [9] has a negative effect on the results, in this case, the difference may affect positively. Despite the fact that both of the datasets are imbalanced, our dataset contains a greater number of requirements (804 requirements) which is more than doubled the number of requirements of the paper's dataset (370 requirements). This excess in the amount of non-functional requirements may help the method gain more insight about each class which then helps it classify these requirements better.

When classifying requirements of only significant 5 non-functional classes, the result of the method M3 is remarkably better than the result obtained when classifying requirements of all non-functional classes, with 0.14 higher in F1. This increase could be due to the less imbalanced used dataset and the decrease in the number of needed-to-classify classes.

In contrast to the method M2, the method M3 shows its ability to classify functional requirements although is not intendedly created for with the gap between the result of classifying functional requirements and of classifying non-functional one equals 0.01 in F1 score.

The result of classifying only requirements from significant functional classes is also higher than the result of classifying requirements from all functional classes. As discussed above, this achievement could be because the number of classes needed to predict is reduced by half - from 6 classes to 3 classes as well as the filtered dataset is less imbalanced as we filtered out the minority classes and only kept significant classes.

The result obtained from classifying mixed-type requirements of the method M3 is moderately lower than the the results obtained in the two previous experiments which shows that the method M3 seems unsuitable for classifying requirements into 17 sub-classes at a time.

6.2.5 | Method M4: MNB + BoW Feature

The method M4 obtained a better result than the original method [11]. As discussed above, this better performance could be because there are more non-functional requirements in our dataset than in the paper's dataset even though it is still an imbalanced one and the larger dataset is not always better in this machine learning field.

It seems that the method M4 classifies requirements remarkably better when we decrease the number of non-functional classes needed to be classified from 11 to 5 with 0.15 higher than the result acquired when classifying requirements from all non-functional classes. This improvement could be due to the change of the used datasets as we removed the minority functional classes from our dataset which makes the dataset less imbalanced than the original dataset.

The method M4 classifies functional requirements worse than classifies non-functional requirements, as shown in Table B.4, with f1 score 0.112 lower than when classifying non-functional ones.

The result of the method M4 of classifying requirements of significant functional classes is also better than the result obtained from classifying requirements from all functional classes. As discussed above, the filtered dataset with only the majority classes and the decrease in the number of classes could be the reasons for this improvement.

Even though the result of the method M4 when classifying mixed-type requirements is worse than the result received when it classifies non-functional ones with the gap as 0.08 in F1, the mixed-type requirements classification result is slightly higher than the results obtained when classifying functional requirements with 0.03 difference in F1 score.

6.2.6 | Method M5: GNB + TF-IDF N-Grams Feature

It can be seen that the method M5 achieved a higher performance than the original method [11]. This achievement could be attributed by the greater amount of non-functional requirements in our dataset than the original dataset [11] as discussed above.

As we removed insignificant classes from the dataset, it is observed that the method M5 achieves a better performance than when it classifies requirements from all 11 non-functional classes with 0.06 higher in F1. This improvement could be caused by using a less imbalanced dataset to train instead of the original dataset and the reduction in number of classes needed to be predicted from 11 classes to 5 classes.

When changing the input data, the method M5 shows that it is still able to classify functional requirements even though the obtained result is not as good as when it classifies non-functional ones, with 0.09 lower in F1.

The result of the method M5 when classifying requirements from only significant functional classes is higher than its results when classifying requirements of all functional classes with approximately 0.09 higher in precision, recall and F1. Training the method with a less imbalanced dataset and the drop in the number of classes needed to be predicted could be the reasons behind this achievement.

It is observed that the ability of the method M5 in classifying mixed-type requirements is worse than its ability in classifying functional requirements or non-functional requirements which is approximately 0.14 lower than non-functional requirements classification and 0.05 lower than functional requirements classification in F1.

6.2.7 | Method M6: GNB + BoW Feature

The result produced by the method M6 as shown in the Table B.6 is slightly higher than the result of the original method. The larger amount in non-functional requirements in our dataset than the original dataset could be the reason for this achievement even though the distribution of requirements between classes are not evenly proportioned.

The method M6 achieved a higher score in classifying requirements of significant non-functional classes than when classifying requirements of all non-functional classes with 0.09 higher in F1. This improvement could be due to that the used dataset is less imbalanced compared to the original dataset and the number of classes needed to predict is reduced, as mentioned in the sections.

It is observed from the Table B.6 that the method M6 when classifying functional requirements is not as good as when classifying non-functional ones with the difference being 0.13 lower in F1.

Similarly to when it classifies non-functional requirements of significant classes, this method also achieves a better score in classifying functional requirements of significant classes compared to when it classifies requirements of all functional classes even though the difference in results is not as much as its result of classifying non-functional requirements, with approximately 0.06 higher in precision, recall and F1. This better performance could be because of using the less imbalanced dataset when training and the reduction in the number of classes that are needed to classify.

The method M6 performance when classifying mixed-type requirements is very close to its performance when classifying functional requirements with just 0.02 difference in F1. While on the other hand, the result of mixed-type requirements classification needs 0.13 point in F1 to bridge the gap between it and the result of non-functional requirements classification.

6.2.8 | Method M7: BNB + TF-IDF Word Level Feature

The method M7 showed a better performance than the original method [11]. As discussed above, the dissimilarity in the used datasets and the greater number of non-functional requirements in our dataset may lead to this improvement.

We witness the improvement in the result of the method M7 when classifying non-functional requirements of only significant classes compared to its result when classifying all non-functional requirements. This could be explained by the characteristic of the used dataset which is now less imbalanced and the number of classes needed to categorize is reduced by half.

The result of the method M7 in classifying the functional requirements surpasses its own result in classifying non-functional one by approximately 0.13 in the F1 score, even though it is not intended for this classification task.

The better performance is also observed as shown in the Table B.7 in the case of classifying functional requirements of only significant classes with 0.09 higher in precision and F1 and 0.05 higher in recall compared to the result of classifying functional requirements of all classes. As discussed in the sections above, this achievement could be because the method M8 is trained with a less imbalanced training dataset compared to the original dataset.

The method M7's ability to classify mixed-type requirements falls behind its ability to classify only one requirement type at a time which result is less than 0.21 in F1 compared to the result for classifying functional requirements and less than approximately 0.08 compared to the result for classifying non-functional requirements.

6.2.9 | Method M8: BNB + BoW Feature

The result of the method M8 is better than the result of the paper's method [11]. The difference in these results could be caused by the difference in the used datasets. Although our dataset is as imbalanced as the paper's, the total number of non-functional requirements in our dataset is more than twice that of the original paper's dataset hence could lead to this improved result.

The result of classifying functional requirements of significant classes of the method M8 is remarkably higher than its result of classifying functional requirements of classes. This better performance could be caused by the difference in the used dataset - switching from a severe imbalanced dataset to a slight imbalanced dataset and the drop in the number of classes that need to be classified.

Similarly, as presented in the Table B.8, the method M8 shows that its ability in classifying functional requirements is better than its ability in classifying non-functional ones which is more than 0.08 in precision and 0.11 in recall and F1 score despite not being originally created for.

It seems that the method M8 classifies requirements noticeably better when we decrease the number of non-functional classes needed to be classified from 11 to 5 with 0.15 higher than the result acquired when classifying requirements from all non-functional classes. This improvement could be due to the change of the used datasets as we removed the minority functional classes from our dataset which makes the dataset less imbalanced than the original dataset.

It is shown that the method M8 cannot extend its ability to classify mixed-type requirements since its result of mixed-type requirements classification falls far behind its other results with 0.17 difference in F1 than the functional requirements classification and 0.06 difference in F1 than the non-functional requirements classification.

6.2.10 | Method M9: KNN + TF-IDF N-Grams Feature

The method M9 obtained higher results than the original method [11]. The difference in the total number of non-functional requirements in the datasets could be the reason for this high obtainment, as discussed above.

Following the same trend as other methods, the method M9 achieved a better result when classifying non-functional requirements of significant classes compared to when it classifies requirements of all non-functional classes with 0.07 higher in F1, 0.06 higher in precision and recall. As discussed in the sections above, this improvement could be because of the difference in the used dataset and the drop in the number of the needed-to-classify classes.

Even though the method M9 classifies functional requirements not as good as it classifies non-functional ones, the gap in the results is quite small, only 0.0651 in F1 score. The result shows that the method M9 is also able to classify functional requirements as is able to classify non-functional ones.

It can be seen that the result of classifying functional requirements of significant classes of the method M9 is higher than the result of classifying requirements of all functional classes with approximately 0.05 higher in precision, recall and F1. This better result can be due to the fact that the training dataset is less imbalanced than the original dataset and the number of classes that are needed to predict is dropped by half.

Compared to other results of the method M9 obtained in the two previous experiments 1 and 2, although the result of mixed-type requirements classification is lower but the difference between the three results are small with 0.04 difference in F1 between mixed-type requirements classification and functional-requirements classification and 0.09 difference in F1 between mixed-type requirements classification and non-functional ones.

6.2.11 | Method M10: KNN + BoW Feature

It is shown that the method M10 gained a better result than the original method [11]. As mentioned above, the dataset difference as well as the greater amount of non-functional requirements in our dataset could be the cause of this achievement.

When classifying only non-functional requirements of significant classes, the result of the method M10 is slightly higher than the result obtained when classifying requirements of all non-functional classes, with 0.03 higher in F1. The improvement could be the result of the change in the used dataset from a severe imbalanced dataset to a slight imbalanced dataset.

Inferred from the results shown in the Table B.10, with the result of classifying functional requirements exceeding the result of classifying non-functional ones by 0.05, the method M10 is able to classify functional requirements in spite of not being originally intended for.

The result of classifying only requirements from significant functional classes is also higher than the result of classifying requirements from all functional classes, with 0.08 higher in precision and F1 and 0.05 higher in recall. As discussed above, this achievement could be because the number of classes needed to predict is reduced by half - from 6 classes to 3 classes as well as the filtered dataset is less imbalanced as we filtered out the minority classes and only kept significant classes.

In contrast to the method M9, the result of the mixed-type requirements classification of the method M10 is quite low compared to the results of this method when classifying only functional requirements or non-functional ones.

6.2.12 | Method M11: SVM + TF-IDF N-Grams Feature

The method M11 performed better than the original method [11]. This improvement could be due to the difference in the used datasets. Our dataset contains more non-functional requirements than the paper's dataset. The larger the dataset, the more insight can be extracted which then helps improve the classification task.

It can be seen that the method M11 classifies requirements remarkably better when we decrease the number of non-functional classes needed to be classified from 11 to 5 with 0.11 higher in precision and F1, and 0.08 higher in recall than the result acquired when classifying requirements from all non-functional classes. This better performance could be due to the change of the used datasets as we removed the minority functional classes from our dataset which makes the dataset less imbalanced than the original dataset and also due to the drop in the number of classes that the method M11 needs to predict.

It is shown that the method M11 is able to classify functional requirements as it is able to classify non-functional requirements although the result obtained when classifying functional requirements is slightly lower, 0.04 less than the result obtained when classifying non-functional requirements.

The result of the method M11 of classifying requirements of significant functional classes is also noticeably better than the result obtained from classifying requirements from all functional classes with 0.11 higher in F1 and 0.09 higher in precision and recall. As discussed above, the filtered dataset with only the majority classes and the decrease in the number of classes could be the reasons for this improvement.

Even though the result of mixed-type requirements classification of the method M11 is not as good as the other results of it, it shows that the method M11 has the ability to categorize mixed-type requirements into different 17 subclasses.

6.2.13 | Method M12: SVM + BoW Feature

The results of the method M12 exceeded the results produced by the original method [11]. The difference in the used datasets as well as the larger amount of non-functional requirements in our dataset could be the reasons for this excess, as discussed above.

When we filtered out insignificant classes from the original dataset, it is observed that the method M12 performs better than when it classifies requirements from all non-functional classes with 0.06 higher in F1. This higher result could be caused by using a less imbalanced dataset to train instead of the original dataset and the reduction in number of classes needed to be predicted from 11 classes to 5 classes.

Similarly to the method M11, the method M12 shows its ability to classify functional requirements which is only 0.019 lower in F1 than its ability to classify non-functional ones.

Similar to other methods, the result of the method M12 when classifying requirements from only significant functional classes is higher than its results when classifying requirements of all functional classes with approximately 0.07 higher in recall and F1 and 0.06 higher in precision. Training the method with a less imbalanced dataset and the drop in the number of classes needed to be predicted could be the reasons behind this improvement.

Similarly to the method M11, even though the ability in mixed-type requirements classification of the method M12 does not surpass its two other abilities, the obtained result is still considered acceptable with 0.6517 in precision, 0.6348 in recall and 0.6311 in F1.

6.2.14 | Method M13: SGD SVM + TF-IDF N-Grams Feature

The method M13 obtained a higher result than the original method [11]. There could be impacts on this obtainment including the dissimilarity in the used datasets and the larger number of non-functional requirements in our used dataset.

The method M13 obtains a higher score in classifying requirements of significant non-functional classes than when classifying requirements of all non-functional classes with 0.06 higher in F1 and 0.04 higher in precision and recall. This achievement could be because the used dataset is less imbalanced compared to the original dataset and the number of classes needed to predict is reduced, as mentioned in the sections.

It is observed from the Table B.13 that the result when using the method M13 to classify functional requirements approximate the result when classifying functional ones with 0.02 difference in the F1 score. From this observation, the method M13 seems to be able to classify both non-functional requirements and functional requirements.

Similarly to when it classifies non-functional requirements of significant classes, this method also acquires a higher score in classifying functional requirements of significant classes compared to when it classifies requirements of all functional classes even though the difference in results is not as much as its result of classifying non-functional requirements, with approximately 0.08 higher in precision, recall and F1. Using a less imbalanced dataset than the original dataset when training and the reduction in the number of classes that are needed to classify could be the reasons for this better performance.

The result of mixed-type requirements classification of the method M13 is only 0.04

lower than the result of functional requirements classification in F1 while being approximately 0.1 lower than the result of non-functional requirements classification in F1.

6.2.15 | Method M14: SGD SVM + BoW Feature

It can be inferred from the results that the method M14 performed better than the original method [11]. This better performance could be caused by the difference in the used datasets. Even though both datasets are imbalanced, the change in the number of requirements could affect the performance of the method.

We witness the slight improvement in the result of the method M14 when classifying non-functional requirements of only significant classes compared to its result when classifying all non-functional requirements with 0.02 higher in precision and F1 and 0.04 higher in recall which could be explained by the characteristic of the used dataset which is now less imbalanced.

When changing the input data to functional requirements, the method M14 shows that it is still able to classify them even though the obtained result is not as good as the result when it classifies non-functional ones, with 0.04 lower in F1.

The remarkably better performance is observed as shown in the Table B.14 in the case of classifying functional requirements of only significant classes with 0.1 higher in F1, 0.12 higher in precision and 0.08 higher in recall compared to the result of classifying functional requirements of all classes. As discussed in the sections above, this achievement could be because the method M14 is trained with a less imbalanced training dataset compared to the original dataset.

The method M14 follows the similar trend as the method M13 with the result of mixed-type requirements classification being 0.06 lower than the result of functional requirements classification but approximately 0.1 lower than the result of non-functional requirements classification in F1.

6.2.16 | Method M15: Decision Tree + TF-IDF N-Grams Feature

The result of the method M15 is greater than the result of the original method [11]. As discussed above, the difference in the used datasets and the change in the distribution of requirements in our classes could be the reasons for this performance improvement.

The result of classifying non-functional requirements of significant classes of the method M15 is slightly higher than its result of classifying functional requirements of classes with 0.02 higher in F1. As discussed above, this slightly improved performance could be caused by the difference in the used dataset - switching from a severe imbalanced dataset to a slight imbalanced dataset.

The result of the method M15 in classifying functional requirements exceeded its own result in classifying non-functional one by approximately 0.09 in the F1 score, even though it is not intended for this task.

It seems that the method M15 classifies requirements better when we decrease the number of functional classes needed to be classified from 6 to 3 with 0.07 higher in F1 than the result acquired when classifying requirements from all non-functional classes. This improvement could be due to the change of the used datasets as we removed the minority functional classes from our dataset which makes the dataset less imbalanced than the original dataset.

The method M15 classifies mixed-type requirements worse than it classifies only functional requirements or only non-functional requirements with 0.05 difference in F1 between mixed-type requirements classification and non-functional requirements classification and 0.14 difference in F1 between mixed-type requirements classification and functional requirements as shown in the Table B.15.

6.2.17 | Method M16: Decision Tree + BoW Feature

Similarly to the method M15, the method M16 obtained a better result than the original method [11]. As explained above, this could happen because of the difference in the used datasets as well as the greater amount of non-functional requirements available in our dataset.

Following the same trend as other methods, the method M16 achieved a better result when classifying non-functional requirements of significant classes compared to when it classifies requirements of all non-functional classes with 0.07 higher in F1 and precision, 0.06 higher in recall. As discussed in the sections above, this improvement could be because of the difference in the used dataset and the drop in the number of the needed-to-classify classes.

As presented in the Table B.16, the method M16 shows that its ability in classifying functional requirements is better than its ability in classifying non-functional ones which is more than 0.05 in F1 score despite not being originally created for.

It can be seen that the result of classifying functional requirements of significant classes of the method M16 is higher than the result of classifying requirements of all functional classes with approximately 0.09 higher in precision, recall and F1. This better result can be due to the fact that the training dataset is less imbalanced than the original dataset and the number of classes that are needed to predict is dropped by half.

The result of the method M16 of mixed-type requirements classification is significantly lower than its results of functional requirements classification and of non-functional requirements classification.

6.2.18 | Method M17: SVM + Multiple Features

The method M17 did not achieve the similar results as of the original method [10]. This dissimilarity could be because Kurtanović and Maalej [10] filtered out the minority classes in the dataset, only kept and classified requirements into the 4 remaining classes that have a large and even amount of requirements. Retaining only major classes instead of all classes makes the used dataset balanced hence could lead to better results. Furthermore, the details of values of the used parameters of models are not provided which could also affect the final results.

When classifying only non-functional requirements of significant classes, the result of the method M17 is slightly higher than the result obtained when classifying requirements of all non-functional classes, with 0.03 higher in F1. This improvement could be caused by the change in the used dataset which is less imbalanced than the original one.

The result in classifying functional requirements of the method M17 surpasses its own result in classifying non-functional ones by approximately 0.26 in the F1 score, even though it is not intended for this kind of task.

The result of classifying only requirements from significant functional classes is also higher than the result of classifying requirements from all functional classes, with 0.07 higher in precision and 0.05 higher in recall and F1. As discussed above, this achievement could be because the number of classes needed to predict is reduced by half - from 6 classes to 3 classes as well as the filtered dataset is less imbalanced as we filtered out the minority classes and only kept significant classes.

In contrast to the result obtained in experiment 2, the result of experiment 3 of method M17 shows it is unable to classify instances into such a large number of classes - 17 classes to be precise. The result is significantly lower than the functional requirements classification result with 0.3 difference in F1 and also lower than the non-functional requirements classification with approximately 0.04 difference in F1.

6.2.19 | Method M18: OVR + Multiple Features

Similarly to the method M17, the method M18 did not produce the same result as the original method [10]. There could have been because only classes that have an evenly large amount of samples are kept which helps form a more balanced dataset. A balanced dataset could help the method get more accurate in classifying requirements because the model is not biased to any particular class. Besides, the possible difference in the configurations of the two methods could also be the cause for this inequality.

It can be seen that the method M18 classifies requirements better when we decrease the number of non-functional classes needed to be classified from 11 to 5 with 0.08 higher in precision, recall and F1 than the result acquired when classifying requirements from all non-functional classes. This better performance could be due to the change of the used datasets as we removed the minority functional classes from our dataset which makes the dataset less imbalanced than the original dataset and also due to the drop in the number of classes that the method M18 needs to predict.

Similarly to the method M17, the method M18 obtained a better result in classifying functional requirements than when it classifies non-functional requirements, with 0.2 higher in F1 score.

The result of the method M18 of classifying requirements of significant functional classes is also noticeably better than the result obtained from classifying requirements from all functional classes with 0.1 higher in F1 and 0.09 higher in precision and recall. As discussed above, the filtered dataset with only the majority classes and the decrease in the number of classes could be the reasons for this improvement.

The method M18 classifies mixed-type requirements worse than it classifies only functional requirements or non-functional requirements. The mixed-type requirements classification result is 0.03 lower than the functional requirements classification and 0.23 lower than the non-functional requirements classification.

6.2.20 | Method M19: ANN

The methods M19 under-performed the original methods [12]. Besides the difference in the used datasets, Baker, Deng, Chakraborty, *et al.* [12] only conducted requirements classification experiments on the 4 majority classes - namely Operational, Performance, Security and Usability instead of all classes. Furthermore, although Baker, Deng, Chakraborty, *et al.* [12] did provide some of the parameters used for defining a network, there are still necessary parameters that are unmentioned. These could be the reasons why our results differ from the paper's results.

When classifying requirements of only significant 5 non-functional classes instead of 11 non-functional classes, the result of the method M19 is better than the result obtained when classifying requirements of all non-functional classes, with 0.1 higher in F1 and 0.09 higher in precision and recall even though the classification ability is still not good. This increase could be due to the less imbalanced used dataset and the decrease in the number of needed-to-classify classes.

The method M19 shows its ability in classifying functional requirements with the result as 0.5877 in precision, 0.5676 in recall and 0.5712 in F1 score which surpassed the result obtained when classifying non-functional requirements.

The result of classifying only requirements from significant functional classes is also higher than the result of classifying requirements from all functional classes. As discussed above, this achievement could be because the number of classes needed to predict is reduced by half - from 6 classes to 3 classes as well as the filtered dataset is less imbalanced as we filtered out the minority classes and only kept significant classes.

6.2.21 | Method M20: CNN

The method M20 performed not as well as the original method [12]. This underperformance could be caused by the difference in the used dataset. Furthermore, Baker, Deng, Chakraborty, *et al.* [12] filtered out the minority classes (i.e. classes that have a significantly small amount compared to other classes), only kept 5 majority classes including maintainability, operational, performance, security and usability. This filter helps form a more balanced dataset as well as reduces the number of classes that a classifier has to predict which could help increase the performance of the method.

We witness the noticeable improvement in the result of the method M20 when classifying non-functional requirements of only significant classes compared to its result when classifying all non-functional requirements with 0.12 higher in F1, 0.1 higher in precision and 0.09 higher in recall which could be explained by the characteristic of the used dataset which is now less imbalanced.

It can be seen that the method M20 achieved a better result when classifying functional requirements than when classifying non-functional ones whose F1 score is 0.03 higher.

The better performance is observed as shown in the Table B.20 in the case of classifying functional requirements of only significant classes with 0.03 higher in F1 compared to the result of classifying functional requirements of all classes. As discussed in the sections above, this achievement could be because the method M20 is trained with a less imbalanced training dataset compared to the original dataset.

It is observed from the Table B.20 that the method M20 cannot extend its capability to classify requirements into 17 different classes with being 0.13 lower in F1 than the non-functional requirements classification result and 0.16 lower than functional requirements classification result.

6.2.22 | Method M21: XGBoost

While XGBoost shows a decent performance in the functional requirements classification problem, it does not perform well in other tasks, non-functional requirements and mixed-type requirements. Furthermore, it is shown that there is a huge difference in results between classifying non-functional requirements of significant classes and classifying requirements of all non-functional classes as well as between classifying functional requirements of significant classes and classifying requirements of all functional classes. This noticeable difference could be due to the change in the used datasets as we filtered out classes that have a small amount of requirements compared to others hence, the filtered dataset is less imbalanced than the original one.

6.2.23 | Method M22: Random Forest

Random Forest performs poorly in all functional requirements classification, non-functional requirements classification and mixed-type requirements classification. Similarly to the result of XGBoost, the results of classifying functional/non-functional requirements of significant classes is significantly higher than the results of classifying requirements of all functional/non-functional classes. While the sensitivity of random forest for imbalanced datasets and overfitting could be the reason for the poor performance, the better result in classifying requirements of significant classes could be caused by using a less imbalanced dataset than the original one.

6.3 | Discussion

6.3.1 | Discussion on Experiment 1

For methods that are originally intended for functional requirements classification, given our dataset, the method M2 (Ensemble Method with BoW feature) performs better than the method M1 (Ensemble Method with Tf-Idf feature) by nearly 10% in F1.

For methods that are originally created to classify non-functional requirements, given our dataset, the method M13 (SGD SVM with Tf-Idf N-Grams feature) performs the best with average accuracy, precision, recall and F1 as 0.7761, 0.7707, 0.7761, and 0.7555 respectively. On the other hand, the method M19 (ANN with BoW feature) has the poorest performance with average accuracy, precision, recall and F1 as 0.3731, 0.508, 0.3731, and 0.3647 respectively.

As we filtered the minority classes and trained these methods on a less imbalanced dataset, it is observed that these methods' results are higher compared to their results when being trained on the original dataset.

6.3.2 | Discussion on Experiment 2

Among two methods whose original purpose is to classify functional requirements, the method M1 shows that it is able to classify non-functional requirements with precision, recall, and F1 score 0.5352, 0.4925, and 0.4811 while the result of non-functional requirements classification of the method M2 is lower than the result obtained when classifying functional ones by 0.11 in F1 score.

The method M20 achieved the best performance in classifying functional requirements with precision, recall and F1 equal 0.755, 0.7422, and 0.7404 among other methods whose purpose is to classify non-functional requirements, not functional ones. On the other hand, the method M6 performed not as good as others with precision, recall and F1 score as 0.5448, 0.556, and 0.5337 respectively.

Similarly to the experiment 1, it is also observed that with a more balanced dataset of only requirements in significant classes, the results of these methods are improved compared to the results obtained with the original dataset.

6.3.3 | Discussion on Experiment 3

As shown in the Appendix B, the methods M9, M11, M12, M13, M14 show acceptable results for mixed-type requirements classification even though these results are not as good as the results for functional requirements classification or non-functional requirements classification. Among the mentioned methods, the method M13 performs classifying mixed-type requirements better than others with 0.6833 in precision, 0.6717 in recall, and 0.6577 in F1.

For other remaining methods, as their results are lower than the median values in precision, recall, and F1 as discussed in the Section 6.1, these methods are considered to be unsuitable for classifying mixed-type requirements. Among these methods, the method M19 obtained the worst result with only 0.4351 in precision, 0.313 in recall, and 0.3383 in F1.

6.3.4 | Discussion on Experiment 4

Overall, for the classification problem, given our dataset, XGBoost performs better than Random Forest. In all of the obtained results, these two methods received the highest score for the functional requirements classification task.

However, it can be observed that Random Forest and XGBoost are not effective when handling imbalanced datasets, in particular, our dataset, as there is a noticeable difference in the result between classifying requirements of significant classes and classifying requirements of all classes as well as the difference between functional requirements classification, non-functional requirements classification and mixed-type requirements classification.

6.4 | Threats to Validity

The threats to the validity of this research includes the manual labeling process and the configurations of models.

1181 out of 1838 requirements were manually labelled by Nguyen Nhu Thuy - a master's student with limited background working in the industry. The labeling person did her best based on her knowledge of the theoretical requirement engineering field. Because of this, the labeling process is subjective to some extent and may contain mislabeled requirements.

Our reconstruction of the original work's methods [9]–[12] is not perfectly identical because the actual implementations of these works are not publicly available, hence this could pose a threat to the validity of the research. To mitigate this threat, we closely followed the descriptions of the papers on how to build the methods from libraries usages to parameters methods.

7

Conclusions and Future Work

7.1 | Revisiting the Research Questions

As presented in the Section 1.3, the following research questions are established to drive the project to follow the critical analysis of the cross-applicability of the ML methods:

RQ1 How effective is it to use ML methods that were intended for functional requirements classification to classify non-functional requirements? And vice versa, how effective is it to use ML methods that were intended for non-functional requirements classification to classify functional requirements?

RQ2 How effective is it to use ML methods that were intended for either functional or non-functional requirements classification to classify both types of requirements?

RQ3 What other ML methods can also be used for the requirements classification problem?

7.1.1 | RQ1: Effectiveness of ML Methods with Cross-Applying Input

The result of the methods M5, M9, M11, M12, M13 and M14 in the Experiment 2 shows that these methods can be used to classify functional requirements even though they are not originally created for. The results of these methods when being used to classify functional requirements are slightly slower, less than 0.1 in the F1 score, than when they are used to classify non-functional requirements.

The ability of classifying functional requirements of the method M7, M8, M10, M15, M16, M17, M19, M20 exceeded its ability to classify non-functional requirements despite not being intended for which are recorded in the Table B.7, Table B.8, Table B.10, Table B.15, Table B.16, Table B.17, Table B.19, Table B.20 that the difference is in the range from 0.3 to 2.6 points.

It is also observed that for the remaining methods M1, M2, M3, M4, M6, their ability to classify requirements from the type that they are not originally intended for is not equal to their ability to classify requirements that are initially assigned for them.

In general, 15 out of 20 methods including the methods M1, M3, M5, M7, M8, M9, M10, M11, M12, M13, M15, M16, M17, M19, M20 demonstrate their cross-applicability (i.e. those that were intended for classifying functional requirements are able to classify non-functional ones and vice versa, those that were intended) through the results of Experiment 2.

7.1.2 | RQ2: Effectiveness of ML Methods with Mixed-Type Input

It can be inferred from Experiment 3 that the methods M9, M11, M12, M13, M14 can extend their use cases to classify mixed-type requirements. Even though the output classes are nearly doubled, their performances are still considered acceptable. As mentioned above, we evaluate the performance of a ML method whether it is good or not by collecting all of the results of the techniques in the original work, keeping best cases and drawing the box plots as shown in the Figure 6.2. Hence, from the calculated result, models with the results above 0.61 are considered acceptable models.

For other remaining methods namely the methods M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M15, M16, M17, M18, M19 and M20 based on their performance when being used to classify mixed type requirements, it seems unlikely that these models cannot be used to classify requirements into such a number of classes (17 classes) given our setups and our dataset.

7.1.3 | RQ3: Effectiveness of Other Methods

For the two new ML methods XGBoost and Random Forest, given our dataset, XGBoost and Random Forest perform decently in the functional requirements classification task. However, it is observed that they are not cross-applicable as the results obtained from functional requirements classification and non-functional requirements classification are not approximate. Furthermore, they are also unsuitable to classify mixed-type requirements as their results are much lower than the acceptable values.

In conclusion, given our dataset, as shown in the Table B.22 and Table B.21, both of the XGBoost and the Random Forest do not show their cross-applicability as well as the capabilities to classify mixed-type requirements.

7.2 | Contribution and Implication for Practitioner

For researchers, our study provides practical evidence for machine learning methods that are intended for classifying non-functional/functional requirements can be used for classifying the other (i.e. functional/non-functional requirements). This research demonstrates the effectiveness of some ML methods when being input with different types of data (functional requirements, non-functional requirements or mixed-type requirements). Thus, researchers should try the similar experiment to investigate the cross-applicability of their own machine learning methods for the requirements classification problem.

For practitioners, this research provides an insight into the cross-applicability of the machine learning methods. Using this insight as a reference, practitioners, if having a ML method at hand, can consider using it for the other requirement type classification. Through our results, it is also seen that, experimented ML methods learning capabilities are not enough to be used to classify mixed-type requirements. Hence, if practitioners want to scale up ML methods for mixed-type requirements classification, they should build a more dedicated method with careful consideration.

7.3 | Future Work

There are some recommendations for future research. With regard to the dataset, more requirements should be added so that the distribution of requirements for each class should be more evenly distributed (i.e. a more balanced dataset). Furthermore, it is necessary to double check the labels with domain experts to ensure the quality of the labels.

For future research questions, there is ongoing research on applying supervised machine learning methods on requirements classification which are not included in this research. These research could be subjected to future research. This research is mainly focused on supervised machine learning methods, hence, research questions on the cross-applicability of semi-supervised and unsupervised methods can be raised in the future research. It can be seen that the performance on mixed type requirements classification of machine learning methods is not very good, thus, how to create or adjust machine learning methods that perform better on mixed-type requirements classification could be subject to future research.

Appendix

A

Method Implementation

A.1 | POS N-Grams

Listing A.1: POS N-Grams Implementation written in Python

```python
class PosNgrams(_VectorizerMixin, BaseEstimator):
    def __init__(self, ngram_range):
        self.ngrams_dict = {}
        self.ngram_range = ngram_range

    def fit(self, list_str: list):
        ngrams_dict = {}
        list_str_w_tokenized_sent = DataPreprocessor.sent_tokenize(list_str)

        for str in list_str_w_tokenized_sent:
            for sent in str:
                tag_list = []
                text_tags = pos_tag(word_tokenize(sent))
                for item in text_tags:
                    tag_list.append(item[1])
                for i in range(len(tag_list) - self.ngram_range + 1):
                    g = ' '.join(tag_list[i:i + self.ngram_range])
                    ngrams_dict.setdefault(g, 0)
        self.ngrams_dict = ngrams_dict

    def fit_transform(self, list_str):
        self.fit(list_str)
        return self.transform(list_str)

    def transform(self, list_str: list):
        ngrams_array = []
        list_str_w_tokenized_sent = DataPreprocessor.sent_tokenize(list_str)
        for str in list_str_w_tokenized_sent:
            for sent in str:
                tag_list = []
                text_tags = pos_tag(word_tokenize(sent))

                for item in text_tags:
                    tag_list.append(item[1])
                for i in range(len(tag_list) - self.ngram_range + 1):
                    g = ' '.join(tag_list[i:i + self.ngram_range])
                    if self.ngrams_dict.get(g) is not None:
                        self.ngrams_dict[g] += 1
            ngrams_array.append(list(self.ngrams_dict.values()))
            self.ngrams_dict = dict.fromkeys(self.ngrams_dict.keys(), 0)

        return np.asarray(ngrams_array)

    def get_feature_names(self):
        return self.ngrams_dict.keys()
```

A.2 | Ensemble Method

Listing A.2: Ensemble Method Weight Calculation

```python
def calculate_weight(models_tp_list):
    weight = [0] * len(models_tp_list)
    for i in range(len(models_tp_list[0])):
        best_models = []
        current_best_tp = models_tp_list[0][i]
        for j in range(len(models_tp_list)):
            current_tp = models_tp_list[j][i]
            if current_tp > current_best_tp:
                current_best_tp = current_tp
                best_models = [j]
            elif current_tp == current_best_tp:
                best_models.append(j)
        for model in best_models:
            weight[model] += 1
    return weight
```

A.3 | ANN

Listing A.3: ANN Implementation

```python
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes, dropout_threshold_hidden
    ↪ ):
        super(NeuralNet, self).__init__()
        self.num_class = num_classes

        self.fc1 = nn.Linear(input_size, hidden_size)
        self.dropout_hidden = nn.Dropout(dropout_threshold_hidden)
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, text):
        x = self.fc1(text)
        x = torch.sigmoid(x)
        x = self.dropout_hidden(x)
        x = self.fc2(x)
        x = torch.sigmoid(x)
        return x
```

A.4 | CNN

Listing A.4: CNN Implementation

```python
class CNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, embedding_weights, dropout,
    ↪ n_filters, filter_sizes, n_classes):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.conv1 = nn.ModuleList([
            nn.Conv2d(in_channels=1,
                      out_channels=n_filters,
                      kernel_size=(fs, embedding_dim))
            for fs in filter_sizes
        ])

        self.fc1 = nn.Linear(len(filter_sizes) * n_filters, n_classes)
        self.dropout = nn.Dropout(dropout)

    def forward(self, text):
        x = self.embedding(text)
        x = x.unsqueeze(1)
        x = [f.relu(conv(x)).squeeze(3) for conv in self.conv1]
        x = [f.max_pool1d(i, i.size(2)).squeeze(2) for i in x]
        x = torch.cat(x, 1)
        x = self.dropout(x)
        x = self.fc1(x)
        return x
```

B Result Tables

B.1 | Method M1

Table B.1: Experiments results of Method M1

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	AC	—	0.3333	0.0444	0.0784	45
		EN	—	0.6733	0.7816	0.7234	87
		SO	—	0.6154	0.7619	0.6809	105
		All	0.6329	0.5831	0.6329	0.5821	—
	All FR Classes	AC	—	0.3750	0.0667	0.1132	45
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6889	0.7126	0.7006	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.5374	0.7524	0.6270	105
		All	0.5560	0.5144	0.5560	0.5092	—
		LF	—	0.5714	0.3636	0.4444	11

(continued on the next page)

Significant

Table B.1: *(continued)* Experiments results of Method M1

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		O	—	0.7241	0.4884	0.5833	43
		PE	—	0.4848	0.8421	0.6154	19
		SE	—	0.7143	0.3846	0.5000	65
		US	—	0.4030	0.8182	0.5400	33
		All	0.5439	0.6220	0.5439	0.5379	—
	All NFR Classes	A	—	0.5385	1.0000	0.7000	7
		FT	—	0.1429	0.3333	0.2000	3
		L	—	0.2000	0.2000	0.2000	5
		LE	—	0.5000	0.3636	0.4211	11
		MN	—	0.2500	0.3333	0.2857	6
		O	—	0.6364	0.3256	0.4308	43
		PE	—	0.5185	0.7368	0.6087	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.3333	0.3333	0.3333	6
		SE	—	0.6512	0.4308	0.5185	65
		US	—	0.4194	0.7879	0.5474	33
		All	0.4925	0.5352	0.4925	0.4811	—
		A	—	0.4000	0.8571	0.5455	7
		FT	—	0.1429	0.3333	0.2000	3
		L	—	0.1667	0.2000	0.1818	5
		LF	—	0.1429	0.3636	0.2051	11

(continued on the next page)

3 ll Classes

Table B.1: *(continued)* Experiments results of Method M1

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.2778	0.1163	0.1639	43
		PE	—	0.5455	0.6316	0.5854	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.4000	0.3333	0.3636	6
		SE	—	0.6667	0.1538	0.2500	65
		US	—	0.3750	0.6364	0.4719	33
		AC	—	0.2857	0.0444	0.0769	45
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.5135	0.6552	0.5758	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.3926	0.6095	0.4776	105
		All	0.4022	0.4018	0.4022	0.3554	—

B.1.0.1 | Method M2

Table B.2: Experiments results of Method M2

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
	Significant	AC	—	0.3571	0.1111	0.1695	45
		EN	—	0.8025	0.7471	0.7738	87
		SO	—	0.6549	0.8857	0.7530	105

(continued on the next page)

1

Table B.2: *(continued)* Experiments results of Method M2

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		All	0.6878	0.6525	0.6878	0.6499	—
	All FR Classes	AC	—	0.3636	0.1778	0.2388	45
		AT	—	0.3077	0.2500	0.2759	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.7952	0.7586	0.7765	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.6214	0.8286	0.7102	105
		All	0.6371	0.6012	0.6371	0.6073	—
2	Significant	LF	—	1.0000	0.0909	0.1667	11
		O	—	0.7436	0.6744	0.7073	43
		PE	—	0.5385	0.7368	0.6222	19
		SE	—	0.6327	0.4769	0.5439	65
		US	—	0.4286	0.7273	0.5393	33
		All	0.5789	0.6343	0.5789	0.5685	—
	All NFR Classes	A	—	0.7000	1.0000	0.8235	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.2500	0.2000	0.2222	5
		LE	—	0.5000	0.1818	0.2667	11
		MN	—	0.5000	0.1667	0.2500	6
		O	—	0.5946	0.5116	0.5500	43
		PE	—	0.5769	0.7895	0.6667	19

(continued on the next page)

Table B.2: *(continued)* Experiments results of Method M2

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.2000	0.3333	0.2500	6
		SE	—	0.6042	0.4462	0.5133	65
		US	—	0.3898	0.6970	0.5000	33
		All	0.5075	0.5200	0.5075	0.4925	—
3	All Classes	A	—	0.5455	0.8571	0.6667	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.3333	0.2000	0.2500	5
		LF	—	0.0500	0.0909	0.0645	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.3571	0.3488	0.3529	43
		PE	—	0.5000	0.4737	0.4865	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.1250	0.1667	0.1429	6
		SE	—	0.6842	0.2000	0.3095	65
		US	—	0.3396	0.5455	0.4186	33
		AC	—	0.1765	0.0667	0.0968	45
		AT	—	0.1538	0.1250	0.1379	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.5636	0.7126	0.6294	87
		PL	—	0.0000	0.0000	0.0000	3

(continued on the next page)

Table B.2: *(continued)* Experiments results of Method M2

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		SO	—	0.4126	0.5619	0.4758	105
		All	0.4130	0.4132	0.4130	0.3850	—

B.2 | Method M3

Table B.3: Experiments results of Method M3

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.3333	0.9091	0.4878	11
		O	—	1.0000	0.5814	0.7353	43
		PE	—	0.8750	0.3684	0.5185	19
		SE	—	0.6966	0.9538	0.8052	65
		US	—	0.9474	0.5455	0.6923	33
		All	0.7135	0.8178	0.7135	0.7136	—
	All NFR Classes	A	—	0.0000	0.0000	0.0000	7
		FT	—	0.0938	1.0000	0.1714	3
		L	—	0.0000	0.0000	0.0000	5
		LE	—	0.0000	0.0000	0.0000	11
		MN	—	0.0000	0.0000	0.0000	6
		O	—	0.7188	0.5349	0.6133	43
		PE	—	1.0000	0.4211	0.5926	19
		PO	—	0.0000	0.0000	0.0000	3

(continued on the next page)

Table B.3: *(continued)* Experiments results of Method M3

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		SC	—	0.0000	0.0000	0.0000	6
		SE	—	0.6117	0.9692	0.7500	65
		US	—	0.9615	0.7576	0.8475	33
		All	0.6070	0.6054	0.6070	0.5715	—
2	Significant	AC	—	0.4655	0.6000	0.5243	45
		EN	—	0.7297	0.6207	0.6708	87
		SO	—	0.6476	0.6476	0.6476	105
		All	0.6287	0.6432	0.6287	0.6327	—
	All FR Classes	AC	—	0.6875	0.2444	0.3607	45
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6526	0.7126	0.6813	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.5944	0.8095	0.6855	105
		All	0.6100	0.5796	0.6100	0.5694	—
		A	—	0.0000	0.0000	0.0000	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.0000	0.0000	0.0000	5
		LF	—	0.0000	0.0000	0.0000	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.7500	0.2093	0.3273	43

(continued on the next page)

3 All Classes

Table B.3: *(continued)* Experiments results of Method M3

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PE	—	1.0000	0.0526	0.1000	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.0000	0.0000	0.0000	6
		SE	—	0.6857	0.7385	0.7111	65
		US	—	1.0000	0.2727	0.4286	33
		AC	—	0.8750	0.1556	0.2642	45
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.4500	0.7241	0.5551	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.3832	0.7810	0.5141	105
		All	0.4761	0.5382	0.4761	0.4141	—

B.3 | Method M4

Table B.4: Experiments results of Method M4

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
	Significant	LF	—	0.5263	0.9091	0.6667	11
		O	—	0.9143	0.7442	0.8205	43
		PE	—	0.8095	0.8947	0.8500	19
		SE	—	0.9077	0.9077	0.9077	65

(continued on the next page)

Table B.4: *(continued)* Experiments results of Method M4

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		US	—	0.8065	0.7576	0.7812	33
		All	0.8363	0.8544	0.8363	0.8395	—
	All NFR Classes	A	—	1.0000	0.4286	0.6000	7
		FT	—	0.0909	0.6667	0.1600	3
		L	—	1.0000	0.2000	0.3333	5
		LE	—	0.3333	0.0909	0.1429	11
		MN	—	0.5000	0.1667	0.2500	6
		O	—	0.7442	0.7442	0.7442	43
		PE	—	0.6667	0.8421	0.7442	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	1.0000	0.1667	0.2857	6
		SE	—	0.8548	0.8154	0.8346	65
		US	—	0.7500	0.9091	0.8219	33
		All	0.6965	0.7459	0.6965	0.6898	—
2	Significant	AC	—	0.5185	0.6222	0.5657	45
		EN	—	0.7738	0.7471	0.7602	87
		SO	—	0.7273	0.6857	0.7059	105
		All	0.6962	0.7047	0.6962	0.6992	—
	All FR Classes	AC	—	0.4762	0.4444	0.4598	45
		AT	—	0.2500	0.0625	0.1000	16
		DE	—	0.0000	0.0000	0.0000	2

(continued on the next page)

Table B.4: *(continued)* Experiments results of Method M4

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		EN	—	0.6832	0.7931	0.7340	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.5909	0.6190	0.6047	105
		All	0.5985	0.5672	0.5985	0.5778	—
3	All Classes	A	—	1.0000	0.4286	0.6000	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	1.0000	0.2000	0.3333	5
		LF	—	0.6000	0.2727	0.3750	11
		MN	—	1.0000	0.4286	0.6000	7
		O	—	0.6591	0.6744	0.6667	43
		PE	—	0.7222	0.6842	0.7027	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.0000	0.0000	0.0000	6
		SE	—	0.6500	0.8000	0.7172	65
		US	—	0.7179	0.8485	0.7778	33
		AC	—	0.6250	0.6667	0.6452	45
		AT	—	0.5000	0.0625	0.1111	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6344	0.6782	0.6556	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5285	0.6190	0.5702	105

(continued on the next page)

Table B.4: *(continued)* Experiments results of Method M4

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		All	0.6239	0.6096	0.6239	0.6005	—

B.4 | Method M5

Table B.5: Experiments results of Method M5

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.8571	0.5455	0.6667	11
		O	—	0.7619	0.7442	0.7529	43
		PE	—	0.8125	0.6842	0.7429	19
		SE	—	0.8507	0.8769	0.8636	65
		US	—	0.7949	0.9394	0.8611	33
		All	0.8129	0.8138	0.8129	0.8092	—
	All NFR Classes	A	—	0.7500	0.4286	0.5455	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.8000	0.8000	0.8000	5
		LE	—	0.7143	0.4545	0.5556	11
		MN	—	0.4000	0.3333	0.3636	6
		O	—	0.7381	0.7209	0.7294	43
		PE	—	0.8235	0.7368	0.7778	19
		PO	—	1.0000	0.3333	0.5000	3
		SC	—	0.5000	0.5000	0.5000	6

(continued on the next page)

Table B.5: *(continued)* Experiments results of Method M5

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		SE	—	0.8028	0.8769	0.8382	65
		US	—	0.7750	0.9394	0.8493	33
		All	0.7512	0.7495	0.7512	0.7426	—
2	Significant	AC	—	0.7879	0.5778	0.6667	45
		EN	—	0.7391	0.7816	0.7598	87
		SO	—	0.7500	0.8000	0.7742	105
		All	0.7511	0.7532	0.7511	0.7485	—
	All FR Classes	AC	—	0.5283	0.6222	0.5714	45
		AT	—	0.4444	0.2500	0.3200	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6535	0.7586	0.7021	87
		PL	—	1.0000	0.5000	0.6667	4
		SO	—	0.7447	0.6667	0.7035	105
		All	0.6564	0.6561	0.6564	0.6504	—
3	All Classes	A	—	0.7500	0.8571	0.8000	7
		FT	—	1.0000	0.6667	0.8000	3
		L	—	1.0000	0.2000	0.3333	5
		LF	—	0.6250	0.4545	0.5263	11
		MN	—	0.5000	0.2857	0.3636	7
		O	—	0.7105	0.6279	0.6667	43
		PE	—	0.6667	0.4211	0.5161	19

(continued on the next page)

Table B.5: *(continued)* Experiments results of Method M5

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PO	—	1.0000	0.3333	0.5000	3
		SC	—	0.8000	0.6667	0.7273	6
		SE	—	0.6515	0.6615	0.6565	65
		US	—	0.7879	0.7879	0.7879	33
		AC	—	0.4889	0.4889	0.4889	45
		AT	—	0.6250	0.3125	0.4167	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6250	0.7471	0.6806	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5280	0.6286	0.5739	105
		All	0.6152	0.6191	0.6152	0.6069	—

B.5 | Method M6

Table B.6: Experiments results of Method M6

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
	Significant	LF	—	0.6667	0.5455	0.6000	11
		O	—	0.6667	0.6512	0.6588	43
		PE	—	0.6316	0.6316	0.6316	19
		SE	—	0.8438	0.8308	0.8372	65
		US	—	0.7838	0.8788	0.8286	33

(continued on the next page)

Table B.6: *(continued)* Experiments results of Method M6

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		All	0.7544	0.7527	0.7544	0.7526	—
	All NFR Classes	A	—	1.0000	0.5714	0.7273	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	1.0000	0.2000	0.3333	5
		LE	—	0.4444	0.7273	0.5517	11
		MN	—	0.4286	0.5000	0.4615	6
		O	—	0.6250	0.5814	0.6024	43
		PE	—	0.6818	0.7895	0.7317	19
		PO	—	1.0000	0.3333	0.5000	3
		SC	—	0.6667	0.3333	0.4444	6
		SE	—	0.7692	0.7692	0.7692	65
		US	—	0.6842	0.7879	0.7324	33
		All	0.6716	0.6909	0.6716	0.6654	—
2	Significant	AC	—	0.4706	0.7111	0.5664	45
		EN	—	0.6463	0.6092	0.6272	87
		SO	—	0.6322	0.5238	0.5729	105
		All	0.5907	0.6067	0.5907	0.5916	—
	All FR Classes	AC	—	0.4000	0.6222	0.4870	45
		AT	—	0.3333	0.0625	0.1053	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6355	0.7816	0.7010	87

(continued on the next page)

Table B.6: *(continued)* Experiments results of Method M6

Expr.	**Cond.**	**Class**	**Accuracy**	**Precision**	**Recall**	**F1**	**Support**
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.5949	0.4476	0.5109	105
		All	0.5560	0.5448	0.5560	0.5337	—
3	All Classes	A	—	1.0000	0.4286	0.6000	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	1.0000	0.4000	0.5714	5
		LF	—	0.7273	0.7273	0.7273	11
		MN	—	0.6000	0.4286	0.5000	7
		O	—	0.5682	0.5814	0.5747	43
		PE	—	0.6250	0.5263	0.5714	19
		PO	—	0.3333	0.3333	0.3333	3
		SC	—	1.0000	0.3333	0.5000	6
		SE	—	0.6667	0.6769	0.6718	65
		US	—	0.6250	0.7576	0.6849	33
		AC	—	0.3836	0.6222	0.4746	45
		AT	—	0.7500	0.1875	0.3000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.5526	0.4828	0.5153	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.4690	0.5048	0.4862	105
		All	0.5413	0.5610	0.5413	0.5357	—

B.6 | Method M7

Table B.7: Experiments results of Method M7

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.2703	0.9091	0.4167	11
		O	—	0.8519	0.5349	0.6571	43
		PE	—	1.0000	0.1053	0.1905	19
		SE	—	0.7143	0.9231	0.8054	65
		US	—	0.9048	0.5758	0.7037	33
		All	0.6667	0.7888	0.6667	0.6552	—
	All NFR Classes	A	—	0.0000	0.0000	0.0000	7
		FT	—	0.1667	1.0000	0.2857	3
		L	—	0.0000	0.0000	0.0000	5
		LE	—	0.0000	0.0000	0.0000	11
		MN	—	0.0000	0.0000	0.0000	6
		O	—	0.6129	0.4419	0.5135	43
		PE	—	0.6667	0.1053	0.1818	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.0000	0.0000	0.0000	6
		SE	—	0.5328	1.0000	0.6952	65
		US	—	0.8148	0.6667	0.7333	33
		All	0.5522	0.5027	0.5522	0.4765	—
2	Significant	AC	—	0.5750	0.5111	0.5412	45
		EN	—	0.7442	0.7356	0.7399	87

(continued on the next page)

Table B.7: *(continued)* Experiments results of Method M7

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		SO	—	0.6937	0.7333	0.7130	105
		All	0.6920	0.6897	0.6920	0.6902	—
	All FR Classes	AC	—	0.7000	0.3111	0.4308	45
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6700	0.7701	0.7166	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.6204	0.8095	0.7025	105
		All	0.6409	0.5982	0.6409	0.6003	—
3	All Classes	A	—	0.0000	0.0000	0.0000	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.0000	0.0000	0.0000	5
		LF	—	0.0000	0.0000	0.0000	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.7143	0.1163	0.2000	43
		PE	—	0.0000	0.0000	0.0000	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.0000	0.0000	0.0000	6
		SE	—	0.6164	0.6923	0.6522	65
		US	—	1.0000	0.1515	0.2632	33
		AC	—	0.9231	0.2667	0.4138	45

(continued on the next page)

Table B.7: *(continued)* Experiments results of Method M7

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.4710	0.7471	0.5778	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.3801	0.8000	0.5153	105
		All	0.4696	0.4918	0.4696	0.3971	—

B.7 | Method M8

Table B.8: Experiments results of Method M8

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.2439	0.9091	0.3846	11
		O	—	0.8621	0.5814	0.6944	43
		PE	—	1.0000	0.1579	0.2727	19
		SE	—	0.7683	0.9692	0.8571	65
		US	—	0.8750	0.4242	0.5714	33
		All	0.6725	0.8045	0.6725	0.6658	—
	All NFR Classes	A	—	0.0000	0.0000	0.0000	7
		FT	—	0.0571	0.6667	0.1053	3
		L	—	0.0000	0.0000	0.0000	5
		LE	—	0.0000	0.0000	0.0000	11

(continued on the next page)

Table B.8: *(continued)* Experiments results of Method M8

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		MN	—	0.0000	0.0000	0.0000	6
		O	—	0.5556	0.5814	0.5682	43
		PE	—	0.6667	0.1053	0.1818	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.0000	0.0000	0.0000	6
		SE	—	0.6129	0.8769	0.7215	65
		US	—	0.7600	0.5758	0.6552	33
		All	0.5224	0.5057	0.5224	0.4812	—
2	Significant	AC	—	0.5510	0.6000	0.5745	45
		EN	—	0.7738	0.7471	0.7602	87
		SO	—	0.7115	0.7048	0.7081	105
		All	0.7004	0.7039	0.7004	0.7019	—
	All FR Classes	AC	—	0.5909	0.2889	0.3881	45
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.7363	0.7701	0.7528	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.5874	0.8000	0.6774	105
		All	0.6332	0.5881	0.6332	0.5949	—
		A	—	0.0000	0.0000	0.0000	7
		FT	—	0.0000	0.0000	0.0000	3

(continued on the next page)

Table B.8: *(continued)* Experiments results of Method M8

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		L	—	0.0000	0.0000	0.0000	5
		LF	—	0.0000	0.0000	0.0000	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.5789	0.2558	0.3548	43
		PE	—	0.0000	0.0000	0.0000	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.0000	0.0000	0.0000	6
		SE	—	0.6567	0.6769	0.6667	65
		US	—	0.8571	0.1818	0.3000	33
		AC	—	0.9412	0.3556	0.5161	45
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.5398	0.7011	0.6100	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.3537	0.7714	0.4850	105
		All	0.4761	0.4833	0.4761	0.4255	—

B.8 | Method M9

Table B.9: Experiments results of Method M9

Expr.	**Cond.**	**Class**	**Accuracy**	**Precision**	**Recall**	**F1**	**Support**
1	Significant	LF	—	0.4500	0.8182	0.5806	11
		O	—	0.8485	0.6512	0.7368	43
		PE	—	0.7857	0.5789	0.6667	19
		SE	—	0.7945	0.8923	0.8406	65
		US	—	0.8710	0.8182	0.8438	33
		All	0.7778	0.7997	0.7778	0.7791	—
	All NFR Classes	A	—	0.5000	0.8571	0.6316	7
		FT	—	0.1667	0.3333	0.2222	3
		L	—	0.6667	0.4000	0.5000	5
		LE	—	0.4615	0.5455	0.5000	11
		MN	—	0.6000	0.5000	0.5455	6
		O	—	0.9600	0.5581	0.7059	43
		PE	—	0.7778	0.7368	0.7568	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.5714	0.6667	0.6154	6
		SE	—	0.8000	0.9231	0.8571	65
		US	—	0.6389	0.6970	0.6667	33
		All	0.7114	0.7392	0.7114	0.7089	—
2	Significant	AC	—	0.4590	0.6222	0.5283	45
		EN	—	0.7558	0.7471	0.7514	87
		SO	—	0.7889	0.6762	0.7282	105

(continued on the next page)

Table B.9: *(continued)* Experiments results of Method M9

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		All	0.6920	0.7141	0.6920	0.6988	—
	All FR Classes	AC	—	0.6429	0.6000	0.6207	45
		AT	—	0.2000	0.1875	0.1935	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6271	0.8506	0.7220	87
		PL	—	0.3333	0.2500	0.2857	4
		SO	—	0.8000	0.6095	0.6919	105
		All	0.6525	0.6642	0.6525	0.6472	—
3	All Classes	A	—	0.4118	1.0000	0.5833	7
		FT	—	0.6667	0.6667	0.6667	3
		L	—	0.4000	0.4000	0.4000	5
		LF	—	0.6667	0.5455	0.6000	11
		MN	—	0.5000	0.2857	0.3636	7
		O	—	0.6585	0.6279	0.6429	43
		PE	—	0.4783	0.5789	0.5238	19
		PO	—	0.5000	0.3333	0.4000	3
		SC	—	0.7500	0.5000	0.6000	6
		SE	—	0.6719	0.6615	0.6667	65
		US	—	0.7879	0.7879	0.7879	33
		AC	—	0.4792	0.5111	0.4946	45
		AT	—	0.5000	0.1875	0.2727	16

(continued on the next page)

Table B.9: *(continued)* Experiments results of Method M9

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6598	0.7356	0.6957	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.6737	0.6095	0.6400	105
		All	0.6174	0.6272	0.6174	0.6159	—

B.9 | Method M10

Table B.10: Experiments results of Method M10

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.1579	0.8182	0.2647	11
		O	—	0.6071	0.3953	0.4789	43
		PE	—	0.9286	0.6842	0.7879	19
		SE	—	0.8400	0.6462	0.7304	65
		US	—	0.6818	0.4545	0.5455	33
		All	0.5614	0.7169	0.5614	0.6079	—
	All NFR Classes	A	—	0.1875	0.4286	0.2609	7
		FT	—	0.1429	0.3333	0.2000	3
		L	—	1.0000	0.4000	0.5714	5
		LE	—	0.4286	0.2727	0.3333	11
		MN	—	0.0000	0.0000	0.0000	6

(continued on the next page)

Table B.10: *(continued)* Experiments results of Method M10

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		O	—	0.7105	0.6279	0.6667	43
		PE	—	1.0000	0.4211	0.5926	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.2500	0.3333	0.2857	6
		SE	—	0.7121	0.7231	0.7176	65
		US	—	0.4490	0.6667	0.5366	33
		All	0.5721	0.6150	0.5721	0.5718	—
2	Significant	AC	—	0.4615	0.8000	0.5854	45
		EN	—	0.8472	0.7011	0.7673	87
		SO	—	0.7701	0.6381	0.6979	105
		All	0.6920	0.7398	0.6920	0.7020	—
	All FR Classes	AC	—	0.6250	0.3333	0.4348	45
		AT	—	0.7500	0.1875	0.3000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6931	0.8046	0.7447	87
		PL	—	1.0000	0.2500	0.4000	4
		SO	—	0.6320	0.7524	0.6870	105
		All	0.6486	0.6594	0.6486	0.6289	—
		A	—	0.4545	0.7143	0.5556	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.3333	0.2000	0.2500	5

(continued on the next page)

sses

Table B.10: *(continued)* Experiments results of Method M10

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		LF	—	0.2632	0.4545	0.3333	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.3860	0.5116	0.4400	43
		PE	—	0.5556	0.2632	0.3571	19
		PO	—	0.5000	0.3333	0.4000	3
		SC	—	0.6000	0.5000	0.5455	6
		SE	—	0.5667	0.5231	0.5440	65
		US	—	0.2319	0.4848	0.3137	33
		AC	—	0.6154	0.3556	0.4507	45
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6351	0.5402	0.5839	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5172	0.5714	0.5430	105
		All	0.4674	0.4820	0.4674	0.4626	—

B.10 | Method M11

Table B.11: Experiments results of Method M11

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		LF	—	1.0000	0.3636	0.5333	11

(continued on the next page)

Significant

Table B.11: *(continued)* Experiments results of Method M11

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		O	—	0.7750	0.7209	0.7470	43
		PE	—	1.0000	0.6842	0.8125	19
		SE	—	0.8289	0.9692	0.8936	65
		US	—	0.8684	1.0000	0.9296	33
		All	0.8421	0.8530	0.8421	0.8315	—
	All NFR Classes	A	—	1.0000	0.5714	0.7273	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	1.0000	0.2000	0.3333	5
		LE	—	0.7500	0.2727	0.4000	11
		MN	—	0.0000	0.0000	0.0000	6
		O	—	0.7255	0.8605	0.7872	43
		PE	—	1.0000	0.7895	0.8824	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	1.0000	0.3333	0.5000	6
		SE	—	0.7722	0.9385	0.8472	65
		US	—	0.6889	0.9394	0.7949	33
		All	0.7662	0.7431	0.7662	0.7267	—
	Significant	AC	—	0.8148	0.4889	0.6111	45
		EN	—	0.8506	0.8506	0.8506	87
		SO	—	0.7724	0.9048	0.8333	105
		All	0.8059	0.8091	0.8059	0.7975	—

(continued on the next page)

2

Table B.11: *(continued)* Experiments results of Method M11

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
	All FR Classes	AC	—	0.6579	0.5556	0.6024	45
		AT	—	1.0000	0.0625	0.1176	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.7524	0.9080	0.8229	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.6957	0.7619	0.7273	105
		All	0.7143	0.7108	0.7143	0.6832	—
3	All Classes	A	—	1.0000	0.5714	0.7273	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	1.0000	0.2000	0.3333	5
		LF	—	1.0000	0.1818	0.3077	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.6341	0.6047	0.6190	43
		PE	—	0.7692	0.5263	0.6250	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.6667	0.3333	0.4444	6
		SE	—	0.7429	0.8000	0.7704	65
		US	—	0.8286	0.8788	0.8529	33
		AC	—	0.9474	0.4000	0.5625	45
		AT	—	1.0000	0.1250	0.2222	16
		DE	—	0.0000	0.0000	0.0000	2

(continued on the next page)

Table B.11: *(continued)* Experiments results of Method M11

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		EN	—	0.7400	0.8506	0.7914	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5000	0.8095	0.6182	105
		All	0.6630	0.6957	0.6630	0.6351	—

B.11 | Method M12

Table B.12: Experiments results of Method M12

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.6667	0.7273	0.6957	11
		O	—	0.7750	0.7209	0.7470	43
		PE	—	0.6957	0.8421	0.7619	19
		SE	—	0.8235	0.8615	0.8421	65
		US	—	0.8929	0.7576	0.8197	33
		All	0.7953	0.8004	0.7953	0.7955	—
	All NFR Classes	A	—	0.5714	0.5714	0.5714	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.5000	0.2000	0.2857	5
		LE	—	0.5000	0.1818	0.2667	11
		MN	—	0.7500	0.5000	0.6000	6
		O	—	0.7143	0.8140	0.7609	43

(continued on the next page)

Table B.12: *(continued)* Experiments results of Method M12

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PE	—	0.7619	0.8421	0.8000	19
		PO	—	0.5000	0.6667	0.5714	3
		SC	—	0.5000	0.6667	0.5714	6
		SE	—	0.9077	0.9077	0.9077	65
		US	—	0.7353	0.7576	0.7463	33
		All	0.7512	0.7436	0.7512	0.7395	—
2	Significant	AC	—	0.6304	0.6444	0.6374	45
		EN	—	0.8750	0.8851	0.8800	87
		SO	—	0.7961	0.7810	0.7885	105
		All	0.7932	0.7936	0.7932	0.7934	—
	All FR Classes	AC	—	0.5208	0.5556	0.5376	45
		AT	—	0.5455	0.3750	0.4444	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.8824	0.8621	0.8721	87
		PL	—	0.3333	0.2500	0.2857	4
		SO	—	0.7232	0.7714	0.7465	105
		All	0.7259	0.7189	0.7259	0.7209	—
3	ll Classes	A	—	0.8333	0.7143	0.7692	7
		FT	—	1.0000	0.0000	0.0000	3
		L	—	0.0000	0.2000	0.3333	5
		LF	—	0.6250	0.4545	0.5263	11

(continued on the next page)

Table B.12: *(continued)* Experiments results of Method M12

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		MN	—	0.5000	0.2857	0.3636	7
		O	—	0.4833	0.6744	0.5631	43
		PE	—	0.6364	0.7368	0.6829	19
		PO	—	0.5000	0.3333	0.4000	3
		SC	—	0.5000	0.5000	0.5000	6
		SE	—	0.7925	0.6462	0.7119	65
		US	—	0.8077	0.6364	0.7119	33
		AC	—	0.6500	0.5778	0.6118	45
		AT	—	0.5000	0.2500	0.3333	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.8125	0.7471	0.7784	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5248	0.7048	0.6016	105
		All	0.6348	0.6517	0.6348	0.6311	—

B.12 | Method M13

Table B.13: Experiments results of Method M13

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
	Significant	LF	—	0.7778	0.6364	0.7000	11
		O	—	0.7500	0.6977	0.7229	43

(continued on the next page)

Table B.13: *(continued)* Experiments results of Method M13

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PE	—	0.8750	0.7368	0.8000	19
		SE	—	0.8356	0.9385	0.8841	65
		US	—	0.8485	0.8485	0.8485	33
		All	0.8187	0.8172	0.8187	0.8155	—
	All NFR Classes	A	—	0.6364	1.0000	0.7778	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	1.0000	0.2000	0.3333	5
		LE	—	0.5556	0.4545	0.5000	11
		MN	—	1.0000	0.3333	0.5000	6
		O	—	0.8500	0.7907	0.8193	43
		PE	—	0.7778	0.7368	0.7568	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.7500	0.5000	0.6000	6
		SE	—	0.8429	0.9077	0.8741	65
		US	—	0.6889	0.9394	0.7949	33
		All	0.7761	0.7707	0.7761	0.7555	—
2	Significant	AC	—	.6897	0.4444	0.5405	45
		EN	—	0.8065	0.8621	0.8333	87
		SO	—	0.7913	0.8667	0.8273	105
		All	0.7848	0.7776	0.7848	0.7751	—
	All FR Classes	AC	—	0.6136	0.6000	0.6067	45

(continued on the next page)

Table B.13: *(continued)* Experiments results of Method M13

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		AT	—	0.5000	0.3125	0.3846	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.7248	0.9080	0.8061	87
		PL	—	0.2000	0.2500	0.2222	4
		SO	—	0.7802	0.6762	0.7245	105
		All	0.7066	0.7004	0.7066	0.6971	—
3	All Classes	A	—	1.0000	0.8571	0.9231	7
		FT	—	1.0000	0.6667	0.8000	3
		L	—	0.3333	0.2000	0.2500	5
		LF	—	0.6667	0.1818	0.2857	11
		MN	—	1.0000	0.1429	0.2500	7
		O	—	0.6429	0.6279	0.6353	43
		PE	—	0.7333	0.5789	0.6471	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.6667	0.3333	0.4444	6
		SE	—	0.7937	0.7692	0.7813	65
		US	—	0.7317	0.9091	0.8108	33
		AC	—	0.6923	0.4000	0.5070	45
		AT	—	0.7500	0.5625	0.6429	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.7727	0.7816	0.7771	87

(continued on the next page)

Table B.13: *(continued)* Experiments results of Method M13

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5395	0.7810	0.6381	105
		All	0.6717	0.6833	0.6717	0.6577	—

B.13 | Method M14

Table B.14: Experiments results of Method M14

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.5000	0.1818	0.2667	11
		O	—	0.7209	0.7209	0.7209	43
		PE	—	0.7222	0.6842	0.7027	19
		SE	—	0.8243	0.9385	0.8777	65
		US	—	0.7812	0.7576	0.7692	33
		All	0.7719	0.7578	0.7719	0.7586	—
	All NFR Classes	A	—	0.6667	0.5714	0.6154	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.6667	0.4000	0.5000	5
		LE	—	0.5000	0.1818	0.2667	11
		MN	—	0.6667	0.3333	0.4444	6
		O	—	0.6981	0.8605	0.7708	43
		PE	—	0.8421	0.8421	0.8421	19

(continued on the next page)

Table B.14: *(continued)* Experiments results of Method M14

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PO	—	0.5000	0.3333	0.4000	3
		SC	—	0.5000	0.5000	0.5000	6
		SE	—	0.9062	0.8923	0.8992	65
		US	—	0.6410	0.7576	0.6944	33
		All	0.7463	0.7367	0.7463	0.7319	—
2	Significant	AC	—	0.7647	0.5778	0.6582	45
		EN	—	0.8642	0.8046	0.8333	87
		SO	—	0.7459	0.8667	0.8018	105
		All	0.7890	0.7929	0.7890	0.7861	—
	All FR Classes	AC	—	0.5581	0.5333	0.5455	45
		AT	—	0.5000	0.1875	0.2727	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.8172	0.8736	0.8444	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.6810	0.7524	0.7149	105
		All	0.7027	0.6785	0.7027	0.6851	—
3	All Classes	A	—	1.0000	0.5714	0.7273	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.2500	0.2000	0.2222	5
		LF	—	0.8000	0.3636	0.5000	11
		MN	—	0.5000	0.1429	0.2222	7

(continued on the next page)

Table B.14: *(continued)* Experiments results of Method M14

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		O	—	0.4394	0.6744	0.5321	43
		PE	—	0.8667	0.6842	0.7647	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	1.0000	0.5000	0.6667	6
		SE	—	0.8627	0.6769	0.7586	65
		US	—	0.7667	0.6970	0.7302	33
		AC	—	0.6000	0.6000	0.6000	45
		AT	—	0.3000	0.1875	0.2308	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.7500	0.7241	0.7368	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5396	0.7143	0.6148	105
		All	0.6304	0.6456	0.6304	0.6248	—

B.14 | Method M15

Table B.15: Experiments results of Method M15

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
	Significant	LF	—	0.3077	0.3636	0.3333	11
		O	—	0.6122	0.6977	0.6522	43
		PE	—	0.7778	0.7368	0.7568	19

(continued on the next page)

Table B.15: *(continued)* Experiments results of Method M15

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		SE	—	0.7925	0.6462	0.7119	65
		US	—	0.4737	0.5455	0.5070	33
		All	0.6316	0.6528	0.6316	0.6380	—
	All NFR Classes	A	—	0.2222	0.2857	0.2500	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.6667	0.4000	0.5000	5
		LE	—	0.0667	0.0909	0.0769	11
		MN	—	0.5000	0.3333	0.4000	6
		O	—	0.6486	0.5581	0.6000	43
		PE	—	0.6154	0.4211	0.5000	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.1333	0.3333	0.1905	6
		SE	—	0.7869	0.7385	0.7619	65
		US	—	0.4884	0.6364	0.5526	33
		All	0.5473	0.5785	0.5473	0.5557	—
2	Significant	AC	—	0.4091	0.4000	0.4045	45
		EN	—	0.8415	0.7931	0.8166	87
		SO	—	0.7117	0.7524	0.7315	105
		All	0.7004	0.7019	0.7004	0.7006	—
	All FR Classes	AC	—	0.6842	0.5778	0.6265	45
		AT	—	0.6364	0.4375	0.5185	16

(continued on the next page)

Table B.15: *(continued)* Experiments results of Method M15

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.8977	0.9080	0.9029	87
		PL	—	0.5000	0.2500	0.3333	4
		SO	—	0.7094	0.7905	0.7477	105
		All	0.7568	0.7551	0.7568	0.7524	—
3	All Classes	A	—	0.2941	0.7143	0.4167	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.0000	0.0000	0.0000	5
		LF	—	1.0000	0.0909	0.1667	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.4167	0.4651	0.4396	43
		PE	—	0.6154	0.4211	0.5000	19
		PO	—	0.1429	0.3333	0.2000	3
		SC	—	0.1429	0.1667	0.1538	6
		SE	—	0.6182	0.5231	0.5667	65
		US	—	0.4412	0.4545	0.4478	33
		AC	—	0.2917	0.3111	0.3011	45
		AT	—	0.5000	0.3125	0.3846	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.7045	0.7126	0.7086	87
		PL	—	0.0000	0.0000	0.0000	3

(continued on the next page)

Table B.15: *(continued)* Experiments results of Method M15

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		SO	—	0.5164	0.6000	0.5551	105
		All	0.4978	0.5116	0.4978	0.4911	—

B.15 | Method M16

Table B.16: Experiments results of Method M16

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.3750	0.2727	0.3158	11
		O	—	0.5682	0.5814	0.5747	43
		PE	—	0.8000	0.8421	0.8205	19
		SE	—	0.7812	0.7692	0.7752	65
		US	—	0.6000	0.6364	0.6176	33
		All	0.6725	0.6686	0.6725	0.6699	—
	All NFR Classes	A	—	0.5000	0.5714	0.5333	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.3333	0.4000	0.3636	5
		LE	—	0.3333	0.3636	0.3478	11
		MN	—	0.0000	0.0000	0.0000	6
		O	—	0.6286	0.5116	0.5641	43
		PE	—	0.7500	0.4737	0.5806	19
		PO	—	0.0000	0.0000	0.0000	3

(continued on the next page)

Table B.16: *(continued)* Experiments results of Method M16

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		SC	—	0.1429	0.1667	0.1538	6
		SE	—	0.6712	0.7538	0.7101	65
		US	—	0.5682	0.7576	0.6494	33
		All	0.5771	0.5639	0.5771	0.5631	—
2	Significant	AC	—	0.6207	0.4000	0.4865	45
		EN	—	0.8025	0.7471	0.7738	87
		SO	—	0.6929	0.8381	0.7586	105
		All	0.7215	0.7194	0.7215	0.7125	—
	All FR Classes	AC	—	0.5000	0.4222	0.4578	45
		AT	—	0.5000	0.5625	0.5294	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.8652	0.8851	0.8750	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.6579	0.7143	0.6849	105
		All	0.6950	0.6751	0.6950	0.6838	—
3	All Classes	A	—	0.3333	0.7143	0.4545	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.6667	0.4000	0.5000	5
		LF	—	0.4545	0.4545	0.4545	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.3953	0.3953	0.3953	43

(continued on the next page)

Table B.16: *(continued)* Experiments results of Method M16

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PE	—	0.6190	0.6842	0.6500	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.3333	0.1667	0.2222	6
		SE	—	0.6066	0.5692	0.5873	65
		US	—	0.4359	0.5152	0.4722	33
		AC	—	0.4211	0.3556	0.3855	45
		AT	—	0.6667	0.2500	0.3636	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.7931	0.7931	0.7931	87
		PL	—	0.2500	0.3333	0.2857	3
		SO	—	0.5854	0.6857	0.6316	105
		All	0.5630	0.5567	0.5630	0.5532	—

B.16 | Method M17

Table B.17: Experiments results of Method M17

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
	Significant	LF	—	0.3333	0.2727	0.3000	11
		O	—	0.5000	0.6512	0.5657	43
		PE	—	0.6000	0.4737	0.5294	19
		SE	—	0.6047	0.4000	0.4815	65

(continued on the next page)

Table B.17: *(continued)* Experiments results of Method M17

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		US	—	0.4375	0.6364	0.5185	33
		All	0.5088	0.5281	0.5088	0.5034	—
	All NFR Classes	A	—	0.3636	0.5714	0.4444	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.5000	0.2000	0.2857	5
		LE	—	0.2500	0.1818	0.2105	11
		MN	—	0.0000	0.0000	0.0000	6
		O	—	0.4310	0.5814	0.4950	43
		PE	—	0.4444	0.4211	0.4324	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.4000	0.3333	0.3636	6
		SE	—	0.5909	0.4000	0.4771	65
		US	—	0.3704	0.6061	0.4598	33
		All	0.4378	0.4368	0.4378	0.4215	—
2	Significant	AC	—	0.7391	0.3778	0.5000	45
		EN	—	0.8784	0.7471	0.8075	87
		SO	—	0.7000	0.9333	0.8000	105
		All	0.7595	0.7729	0.7595	0.7458	—
	All FR Classes	AC	—	0.6818	0.3333	0.4478	45
		AT	—	0.2778	0.3125	0.2941	16
		DE	—	0.0000	0.0000	0.0000	2

(continued on the next page)

Table B.17: *(continued)* Experiments results of Method M17

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		EN	—	0.8784	0.7471	0.8075	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.6690	0.9238	0.7760	105
		All	0.7027	0.7019	0.7027	0.6818	—
3	All Classes	A	—	0.2857	0.5714	0.3810	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.5000	0.2000	0.2857	5
		LF	—	0.0541	0.1818	0.0833	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.3667	0.5116	0.4272	43
		PE	—	0.3333	0.4211	0.3721	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.2000	0.3333	0.2500	6
		SE	—	0.3617	0.2615	0.3036	65
		US	—	0.2346	0.5758	0.3333	33
		AC	—	0.4667	0.1556	0.2333	45
		AT	—	0.2308	0.1875	0.2069	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6364	0.5632	0.5976	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5584	0.4095	0.4725	105

(continued on the next page)

Table B.17: *(continued)* Experiments results of Method M17

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		All	0.3848	0.4312	0.3848	0.3872	—

B.17 | Method M18

Table B.18: Experiments results of Method M18

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.4444	0.3636	0.4000	11
		O	—	0.5909	0.6047	0.5977	43
		PE	—	0.6250	0.5263	0.5714	19
		SE	—	0.6304	0.4462	0.5225	65
		US	—	0.4107	0.6970	0.5169	33
		All	0.5380	0.5655	0.5380	0.5379	—
	All NFR Classes	A	—	0.4000	0.5714	0.4706	7
		FT	—	0.5000	0.3333	0.4000	3
		L	—	0.0000	0.0000	0.0000	5
		LE	—	0.5000	0.3636	0.4211	11
		MN	—	0.0000	0.0000	0.0000	6
		O	—	0.4889	0.5116	0.5000	43
		PE	—	0.4545	0.5263	0.4878	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.4286	0.5000	0.4615	6

(continued on the next page)

Table B.18: *(continued)* Experiments results of Method M18

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		SE	—	0.6410	0.3846	0.4808	65
		US	—	0.3966	0.6970	0.5055	33
		All	0.4577	0.4815	0.4577	0.4507	—
2	Significant	AC	—	0.7857	0.4889	0.6027	45
		EN	—	0.8857	0.7126	0.7898	87
		SO	—	0.6978	0.9238	0.7951	105
		All	0.7637	0.7835	0.7637	0.7566	—
	All FR Classes	AC	—	0.7500	0.2667	0.3934	45
		AT	—	0.2308	0.3750	0.2857	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.8750	0.7241	0.7925	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.6414	0.8857	0.7440	105
		All	0.6718	0.6985	0.6718	0.6538	—
3	All Classes	A	—	0.3077	0.5714	0.4000	7
		FT	—	0.3333	0.3333	0.3333	3
		L	—	0.2500	0.2000	0.2222	5
		LF	—	0.0556	0.1818	0.0851	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.4000	0.3721	0.3855	43
		PE	—	0.3889	0.3684	0.3784	19

(continued on the next page)

Table B.18: *(continued)* Experiments results of Method M18

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.2143	0.5000	0.3000	6
		SE	—	0.4130	0.2923	0.3423	65
		US	—	0.3662	0.7879	0.5000	33
		AC	—	0.3529	0.1333	0.1935	45
		AT	—	0.1500	0.1875	0.1667	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6977	0.6897	0.6936	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5610	0.4381	0.4920	105
		All	0.4217	0.4515	0.4217	0.4208	—

B.18 | Method M19

Table B.19: Experiments results of Method M19

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
	Significant	LF	—	0.3636	0.3636	0.3636	11
		O	—	0.6087	0.3256	0.4242	43
		PE	—	0.2745	0.7368	0.4000	19
		SE	—	0.8148	0.3385	0.4783	65
		US	—	0.4237	0.7576	0.5435	33

(continued on the next page)

1

Table B.19: *(continued)* Experiments results of Method M19

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		All	0.4620	0.5985	0.4620	0.4612	—
	All NFR Classes	A	—	0.2941	0.7143	0.4167	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.4167	1.0000	0.5882	5
		LE	—	0.2222	0.1818	0.2000	11
		MN	—	0.2143	0.5000	0.3000	6
		O	—	0.4545	0.1163	0.1852	43
		PE	—	0.3333	0.5263	0.4082	19
		PO	—	0.0769	0.3333	0.1250	3
		SC	—	0.1538	0.3333	0.2105	6
		SE	—	0.8400	0.3231	0.4667	65
		US	—	0.3818	0.6364	0.4773	33
		All	0.3731	0.5080	0.3731	0.3647	—
2	Significant	AC	—	0.4000	0.3556	0.3765	45
		EN	—	0.8000	0.5977	0.6842	87
		SO	—	0.6136	0.7714	0.6835	105
		All	0.6287	0.6415	0.6287	0.6255	—
	All FR Classes	AC	—	0.2581	0.3556	0.2991	45
		AT	—	0.3333	0.1875	0.2400	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.7500	0.7586	0.7543	87

(continued on the next page)

Table B.19: *(continued)* Experiments results of Method M19

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		PL	—	1.0000	0.2500	0.4000	4
		SO	—	0.6289	0.5810	0.6040	105
		All	0.5676	0.5877	0.5676	0.5712	—
3	All Classes	A	—	0.2500	0.7143	0.3704	7
		FT	—	0.0370	0.3333	0.0667	3
		L	—	0.0000	0.0000	0.0000	5
		LF	—	0.0769	0.2727	0.1200	11
		MN	—	0.2000	0.4286	0.2727	7
		O	—	0.2188	0.1628	0.1867	43
		PE	—	0.3548	0.5789	0.4400	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.0909	0.5000	0.1538	6
		SE	—	0.6522	0.2308	0.3409	65
		US	—	0.3673	0.5455	0.4390	33
		AC	—	0.3600	0.2000	0.2571	45
		AT	—	0.2000	0.1875	0.1935	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6500	0.4483	0.5306	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.4655	0.2571	0.3313	105
		All	0.3130	0.4351	0.3130	0.3383	—

B.19 | Method M20

Table B.20: Experiments results of Method M20

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	1.0000	0.5000	0.6667	10
		O	—	0.9000	0.6750	0.7714	40
		PE	—	1.0000	0.7222	0.8387	18
		SE	—	0.8592	0.9531	0.9037	64
		US	—	0.6341	0.9286	0.7536	28
		All	0.8250	0.8546	0.8250	0.8222	—
	All NFR Classes	A	—	0.7500	0.8571	0.8000	7
		FT	—	0.0000	0.0000	0.0000	2
		L	—	1.0000	0.4000	0.5714	5
		LE	—	0.7500	0.2727	0.4000	11
		MN	—	0.5000	0.1667	0.2500	6
		O	—	0.5625	0.6750	0.6136	40
		PE	—	1.0000	0.7222	0.8387	18
		PO	—	1.0000	0.3333	0.5000	3
		SC	—	1.0000	0.1667	0.2857	6
		SE	—	0.8571	0.9231	0.8889	65
		US	—	0.6279	0.9310	0.7500	29
		All	0.7344	0.7548	0.7344	0.7122	—
2	Significant	AC	—	0.6346	0.7857	0.7021	42
		EN	—	0.8370	0.9059	0.8701	85

(continued on the next page)

Table B.20: *(continued)* Experiments results of Method M20

Expr.	**Cond.**	**Class**	**Accuracy**	**Precision**	**Recall**	**F1**	**Support**
		SO	—	0.8500	0.7010	0.7684	97
		All	0.7946	0.8047	0.7946	0.7945	—
	All FR Classes	AC	—	0.4925	0.7674	0.6000	43
		AT	—	0.9000	0.5625	0.6923	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.8690	0.8391	0.8538	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.7895	0.7212	0.7538	104
		All	0.7422	0.7550	0.7422	0.7404	—
3	All Classes	A	—	1.0000	0.2857	0.4444	7
		FT	—	1.0000	0.3333	0.5000	3
		L	—	1.0000	0.2000	0.3333	5
		LF	—	1.0000	0.2727	0.4286	11
		MN	—	1.0000	0.3333	0.5000	6
		O	—	0.4872	0.4524	0.4691	42
		PE	—	0.6667	0.4444	0.5333	18
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.5000	0.1667	0.2500	6
		SE	—	0.7778	0.7424	0.7597	66
		US	—	0.5676	0.6562	0.6087	32
		AC	—	0.3913	0.4615	0.4235	39

(continued on the next page)

Table B.20: *(continued)* Experiments results of Method M20

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		AT	—	0.5000	0.2500	0.3333	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6947	0.7765	0.7333	85
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.5441	0.7184	0.6192	103
		All	0.6004	0.6145	0.6004	0.5857	—

B.20 | XGBoost

Table B.21: Experiments results of XGBoost

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
	Significant	LF	—	0.2000	0.4545	0.2778	11
		O	—	0.4750	0.4419	0.4578	43
		PE	—	0.5714	0.6316	0.6000	19
		SE	—	0.5625	0.2769	0.3711	65
		US	—	0.3774	0.6061	0.4651	33
		All	0.4327	0.4824	0.4327	0.4305	—
1	All NFR Classes	A	—	0.5455	0.8571	0.6667	7
		FT	—	0.0909	0.3333	0.1429	3
		L	—	0.0000	0.0000	0.0000	5
		LE	—	0.2500	0.2727	0.2609	11

(continued on the next page)

Table B.21: *(continued)* Experiments results of XGBoost

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		MN	—	0.1429	0.1667	0.1538	6
		O	—	0.5435	0.5814	0.5618	43
		PE	—	0.4800	0.6316	0.5455	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.5000	0.1667	0.2500	6
		SE	—	0.7059	0.3692	0.4848	65
		US	—	0.4118	0.6364	0.5000	33
		All	0.4677	0.5107	0.4677	0.4623	—
2	Significant	AC	—	0.7407	0.4444	0.5556	45
		EN	—	0.8551	0.6782	0.7564	87
		SO	—	0.6809	0.9143	0.7805	105
		All	0.7384	0.7562	0.7384	0.7289	—
	All FR Classes	AC	—	0.7037	0.4222	0.5278	45
		AT	—	0.3043	0.4375	0.3590	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.8158	0.7126	0.7607	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.6617	0.8381	0.7395	105
		All	0.6795	0.6833	0.6795	0.6692	—
		A	—	0.4444	0.5714	0.5000	7
		FT	—	0.1111	0.3333	0.1667	3

(continued on the next page)

Table B.21: *(continued)* Experiments results of XGBoost

Expr.	**Cond.**	**Class**	**Accuracy**	**Precision**	**Recall**	**F1**	**Support**
		L	—	0.0000	0.0000	0.0000	5
		LF	—	0.0500	0.1818	0.0784	11
		MN	—	0.1000	0.1429	0.1176	7
		O	—	0.3902	0.3721	0.3810	43
		PE	—	0.5625	0.4737	0.5143	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.2857	0.3333	0.3077	6
		SE	—	0.4167	0.2308	0.2970	65
		US	—	0.2361	0.5152	0.3238	33
		AC	—	0.4737	0.2000	0.2812	45
		AT	—	0.4286	0.3750	0.4000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.6087	0.6437	0.6257	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.5111	0.4381	0.4718	105
		All	0.4000	0.4425	0.4000	0.4059	—

B.21 | Random Forest

Table B.22: Experiments results of Random Forest

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
1	Significant	LF	—	0.4615	0.5455	0.5000	11
		O	—	0.5172	0.3488	0.4167	43
		PE	—	0.6923	0.4737	0.5625	19
		SE	—	0.5556	0.6154	0.5839	65
		US	—	0.5455	0.7273	0.6234	33
		All	0.5497	0.5531	0.5497	0.5417	—
	All NFR Classes	A	—	0.4286	0.4286	0.4286	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.0000	0.0000	0.0000	5
		LE	—	0.0000	0.0000	0.0000	11
		MN	—	0.0000	0.0000	0.0000	6
		O	—	0.4545	0.3488	0.3947	43
		PE	—	0.5882	0.5263	0.5556	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.0000	0.0000	0.0000	6
		SE	—	0.4750	0.5846	0.5241	65
		US	—	0.3934	0.7273	0.5106	33
		All	0.4478	0.3860	0.4478	0.4052	—
2	Significant	AC	—	0.7273	0.3556	0.4776	45
		EN	—	0.9362	0.5057	0.6567	87
		SO	—	0.5952	0.9524	0.7326	105

(continued on the next page)

Table B.22: *(continued)* Experiments results of Random Forest

Expr.	Cond.	Class	Accuracy	Precision	Recall	F1	Support
		All	0.6751	0.7455	0.6751	0.6563	—
	All FR Classes	AC	—	0.8889	0.1778	0.2963	45
		AT	—	0.0000	0.0000	0.0000	16
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.9565	0.5057	0.6617	87
		PL	—	0.0000	0.0000	0.0000	4
		SO	—	0.5172	1.0000	0.6818	105
		All	0.6062	0.6854	0.6062	0.5501	—
3	All Classes	A	—	0.5714	0.5714	0.5714	7
		FT	—	0.0000	0.0000	0.0000	3
		L	—	0.0000	0.0000	0.0000	5
		LF	—	0.0000	0.0000	0.0000	11
		MN	—	0.0000	0.0000	0.0000	7
		O	—	0.5000	0.1628	0.2456	43
		PE	—	0.5000	0.4211	0.4571	19
		PO	—	0.0000	0.0000	0.0000	3
		SC	—	0.1667	0.1667	0.1667	6
		SE	—	0.3462	0.1385	0.1978	65
		US	—	0.5926	0.4848	0.5333	33
		AC	—	0.5294	0.2000	0.2903	45
		AT	—	0.0000	0.0000	0.0000	16

(continued on the next page)

Table B.22: *(continued)* Experiments results of Random Forest

Expr.	**Cond.**	**Class**	**Accuracy**	**Precision**	**Recall**	**F1**	**Support**
		DE	—	0.0000	0.0000	0.0000	2
		EN	—	0.5368	0.5862	0.5604	87
		PL	—	0.0000	0.0000	0.0000	3
		SO	—	0.3420	0.7524	0.4702	105
		All	0.4000	0.4011	0.4000	0.3607	—

Bibliography

[1] I. Sommerville, *Software Engineering*, 9th. USA: Addison-Wesley Publishing Company, 2010, ISBN: 0137035152 (cit. on pp. 3, 8).

[2] S. Lauesen, *Software Requirements-Styles and Techniques*. Jan. 2002 (cit. on pp. 3, 9).

[3] J. Sayyad Shirabad and T. Menzies, *The PROMISE Repository of Software Engineering Databases.* School of Information Technology and Engineering, University of Ottawa, Canada, 2005. [Online]. Available: http://promise.site.uottawa.ca/SERepository (visited on 02/2021) (cit. on pp. 3, 9, 17, 18, 22).

[4] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943 (cit. on p. 6).

[5] K. Fukushima and S. Miyake, "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position," *Pattern Recognition*, vol. 15, no. 6, pp. 455–469, 1982, ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(82)90024-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0031320382900243 (cit. on p. 8).

[6] A. Ghazarian, "Characterization of functional software requirements space: The law of requirements taxonomic growth," in *2012 20th IEEE International Requirements Engineering Conference (RE)*, 2012, pp. 241–250. DOI: 10.1109/RE.2012.6345810 (cit. on p. 9).

[7] G. Koelsch, *Requirements Writing for System Engineering*. Apress, 2016, ISBN: 9781484220993. [Online]. Available: https://books.google.nl/books?id=2OtNDQAAQBAJ (cit. on p. 9).

[8] P. Jain, K. Verma, A. Kass, and R. G. Vasquez, “Automated review of natural language requirements documents: Generating useful warnings with user-extensible glossaries driving a simple state machine,” in *Proceedings of the 2nd India Software Engineering Conference*, ser. ISEC ’09, Pune, India: Association for Computing Machinery, 2009, pp. 37–46, ISBN: 9781605584263. DOI: 10.1145/1506216.1506224. [Online]. Available: https://doi.org/10.1145/1506216.1506224 (cit. on pp. 9, 21).

[9] N. Rahimi, F. Eassa, and L. Elrefaei, “An ensemble machine learning technique for functional requirement classification,” *Symmetry*, vol. 12, no. 10, 2020, ISSN: 2073-8994. DOI: 10.3390/sym12101601. [Online]. Available: https://www.mdpi.com/2073-8994/12/10/1601 (cit. on pp. 10, 25, 31–33, 36–38, 46, 56–58, 75).

[10] Z. Kurtanović and W. Maalej, “Automatically classifying functional and non-functional requirements using supervised machine learning,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 490–495. DOI: 10.1109/RE.2017.82 (cit. on pp. 10, 25, 27, 28, 32–34, 37, 38, 46, 69, 70, 75).

[11] M. A. Haque, M. Abdur Rahman, and M. S. 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)*, 2019, pp. 1–5. DOI: 10.1109/ICASERT.2019.8934499 (cit. on pp. 10, 25, 31, 33, 34, 36–39, 46, 58–68, 75).

[12] C. Baker, L. Deng, S. Chakraborty, and J. Dehlinger, “Automatic multi-class non-functional software requirements classification using neural networks,” in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2019, pp. 610–615. DOI: 10.1109/COMPSAC.2019.10275 (cit. on pp. 10, 25, 31, 34, 35, 37, 38, 46, 70, 71, 75).

[13] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, “Crisp-dm 1.0 step-by-step data mining guide,” The CRISP-DM consortium, Tech. Rep., Aug. 2000. [Online]. Available: https://maestria-datamining-2010.googlecode.com/svn-history/r282/trunk/dmct-teorica/tp1/CRISPWP-0800.pdf (cit. on p. 12).

[14] J. Cleland-Huang, M. Vierhauser, and S. Bayley, “Dronology: An incubator for cyber-physical systems research,” in *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*, 2018, pp. 109–112 (cit. on pp. 17, 20).

[15] FMT ISTI. (). "Natural Language Requirements Dataset," [Online]. Available: http://fmt.isti.cnr.it/nlreqdataset/ (visited on 05/2021) (cit. on pp. 17, 18, 21).

[16] Van Domburg. (2009). "Web Architectures for Services Platforms (WASP) Requirements," [Online]. Available: https://www.zenodo.org/record/581655 (visited on 05/2021) (cit. on pp. 17, 20).

[17] Leeds University Library Requirements. (2013). "Web Architectures for Services Platforms (WASP) Requirements," [Online]. Available: https://leedsunilibrary.wordpress.com/2013/06/05/research-data-repository-requirements/ (visited on 05/2021) (cit. on pp. 17, 20).

[18] INSPIRE. (). "INSPIRE Helpdesk "MIWP 2014–2016" MIWP-16: Monitoring," [Online]. Available: https://wayback.archive-it.org/12090/20201223150141/https://ies-svn.jrc.ec.europa.eu/documents/33 (visited on 05/2021) (cit. on pp. 17, 18, 21).

[19] S. H. Houmb, S. Islam, E. Knauss, J. Jürjens, and K. Schneider, "Eliciting security requirements and tracing them to design: An integration of common criteria, heuristics, and umlsec," *Requir. Eng.*, vol. 15, no. 1, pp. 63–93, Mar. 2010, ISSN: 0947-3602. DOI: 10.1007/s00766-009-0093-9. [Online]. Available: https://doi.org/10.1007/s00766-009-0093-9 (cit. on pp. 17, 19).

[20] F. Dalpiaz, D. Dell'Anna, F. B. Aydemir, and S. Çevikol, "Requirements classification with interpretable machine learning and dependency parsing," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 2019, pp. 142–152. DOI: 10.1109/RE.2019.00025 (cit. on pp. 18, 20, 24).

[21] ReqView. (). "Example Requirements Specification Documents," [Online]. Available: https://www.reqview.com/doc/example-requirements-documents (visited on 05/2021) (cit. on p. 20).

[22] F.-L. Li, J. Horkoff, J. Mylopoulos, R. S. S. Guizzardi, G. Guizzardi, A. Borgida, and L. Liu, "Non-functional requirements as qualities, with a spice of ontology," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, 2014, pp. 293–302. DOI: 10.1109/RE.2014.6912271 (cit. on p. 20).

[23] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009 (cit. on pp. 25, 27).

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011 (cit. on pp. 26, 32–34).

[25] G. Salton and M. J. McGill, "Introduction to modern information retrieval," 1986 (cit. on p. 26).

[26] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguist.*, vol. 19, no. 2, pp. 313–330, Jun. 1993, ISSN: 0891-2017 (cit. on p. 27).

[27] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: ACM, 2016, pp. 785–794, ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785 (cit. on p. 37).

[28] A. Liaw and M. Wiener, "Classification and regression by random forest," *R News*, vol. 2, no. 3, pp. 18–22, 2002. [Online]. Available: https://CRAN.R-project.org/doc/Rnews/ (cit. on p. 37).

[29] Walber. (Nov. 2014). "Precision and recall," [Online]. Available: https://commons.wikimedia.org/wiki/File:Precisionrecall.svg (visited on 05/2021) (cit. on p. 45).

[30] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? a study of classifying requirements," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 496–501. DOI: 10.1109/RE.2017.36.

[31] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, 2016, pp. 39–45. DOI: 10.1109/REW.2016.021.

[32] R. Yadav, "Light-weighted CNN for text classification," *CoRR*, vol. abs/2004.07922, 2020. arXiv: 2004.07922. [Online]. Available: https://arxiv.org/abs/2004.07922.

[33] I. Good, "Some terminology and notation in information theory," 1956.

[34] S. Ruder, “An overview of gradient descent optimization algorithms,” *CoRR*, vol. abs/1609.04747, 2016. arXiv: 1609.04747. [Online]. Available: http://arxiv.org/abs/1609.04747.

[35] C. Raffel, *Http://colinraffel.com/wiki*, Dec. 2015. [Online]. Available: http://colinraffel.com/wiki/neural_network_hyperparameters.

Index