

**SOFTWARE REQUIREMENTS CLASSIFICATION
USING DEEPLARNING: DEALING WITH DATA
SCARCITY**

BY

OSAMAH ALDHAFER

A Dissertation Presented to the
FACULTY OF THE COLLEGE OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

In

INFORMATION & COMPUTER SCIENCE

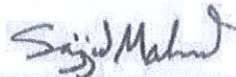
May 2022

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

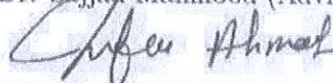
DEANSHIP OF GRADUATE STUDIES

This dissertation, written by **OSAMAH ALDHAFER** under the direction of his dissertation adviser and approved by his dissertation committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **DOCTOR OF PHILOSOPHY IN INFORMATION & COMPUTER SCIENCE**.

Dissertation Committee



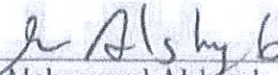
Dr. Sajjad Mahmood (Adviser)



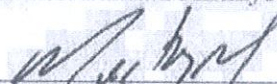
Dr. Irfan Ahmad (Co-Adviser)



Dr. Mahmood Niazi (Member)



Dr. Mohammad Alshayeb (Member)



Dr. Moataz Ahmed (Member)



Dr. Hamoud Aljamaan
Department Chairman



Dr. Suliman Al-Homidan
Dean of Graduate Studies



Date

©Osamah Aldhafer
2022

To my beloved father soul

ACKNOWLEDGMENTS

This thesis is the product of a lot of hard work and study, but it is also the result of guidance and support that I have kindly received from many people: my professors, colleagues, family, and friends to whom I am indebted.

I would like to thank my advisor Dr. Sajjad Mahmood and co-advisor Irfan Ahmad. They have always been there to listen to my questions and to provide me with constructive criticism on my research. I also thank my doctoral committee members: Dr. Mahmood Niazi, Dr. Mohammad Alshayeb, and Dr. Moataz Ahmed for their insightful comments and advice on my thesis, and final defense.

I am so grateful to all my family members for their continuous love and support. I can't possibly express my gratitude to them. Their care and love have always been the source of my ambitious aspirations. Also to my friends who have been a constant source of inspiration with their positivity and enthusiasm.

My sincere appreciation goes to the Higher Education Ministry of Yemen and to King Fahd University of Petroleum and Minerals (KFUPM) for providing me with the opportunity to continue my graduate studies. Special thanks to the ICS Department at KFUPM for their invaluable support.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	vi
LIST OF TABLES	xi
LIST OF FIGURES	xv
LIST OF ABBREVIATIONS	xviii
ABSTRACT (ENGLISH)	xix
ABSTRACT (ARABIC)	xxii
CHAPTER 1 INTRODUCTION	1
1.1 Thesis Motivations	7
1.2 Thesis Main Contributions	12
1.3 Thesis Organization	16
CHAPTER 2 BACKGROUND	17
2.1 Software Requirements	17
2.2 Machine Learning Techniques	19
2.2.1 Conventional Machine Learning Techniques	20
2.2.2 Deep Learning (DL) Techniques	21
2.3 Natural Language Processing	23
CHAPTER 3 LITERATURE REVIEW	25

3.1	Systematic Literature Review: Software Requirements Classification using Deep Learning	25
3.1.1	Existing SLR in Software Requirements Classification	25
3.1.2	Methodology	27
3.1.3	Results	31
3.1.4	Open Challenges for Future Research	49
3.1.5	Threats to Validity	55
3.1.6	Conclusions	56
3.1.7	Primary Studies List:	57
3.2	Arabic Software Requirements Classification	60
3.3	English Software Requirements Classification	65
3.3.1	Binary Classification	66
3.3.2	Multiclass and Multilabel Classification	68
3.4	Data Augmentation in Software Requirements	71

CHAPTER 4 AN END-TO-END DEEP LEARNING SYSTEM FOR REQUIREMENTS CLASSIFICATION USING RECURRENT NEURAL NETWORKS 72

4.1	Introduction	72
4.2	Research Design	73
4.2.1	Motivation	73
4.2.2	Research Methodology	75
4.2.3	Datasets	79
4.3	Experiments, Results and Discussions	84
4.3.1	Evaluation Measures	84
4.3.2	Experiment Setup	85
4.3.3	Results	88
4.3.4	Discussion	94
4.4	Comparison with Published Works	98
4.4.1	Binary Classification	99

4.4.2	Multiclass Classification	100
4.4.3	Multilabel Classification	101
4.5	Threats to Validity	103
4.6	Conclusions and Future Work	105

CHAPTER 5 AUGMENTATION-BASED TECHNIQUES FOR SOFTWARE REQUIREMENTS CLASSIFICATION 106

5.1	Introduction	106
5.2	Research Design	107
5.2.1	Research Methodology	107
5.2.2	Existing Datasets	108
5.2.3	Architecture of Augmentation-based Software Requirement Classifier	111
5.2.4	Architecture of deep learning Classifier	120
5.2.5	Evaluation Measures	121
5.3	Experiment Setup	122
5.4	Results and Discussion	122
5.4.1	Results of Common Data Augmentation Technique	122
5.4.2	Results of The Proposed Techniques	124
5.5	Conclusion	133

CHAPTER 6 SOFTWARE REQUIREMENT CLASSIFICATION IN A LOW RESOURCE LANGUAGE 135

6.1	Introduction	135
6.2	Research Design	136
6.2.1	Existing Datasets	136
6.2.2	Research Methodology	138
6.2.3	Evaluation Measures	142
6.3	Experiment Setup	144
6.4	Results and Discussion	145

6.4.1	Arabic Software Requirements Classification in En2ArPROMISE Dataset	145
6.4.2	Impact of Translation on Classifying Arabic Software Requirements	154
6.4.3	Summary and Recommendations	158
6.5	Threats to Validity	160
6.6	Conclusions and Future Works	161
CHAPTER 7 CONCLUSIONS AND FUTURE WORK		163
7.1	Conclusions	163
7.2	Limitations	165
7.3	Future Directions	166
7.4	Outcomes	166
REFERENCES		168
VITAE		186

LIST OF TABLES

3.1	Comparison between the different reviews	27
3.2	Keywords and their synonyms	28
3.3	Quality assessment template	30
3.4	Headings of extraction form	30
3.5	Papers distribution based on the publisher	31
3.6	the objectives of the primary studies	32
3.7	Available software requirement datasets	35
3.8	Versions and modalities of PROMISE dataset	36
3.9	Distribution of attributes in every dataset for functional and non- functional requirements	37
3.10	Distribution of classes in every dataset for security requirements	37
3.11	NLP techniques in the primary studies	39
3.12	Feature representation in the primary studies	40
3.13	DL algorithms in the primary studies	42
3.14	CNN hyperparameters	43
3.15	RNN hyperparameters	45
3.16	Transformers hyperparameters	46
3.17	performance results of DL models	48
4.1	Distribution of the classes in the PROMISE dataset along with the average number of words and characters counts per sample.	81
4.2	Descriptive statistics for the PROMISE, PROMISE ML and EHR datasets.	82

4.3	Distribution of the classes in the PROMISE ML dataset along with the average number of words and characters counts per sample.	82
4.4	PROMISE ML dataset examples	83
4.5	Distribution of the classes in the EHR dataset along with the average number of words and characters counts per sample.	84
4.6	Parameters of the embedding layer using different architectures.	86
4.7	Summary of important hyperparameters for binary classification using the word and character models on the PROMISE dataset	88
4.8	Binary classification results for the word and character models on the PROMISE dataset.	90
4.9	Summary of important hyperparameters for multiclass classification using word and character models on the PROMISE dataset.	90
4.10	Multiclass classification results for the word and character models on the PROMISE dataset.	91
4.11	Summary of the important hyperparameters for multiclass multilabel classification using the word and character models on the PROMISE ML dataset.	91
4.12	Multiclass multilabel classification results using the word and character models on the PROMISE ML dataset.	92
4.13	Summary of the important hyperparameters for binary multilabel classification using the word and character models on the EHR dataset.	93
4.14	Binary multilabel classification results using the word and character models on the EHR dataset.	93
4.15	Summary of the important hyperparameters for multiclass multilabel classification using the word and character models on the EHR dataset.	94
4.16	Multiclass multilabel classification results using the word and character models on the EHR dataset.	94
4.17	Comparison with published works on binary requirements classification tasks.	100

4.18	Comparison with published works on multiclass requirements classification tasks	102
4.19	Comparison with published works on multilabel requirements classification tasks.	103
5.1	Distribution of the classes in the PROMISE dataset along with the average number of words counts per sample.	110
5.2	Distribution of the classes in the EHR dataset along with the average number of words counts per sample.	110
5.3	Results of using common data augmentation technique on several portions of PROMISE dataset.	123
5.4	Comparison between the F-score of the augmented and non-augmented datasets in three different modalities.	123
5.5	Computing the number of augmentation that need to be generated based on the requirement length in one label software requirements augmentation technique.	125
5.6	Comparison between the results of common and weighted one label augmentation on the PROMISE dataset.	129
5.7	Computing the number of augmentation that need to be generated based on the class and requirement length in multilabel software requirements augmentation technique.	129
5.8	Comparison between the results of common and weighted multilabel augmentation on EHR dataset.	132
5.9	Comparison between the results of common and dictionary and transformer-based software requirements augmentation on four classes PROMISE dataset.	132
5.10	Comparison between the results of common and dictionary and transformer-based software requirements augmentation on ten classes PROMISE dataset.	133
6.1	Descriptive statistics of English and Arabic PROMISE datasets	137

6.2	Descriptive statistics of English and Arabic News dataset	137
6.3	Descriptive statistics of English and Arabic requirement dataset	138
6.4	Comparison between the experiments	139
6.5	Word and character binary model results in En2ArPROMISE dataset .	146
6.6	Comparison between PROMISE and En2ArPROMISE datasets	147
6.7	Comparison of PROMISE binary classification	150
6.8	Word and character four classes model results in En2ArPROMISE dataset	150
6.9	Word and character Ten classes model results in En2ArPROMISE dataset	151
6.10	Comparison of PROMISE multi-class classification	154
6.11	Sentence length and vocabulary size in ArNews and Ar2ENNews datasets	154
6.12	Five news classification results in ArNews and Ar2EnNews datasets . .	155
6.13	Word and character Ten classes model results	157

LIST OF FIGURES

1.1	Three interacting fields in software requirement classification	1
1.2	Data scarcity in software requirements classification and our three objectives	6
1.3	Scenarios of classifying Arabic software requirements	11
1.4	General proposed model for cross-lingual software requirements classification	12
2.1	Main components of SRS document	20
2.2	Conventional Neural Networks	22
2.3	Conventional Neural Networks [1]	22
2.4	Natural Language Processing Pipeline	24
2.5	Example of Word Embedding	24
3.1	General stages of SLR protocol	28
3.2	Summary of papers identification and selection	30
3.3	The pipeline for solving the problems of requirement classification using DL	31
3.4	Different granularity in classifying software requirements	33
3.5	Number of classes and used performance metrics in the primary studies	49
3.6	Used dataset and evaluation techniques in the primary studies	49
4.1	A sample requirement from the PROMISE dataset is separated into three sentences with multiple labels.	75
4.2	Steps of minimal pre-processing using word and character sequences. .	76
4.3	Binary word (<i>left</i>) and character (<i>right</i>) model summary	89

4.4	PROMISE NFR4 confusion matrices for word and character models . .	91
4.5	PROMISE NFR5 confusion matrices for word and character models . .	92
4.6	PROMISE NFR10 confusion matrices for word and character models .	92
4.7	Performance of the word and character models on the ten folds of the PROMISE dataset.	98
4.8	F1-score of the word and character models on single-label and multilabel classifications.	99
5.1	Overview of the research methodology in classifying software requirements using augmentation-based technique.	109
5.2	Example of WordNet semantic network of the word book.	112
5.3	Architecture of common data augmentation that have been proposed in [2].	114
5.4	Examples of three words with their synonyms that belong to three nonfunctional classes.	119
5.5	Architecture of the proposed dictionary and transformer-based data augmentation technique in software requirement classification.	121
5.6	The statistics of classes using the common one label augmentation on binary PROMISE dataset.	126
5.7	The statistics of classes using the weighted one label augmentation on binary PROMISE dataset.	126
5.8	The statistics of classes using the common one label augmentation on four PROMISE dataset.	127
5.9	The statistics of classes using the weighted one label augmentation on four PROMISE dataset.	127
5.10	The statistics of classes using the common one label augmentation on ten PROMISE dataset.	128
5.11	The statistics of classes using the weighted one label augmentation on ten PROMISE dataset.	128
5.12	The statistics of classes using common multilabel augmentation on EHR dataset.	130

5.13	The statistics of classes using the weighted multilabel augmentation on EHR dataset.	131
5.14	Steps of validation stage in dictionary and transformer-based software requirements augmentation.	132
6.1	The research framework of the research	139
6.2	The main components of research methodology	140
6.3	Confusion matrix	143
6.4	The four dimensions of conducting experiments on En2ArPROMISE . .	145
6.5	Word and character models in En2ArPROMISE classification	147
6.6	Word and character Four classes models in En2ArPROMISE classification	151
6.7	Word and character ten classes models in En2ArPROMISE classification	152
6.8	The four dimensions of conducting experiments on ArNews and Ar2EnNews	155
6.9	Arabic and English models in news classification	156
6.10	The three dimensions of conducting experiments on ArSRS and Ar2EnSRS	156
6.11	Confusion matrices of classifying ArSRS and Ar2EnSRS datasets	158
6.12	Semi-formal Arabic requirements style converter	160

LIST OF ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Networks
DL	Deep Learning
KFUPM	King Fahd University of Petroleum and Minerals
ML	Machine Learning
Ph.D.	Doctor of Philosophy
RNN	Recurrent Neural Networks

DISSERTATION ABSTRACT

NAME: Osamah AlDhafer

TITLE OF STUDY: Software Requirements Classification Using Deep Learning: Dealing with Data Scarcity

MAJOR FIELD: Information & Computer Science

DATE OF DEGREE: April 2022

Natural language documents are the primary source of requirements in a software project. Requirements classification is an essential step toward automatically analyzing natural language-based requirement artifacts. Requirements are typically classified into two main categories, namely, Functional Requirements (FRs) and Nonfunctional Requirements (NFRs). Nonfunctional requirements are further classified into a range of quality requirements, such as 'security', 'availability', and 'usability'. Software success depends on adherence to both functional and nonfunctional requirements. Hence, requirement engineers need to identify and categorize both functional and nonfunctional requirements from a range of natural language-based artifacts. Existing requirements classification approaches mainly use lexical and syntactical features to classify requirements using both traditional machine learning and deep learning approaches

with promising results. However, the existing techniques depend on word and sentence structures and employ preprocessing and feature engineering techniques to classify requirements from textual natural language documents. Moreover, existing studies deal with requirements classification as binary or multiclass classification problems and not as multilabel classification, although a given requirement can belong to multiple classes at the same time. Additionally, data scarcity and data imbalanced are two main challenges in software requirements classification using deep learning techniques. As a result, the classifier’s performance is affected and biased against classes with few samples. The issue of data scarcity is further exacerbated for low-resource languages such as Arabic. In previous studies, existing approaches investigated software requirement classification for requirements written mostly in English language and other languages using the Roman script. In this dissertation, we classify requirements into functional and nonfunctional and investigate the multilabel classification problem. An end-to-end deep learning pipeline is presented to perform requirements classification with minimal preprocessing and no feature engineering. In addition, we investigate data augmentation techniques for dealing with the issues of limited and imbalanced training data. We present Adaptive data augmentation techniques using class and sample length to dynamically assign percent of change and number of augmentation for one and multilabel classification. We present dictionary-based and transformer-based data augmentation techniques to address issues of textual data augmentation in the software requirements domain. Finally, the classification of the requirements written in Arabic as a low-resource language has been investigated. Several experiments have been conducted to highlight the effects of different language-specific NLP techniques

on the performance of Arabic requirements classification. Machine Translation (MT) impact has been examined on datasets written originally either in Arabic or English languages to provide an alternative approach for low-resource language like Arabic by translating the Arabic requirements into English and then using the system trained on English dataset to perform requirements classification. Experiments conducted on publicly available datasets show the effectiveness of the presented techniques. We achieve state-of-the-art results on most tasks using word sequences as tokens. Requirements can be effectively classified into functional and nonfunctional categories using the proposed recurrent neural network-based deep learning system, which involves minimal text preprocessing and no feature engineering. Further improvements in classification results are reported using our adaptive data augmentation techniques and dictionary-based transformer-based data augmentation technique. Several experiments show the possibility of using machine translation of Arabic requirements as a viable alternative to employ language-specific NLP processing and to deal with data scarcity issues.

ملخص الرسالة

الاسم: أسامة الظافر

عنوان الدراسة: تصنيف متطلبات البرامج باستخدام التعلم العميق: التعامل مع بيانات التدريب المحدودة

التخصص: علوم الحاسب الآلي

تاريخ الدرجة العلمية: إبريل 2022

يعد تصنيف المتطلبات تصنيف المتطلبات المستندة إلى اللغة الطبيعية يدويًا إلى فئات أمرًا معقدًا وعرضة للخطأ ويستغرق وقتًا طويلاً. عملية تصنيف المتطلبات مؤتمتة باستخدام خوارزميات التعلم الآلي (ML) والتعلم العميق (DL) التقليدية. حققت الخوارزميات التقليدية القائمة على ML نتائج واعدة مقارنة بالطرق القائمة على القواعد. ومع ذلك ، فإن الخوارزميات التقليدية القائمة على ML لها العديد من أوجه القصور ، مثل ضرورة تحفيز هندسة الميزات الشاملة الباحثين على إيجاد حلول بديلة. لقد تجاوزت النماذج المستندة إلى DL أحدث ما توصلت إليه التكنولوجيا في جميع مهام البرمجة اللغوية العصبية الهامة تقريبًا. تستخدم مناهج تصنيف المتطلبات الحالية بشكل أساسي الميزات المعجمية والنحوية لتصنيف المتطلبات باستخدام كل من التعلم الآلي التقليدي وأساليب التعلم العميق ذات النتائج الواعدة. ومع ذلك ، تعتمد التقنيات الحالية على تراكيب الكلمات والجمل وتستخدم تقنيات المعالجة المسبقة وهندسة الميزات لتصنيف المتطلبات من مستندات اللغة الطبيعية النصية. علاوة على ذلك ، تتعامل الدراسات الحالية مع تصنيف المتطلبات كمشكلات تصنيف ثنائية أو متعددة الطبقات وليس كتصنيف متعدد العلامات ، على الرغم من أن أحد المتطلبات يمكن أن ينتمي إلى فئات متعددة في نفس الوقت. بسبب مشكلة وضع العلامات ، هناك فجوة كبيرة في دقة النماذج بين لغات الموارد المحدودة مثل العربية مقارنة باللغات الغنية بالموارد. على حد علمنا ، نظرت الأساليب الحالية في متطلبات البرامج المكتوبة في الغالب باللغة الإنجليزية ولغات أخرى باستخدام النص الروماني. جميع مجموعات البيانات المتاحة غير متوازنة حيث يكون الاختلاف بين عدد عينات الفئات مرتفعًا. هناك نقص في تقنية زيادة البيانات التي يمكنها تحسين النتائج وتقليل تحيز مجموعة البيانات غير المتوازنة في مجال متطلبات البرامج. في

هذه الرسالة ، نضيف المتطلبات إلى أنواع وظيفية وغير وظيفية مختلفة مع الحد الأدنى من المعالجة المسبقة ونمذجة المهمة كمسألة تصنيف متعددة التسميات. بالإضافة إلى ذلك ، تم التحقق من تصنيف المتطلبات من اللغة الأم أو مستند البرنامج المترجم باستخدام الترجمة الآلية (AMT) مع فحص تأثيرها على ثلاث مجموعات بيانات مختلفة مكتوبة في الأصل إما باللغة العربية أو الإنجليزية. أخيرًا ، نقترح تقنية زيادة البيانات المستندة إلى القاموس والمعتمدة على المحولات لمعالجة مشكلات الحجم غير المتوازن أو المحدود لمجموعة البيانات في مجال متطلبات البرامج. التجارب التي أجريت على مجموعات البيانات PROMISE و EHR المتاحة للجمهور تظهر فعالية التقنيات المقدمة. نحقق أحدث النتائج في معظم المهام باستخدام تسلسل الكلمات كرموز. يمكن تصنيف المتطلبات بشكل فعال إلى فئات وظيفية ومختلفة غير وظيفية باستخدام نظام التعلم العميق القائم على الشبكات العصبية المتكرر المقدم ، والذي يتضمن الحد الأدنى من استيعاب النص وعدم وجود هندسة للميزات..

CHAPTER 1

INTRODUCTION

Natural language documents are the primary source of requirements in a software project. The mistakes in the requirement stage will raise the likelihood of costly problems in subsequent phases [3]. Classifying requirements automatically into categories assists requirement engineers in reducing the time and risk of failure [4], [5]. Three main domains, Requirement Engineering (RE), onventional Machine Learning (ML) or Deep Learning (DL), and Natural Language Processing (NLP), interact to achieve the objective of automating requirement classification, as shown in Figure 1.1.

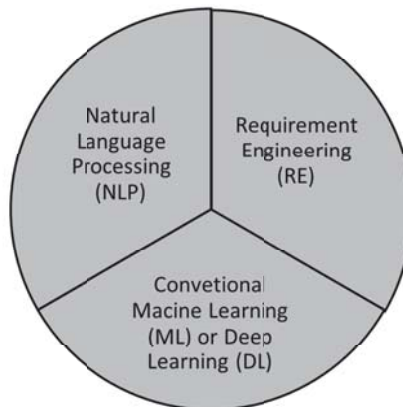


Figure 1.1: Three interacting fields in software requirement classification

Requirements are typically classified into two main categories, namely, Functional Requirements (FRs) and Nonfunctional Requirements (NFRs). Nonfunctional requirements are further classified into a range of quality requirements, such as 'security', 'availability', and 'usability'. Software success depends on adherence to both functional and nonfunctional requirements. Although the controversial distinction between functional and nonfunctional requirements [6], [7], automatically categorizing software requirements can play an essential role in addressing the issues of requirement misinterpretation or misunderstanding. Hence, requirement engineers need to identify and categorize both functional and nonfunctional requirements from a range of natural language-based artifacts.

Over the years, several automated requirements classification approaches have been developed based on lexical and syntactical features using both traditional and deep learning techniques [8]. For example, Cleland-Huang et al. [5] proposed a technique for classifying nonfunctional requirements using information retrieval to identify indicator terms that guide the requirements classification process. Hussain et al. [9] developed an approach that uses linguistic features such as Part Of Speech (POS) tagging to classify functional and nonfunctional requirements with higher accuracy. Similarly, Kurtanovic and Maalej [10] applied Support Vector Machines (SVMs) which achieved precision and recall of up to 92%, employing a large number of features at the word level using metadata, lexical, and syntactical features to classify requirements.

Using conventional ML techniques, several requirements classification approaches based on lexical and syntactical features have been developed [8]. However, conventional ML algorithms have several shortcomings that motivate researchers to find

alternative solutions. For example, conventional ML techniques require extensive feature engineering and substantial domain knowledge to convert raw data into suitable internal representations [11]. With the massiveness of data, DL techniques outperform conventional ML techniques in different domains and fully utilize affordable hardware such as Graphical Processing Units (GPUs) [12]. Other limitations of using conventional ML techniques such as sparse representation, context loss, and arbitrary encoding [13].

Recently, deep learning approaches for classifying software requirements have gained increasing attention. Classifying natural language sentences has been studied by many researchers using DL approaches [14]. For example, Navarro-Almanza [15] used the Convolutional Neural Network (CNN) and word2vec, a word representation technique to classify 12 categories of functional and nonfunctional requirements. A comparison between the performance of the convolutional neural network and two Recurrent Neural Network (RNN) variants, the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM), in NFR classification was reported in [13]. The study used preprocessing steps such as lemmatization and stopword removal to improve the classifier’s performance. The transformer models based on an attention mechanism achieved state-of-the-art results on different problems and improved the model’s generalization to diverse domains [16].

In addition to using deep learning models, state-of-the-art performance is achieved by using a high-quality word representation that captures the syntactic and semantic similarities of words such as word2vec [17]. Each word in word2vec is represented in a fixed form that does not alter with context. BERT is a neural network-based

model that can be used to dynamically represent words based on their context [18]. Superior results in several domains are gained using deep learning, word embeddings, and different NLP preprocessing steps, such as lemmatization and stopword removal, to improve the classifier's performance.

When there are limited or no labeled training data available, or when there are not enough data for a particular label compared to other labels, this is known as data scarcity. Clearly, there are several main challenges of data scarcity in the domain of software requirements classification. Labeling the data to train supervised machine learning techniques is one of the main challenging issues because of the required time and domain knowledge. Generally, DL classifiers are considered data-hungry models that require a large quantity of data to show robust performance [19]. An imbalanced dataset collected from SRS is another challenge in software requirements classification. Based on the literature, no previous study has surveyed applying DL models in classifying software requirements. The published reviews considered only the identification and classification of non-functional software requirements using conventional ML in [8].

Limited labeled datasets and a limited number of samples in some classes might cause some difficulty in training the DL classifier. The ideal way to add new instances is to have humans rephrase the statements in the texts. However, paraphrasing the existing samples takes a long time to generate new samples by humans, especially when we have a large number of samples. Applying image augmentation in the text is not an option because of the semantics of the features (words). Data Augmentation (DA) is one way of dealing with the imbalanced or limited size of the dataset by automating the

creation of novel and realistic-looking training data after applying a transformation to an example without changing its label. Several automated augmentation techniques have been proposed in the literature using back-translation [20], data nosing [21], and synonym replacement using predictive language models [22]. However, the suitability of these techniques in classifying domain-specific tasks, such as software requirement classification, needs to be investigated.

In addition, no research has been found that investigates the analysis of software requirements documents written in other languages such as Arabic. Based on the literature, the analysis of software requirements written in the English language and other languages using the Roman script has been well-researched. However, Arabic is a challenging language due to its complex linguistic structure [23]. There are several variations between Arabic and English languages, as in [24], such as the order of the words and many personal names, morphological markers, particles, and other pronouns. The Arabic SRS documents are bilingual and contain Arabic sentences with technical terms such as the names of technology such as Apache Tomcat or Oracle DB2 MySql and file extensions such as HTML or XML.

In summary, it has been shown from the literature that data scarcity has affected the requirement classification task by either limited imbalanced labeled requirements or lack of labeled requirements. The first objective of this dissertation is to develop deep learning-based models that use the raw word or character sequences as input to classify one and multilabel requirements with lightweight preprocessing of the requirements texts and no feature engineering. The second objective is proposing different augmentation techniques adaptive data augmentation techniques using class and sam-

ple length to dynamically assign percent of change to tackle the issues of imbalanced problems in the domain of software requirements classifications. For data-hungry issue, dictionary-based and transformer-based data augmentation techniques have been proposed in the software requirements domain. In the third objective, we improve the performance of requirements categorization in Arabic by utilizing resources accessible in rich-resource languages. The three objectives address different aspects of data scarcity where the first two objectives have been implemented if we have a limited or imbalanced dataset. Data scarcity in software requirements classification and our three objectives are shown in Figure 1.2.

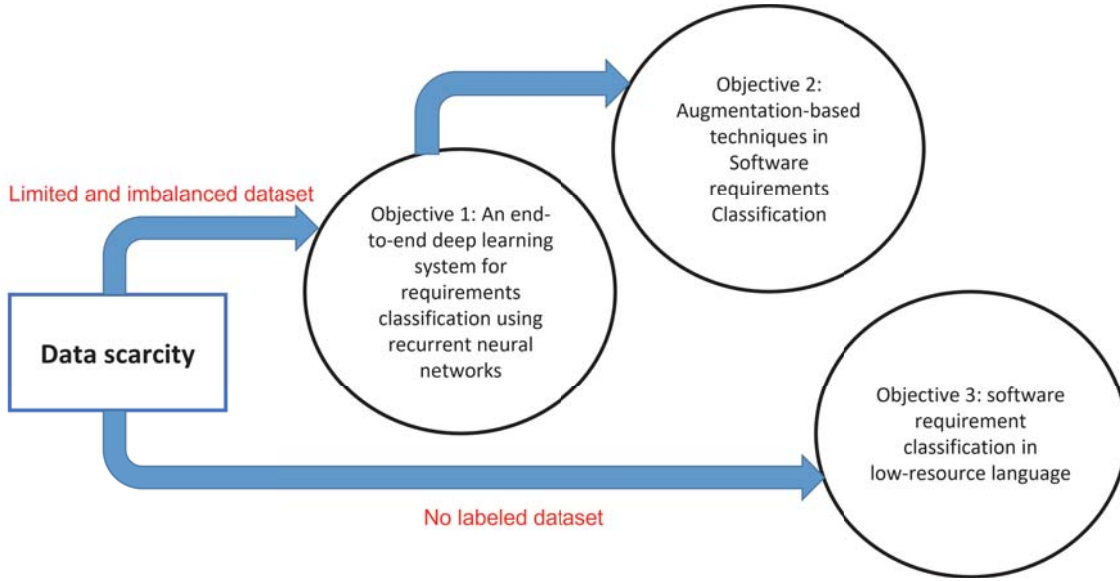


Figure 1.2: Data scarcity in software requirements classification and our three objectives

We achieve state-of-the-art results on most of the tasks using word sequences as tokens. Requirements can be effectively classified into functional and nonfunctional categories using the presented recurrent neural network-based deep learning system, which involves minimal text preprocessing and no feature engineering. Further im-

improvements in classification results are reported using our adaptive data augmentation techniques and dictionary-based transformer-based data augmentation technique. Several experiments show the possibility of using machine translation of Arabic requirements as a viable alternative to employ language-specific NLP processing and to deal with data scarcity issues.

1.1 Thesis Motivations

Based on the literature, the existing techniques mainly depend on word and sentence structures and require preprocessing and feature engineering to classify requirements from natural language documents. Accordingly, requirements engineers need a background in both Natural Language Processing (NLP) and machine learning. Techniques that can work with little or no preprocessing and feature engineering can help alleviate the need for in-depth knowledge in NLP and enable classification techniques to be readily accessible to researchers and practitioners alike.

Existing studies use different feature extraction and feature selection techniques to classify natural language requirements into functional and nonfunctional requirements. In this study, minimizing the usage of NLP processing is one of our goals in classifying the requirements. As a result, we investigate using raw texts as features for requirements classification. Raw text can be fed to the model in many ways. In the simplest form, raw text can be seen as a sequence of characters or words.

Existing studies use different feature extraction and feature selection techniques to classify natural language requirements into functional and nonfunctional requirements.

The present work is an attempt to narrow the research gap by proposing an end-to-end system for software requirements classification using recurrent neural networks. The approach works with minimal text preprocessing and no explicit feature engineering. The NLP preprocessing steps employed in this study are 'minimal' in the sense that the steps are trivial and related mainly to data cleaning and normalization. We are not employing sophisticated NLP processing either as preprocessing or feature engineering. Even steps such as stopword removal, which is considered 'basic', are not employed.

The study uses raw word or character sequences as input and does not employ modern NLP tokenization techniques such as SentencePiece [25] or stochastic tokenizers [26] or steps such as stemming and lemmatization. Generally, any processing step that requires a study of the trade-off between different choices is avoided because it entails performing experiments to decide the best settings that will work for a dataset. This additional work requires knowledge of how a particular choice can affect the overall performance. Thus, we believe that a process that can eliminate some or many of the calibration steps is easier to use in practical scenarios and may also generalize well to new environments and datasets.

Using words as tokens has a benefit over character tokens from the point of view that words have inherent meaning, whereas characters do not have such meaning. However, using the word sequence as an input requires building the input vocabulary which can lead to the Out-of-Vocabulary (OOV) problem, i.e., words in the test set that were not seen during the training. This leads to a large vocabulary size with thousands of unique words. Additionally, misspelled or unlemmatized words will not be handled appropriately.

Moreover, existing studies address the automatic processing of the requirements problem using binary and multiclass classification. Annotating the requirement with one label might raise some threats to validity due to the overlap between functional and quality aspects [27]. One requirement may be labeled by more than one type of nonfunctional or functional requirement. For example, text such as "The product shall provide asynchronous messaging to enable parallel processing and reduce the overhead in communications" belongs to functional class and nonfunctional classes, 'capacity' and 'performance', at the same time. Annotating sentences with multilabels is considered one of the ways to partially mitigate such issues associated with single labeling in requirements classification problems.

The current work applies a BiGRU-based deep learning system that uses raw word or character sequences as input to the model with minimal preprocessing of the requirements texts. This also alleviates the need for complex NLP-based feature engineering. BiGRU-based deep learning systems have shown promising results in other NLP and machine learning tasks but have not yet been adequately researched in relation to requirements classification. Moreover, we also present multilabel classification, in addition to both binary and multiclass classification, which has also been inadequately researched thus far.

DL techniques can process large datasets effectively; however, all publicly available datasets in the software requirements domain are considered small. Annotating different SRSs from different domains by domain experts will advance the research in the requirement classification field. The main challenge is that labeling the dataset manually is time and effort-consuming. Usually, collecting requirements from real

SRS documents eventually results in imbalanced datasets that contain different class sizes. Limited dataset sizes or imbalanced datasets can affect the performance of the classifier.

Some techniques can improve the results and minimize the bias of the imbalanced dataset such as class weight as in [13], [28], [29], oversampling [30], and random undersampling [30], [31] or undersampling using a clustering algorithm as in [32]. The aforementioned strategies are simple and use the same samples in the dataset. Another way of dealing with the imbalanced or limited size of the dataset is by paraphrasing the labeled requirements or using data augmentation techniques. Based on the literature, there is a lack of data augmentation techniques suitable in the software requirements domain.

Software Requirement Specification (SRS) documents contain the capabilities of software that have been agreed upon by stakeholders [33]. Writing the requirements in natural language may lead to inheriting challenges from the nature of language. English and numerous other languages have abundant resources, such as annotated data for requirement classification; however, similar resources are unavailable in constrained-resource languages such as Arabic. Therefore, the classification measures of supervised machine learning models in constrained-resource languages differ significantly from those of rich-resource languages.

Two scenarios can be followed by the researchers or practitioners to classify Arabic software requirements as illustrated in Figure 1.3. The first scenario depends on annotating and preprocessing Arabic texts. The challenge of collecting an annotated dataset is the main reason for limited research in the domain. Another challenge in

the first scenario is the difficulties in preprocessing Arabic texts, and scholars need to cope with a number of issues, such as the complex and rich structure of the word and morphology [34].

The normalizing of inconsistencies in the usage of specific letters, dialect terms, or diacritical signs is another problem. Another problem for academics and developers is overcoming challenges with NLP techniques such as tokenization, stemming, and morphological analysis. The second scenario overcomes the challenge of labeling the datasets by translating the Arabic requirements into English texts to utilize the availability of well-trained models on large datasets. The second advantage is the limited vocabulary and simplicity of preprocessing English texts by a variety of NLP tools.

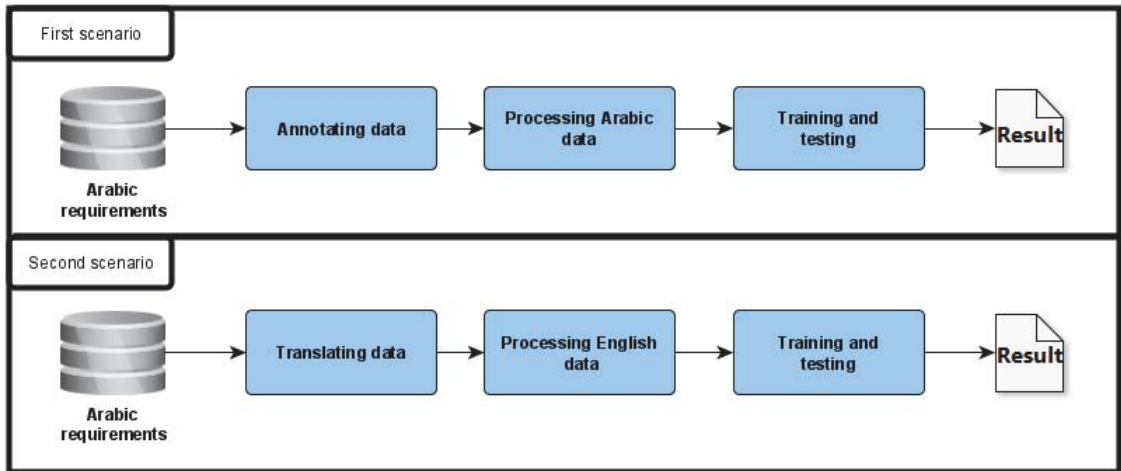


Figure 1.3: Scenarios of classifying Arabic software requirements

Obtaining state-of-the-art results in languages with limited resources is a difficult task. Although deep learning models outperform older methods, they require a huge amount of labeled data to be adequately trained. Labeling raw data, on the other hand, might be costly and time-consuming. Approaches such as transfer learning and active learning have emerged to address these difficulties in languages with limited

resources. This study aims to enhance the performance of software requirements categorization in Arabic by utilizing resources accessible in rich-resource languages.

Since it takes a long time to create corpora and lexical resources for a new language, one way to use previously built resources for one language to generate analysis tools for another language more quickly. This drives our investigation and application of cross-lingual lexicon translations and annotation as shown in Figure 4.1. Since Languages can differ significantly in terms of characteristics that impact software requirements classification, several experiments have been conducted to highlight the challenges of Arabic requirements classification.

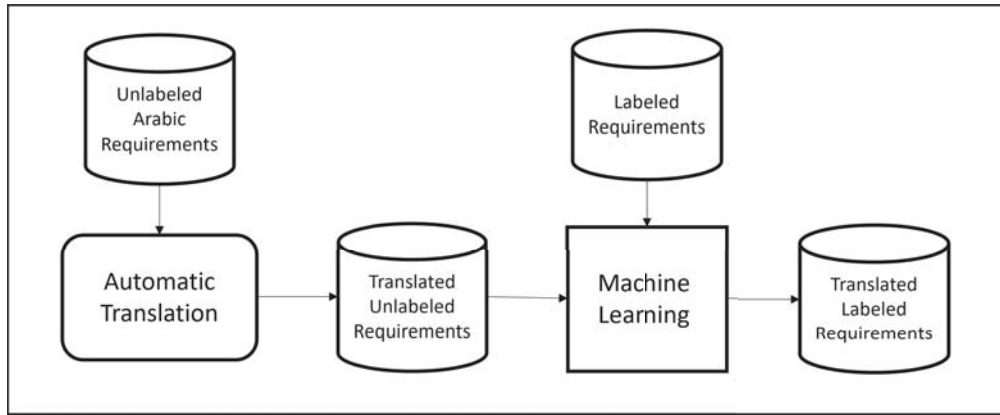


Figure 1.4: General proposed model for cross-lingual software requirements classification

1.2 Thesis Main Contributions

In this dissertation, we intend to achieve the following contributions:

1. Systematically review studies that have used deep learning-based techniques to classify requirements.

2. Propose an end-to-end deep learning system to classify one label and multilabel requirements with minimal processing.
3. Propose different augmentation techniques to address the limited and imbalanced issues in the software requirements dataset.
4. Propose a machine translation-based approach to tackle the lack of labeled datasets in the Arabic requirements domain.

Details of the thesis contributions are discussed in detail as follows:

- The systematic literature review aims to provide a body of knowledge for researchers about existing DL-based software requirements classification techniques and highlight research gaps. To achieve this objective, the systematic literature review will address four research questions as follows:
 - *RQ1: What are the available datasets used in the requirements classification task?*
 - *RQ2: What are the main text preprocessing and feature representation steps required to classify requirements using DL models?*
 - *RQ3: What are the most popular DL models with their hyperparameters implemented in the literature?*
 - *RQ4: What are the opportunities for future research in software requirement classification using DL techniques?*
- We develop an end-to-end deep learning system to classify requirements using two approaches: (i) word sequences and (ii) character sequences as input.

Next, we classify the NFRs into four, five, and ten subcategories. The developed system is evaluated on the publicly available PROMISE dataset for binary and multiclass classification tasks. Finally, we develop a deep learning-based model to perform multilabel classification on the modified PROMISE and EHR datasets. The main contributions of our work are as follows:

1. An end-to-end system to classify functional and nonfunctional requirements with minimal NLP-based preprocessing and no feature engineering.
 2. We present BiGRU models that receive a raw word or character sequences as inputs to classify the requirements.
 3. We evaluate the proposed models on three public datasets containing single-label and multilabel annotations and compare the results with the state-of-the-art approaches published in the literature.
- Software requirements augmentation technique that can tackle the issues of data-hungry and imbalanced software requirements classifier. Dynamic weights are considered based on the class shortage and the samples length. Using the general augmentation techniques without considering the domain characteristics might not achieve excellent results. Proposing domain-vocabulary and transformer-based augmentation techniques to address the domain-specific nature of the problem and prevent the distortion of the main features in the augmented sample. The following is a summary of our contributions:
 1. We apply data augmentation techniques to improve the software requirement classification.

2. We examine the data augmentation technique to improve the software requirement classification with a small dataset.
 3. We propose two versions of class-aware and length-aware data augmentation techniques to improve the model classification.
 4. We propose a domain vocabulary to control the augmentation operations to improve the software requirement classification
- Improving the performance of requirements categorization in Arabic requirements by utilizing resources accessible in rich-resource languages. In this part, the translation from low-resource to rich-resource language and vice versa have been experimented with to check whether classifying the native directly is better than translated SRS documents. In addition, two different tokenization approaches have been used to find the best tokenization suitable for Arabic requirements classification. The tokenized word and character sequences have been used to classify binary and multiclass Arabic requirements. The impact of preprocessing on the performance of classifiers has been investigated. The following is a summary of our contributions:
 1. We highlight several challenges of Arabic software requirements classification.
 2. We investigate the impact of preprocessing steps on the software requirements classification measurements.
 3. We examine the impact of automatic translation on the software requirements classification measurements.

4. We annotate a new Arabic software requirements dataset and evaluate the trained models on PROMISE and translated PROMISE to Arabic with different evaluation splits.

1.3 Thesis Organization

The following is a breakdown of the dissertation’s structure:

Chapter 2 highlights some important software requirements, machine learning, and natural language processing concepts. Chapter 3 details the related work about requirement classification and data augmentation techniques. Chapter 4 details the research methodology of classifying the requirements with minimal processing and existing datasets. The results of the classifiers and the discussions are presented. Then, we compare the reported results with state of the art in literature. Finally, we highlight the threats to validity. Chapter 5 details the research methodology of classifying requirements using augmentation techniques with existing datasets. The results of the classifiers and the discussions are presented. Then, we compare the reported results with state of the art in literature. Finally, we highlight the threats to validity. Chapter 6 details the research methodology of classifying Arabic requirements with existing datasets. The results of the classifiers and the discussions are presented. Then, we compare the reported results with state of the art in literature. Finally, we highlight the threats to validity. Chapter 7 presents the overall conclusion, limitations, and future directions.

CHAPTER 2

BACKGROUND

In this chapter, we are going to elaborate on different parts of this research to improve software quality.

2.1 Software Requirements

In this section, the definition of every software requirement phase is explained. The IEEE standard defines software requirement as:

- "An ability a user needs to solve a problem or achieve a goal".
- "A capability that must be met or possessed by a system or component of a system to satisfy a contract, standard, specification, or formally imposed document".
- "The set of all requirements that form the basis for the subsequent development of software or components of software".

The process of collecting the user requirements begins with the elicitation of the

data, which are collected through interviews, documents, meetings, workshops, prototyping and etc until writing formally the software requirements specification documents.. the four main processes are:-

- Requirements Elicitation and Analysis: Initially we identify the problem (a brief description) to solve and those involved in the system known as stakeholders. Stakeholders are the owners of the business and often with whom the precious information is, also interview people who directly or indirectly influence the product (system/app), who will use it and from there collect the preliminary information for the requirements. It is noteworthy that the requirements are fundamental for both price estimation and modeling, design, implementation, testing and also maintainability.
- Requirements Specification: In the specification, it may contain functional and non-functional requirements and even a use case diagram or prototyping of part of the product.
- Requirements Validation: After writing customer/user needs, it is important that you validate this data by being aware of the responsibility of the requirements raised. This step is also for corrections and other functionality can be discovered/included.
- Requirements Document: The document is intended to give a functional and overview of the product.

Software requirements specification (SRS) is a detailed description of the intended

system and environment for software that is being developed. The SRS outlines what the system will perform and how it is expected to function. There are two types of a requirement either functional or non-functional. Functional requirement as its name implies refers to system functionalities are the functions that the system/application must have to meet the business and its rules. Non-functional (Quality) requirements tangent the technical requirement of an environment, as aspects of system security, performance, and failure prevention.

An SRS reduces the time and effort necessary for developers to reach their objectives, as well as the cost of development. A good SRS specifies how an application will interact with system hardware, other programs, and human users. Usability, performance, operational, availability, portability, maintainability, security, recoverability from unexpected events are types of non-functional requirements that need to be addressed. As stated in the IEEE standard [35], the SRS should include three primary parts: an overview, requirements, and an appendix as in Figure 2.1. The requirements part is the technical part of the document that provides developers, designers, and testers with the information they need to create a product that meets business and user needs.

2.2 Machine Learning Techniques

ML algorithms are basically seek for patterns or discrepancies in data and are capable of detecting and incorporating such patterns into their data. there are three types or categories of machine learning: supervised, semi-supervised and unsupervised learn-



Figure 2.1: Main components of SRS document

ing. Supervised learning consists of indicating to the program a series of valid patterns so that it considers them as a reference. Unsupervised learning allows a machine to explore a data set and identify hidden patterns that link different variables. Unsupervised learning allows a machine to explore a data set and identify hidden patterns that link different variables. The machine learning techniques can be grouped into two categories conventional Machine Learning and deep learning techniques.

2.2.1 Conventional Machine Learning Techniques

conventional machine learning refers to a set of approaches that enable computer systems to predict or categorize without explicitly defining the rules. It is a branch of artificial intelligence that tries to educate computers to do example-based tasks without the need of explicit programming. There are several conventional ML techniques that have been used in the literature such as support vector machine, naive bayes and logistic regression.

2.2.2 Deep Learning (DL) Techniques

DL differs from typical ML approaches in that it automates the learning of data representation via many abstract layers. It technically works in the same way as machine learning, but with different capabilities and approaches. In deep learning, models use different layers to learn and discover insights from data. Deep learning is inspired by how human brain cells known as neurons work, which leads us to the concept of artificial neural networks . Also called deep neural network or deep neural learning. There are two main examples of DL models: convolutional neural networks (CNN) and recurrent neural network (RNN)

2.2.2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a type of feedforward ANN that has lately gained prominence for processing pictures as matrices. They are distinguished by layers for highlighting representations and are made up of convolutional, pooling, and fully connected layers [36]. The architecture of CNN has been illustrated in Figure 2.2

2.2.2.2 Recurrent Neural Network (RNN)

Natural Language Processing (NLP) and speech recognition employ commonly Recurrent Neural Networks (RNNs) because of the capability of utilizing sequential data. It has a directed circulation topology and employs internal memory to accommodate longer history data. It is referred to as recurrent because it uses the output of earlier calculations for the present computation of each element in a sequence [37]. There

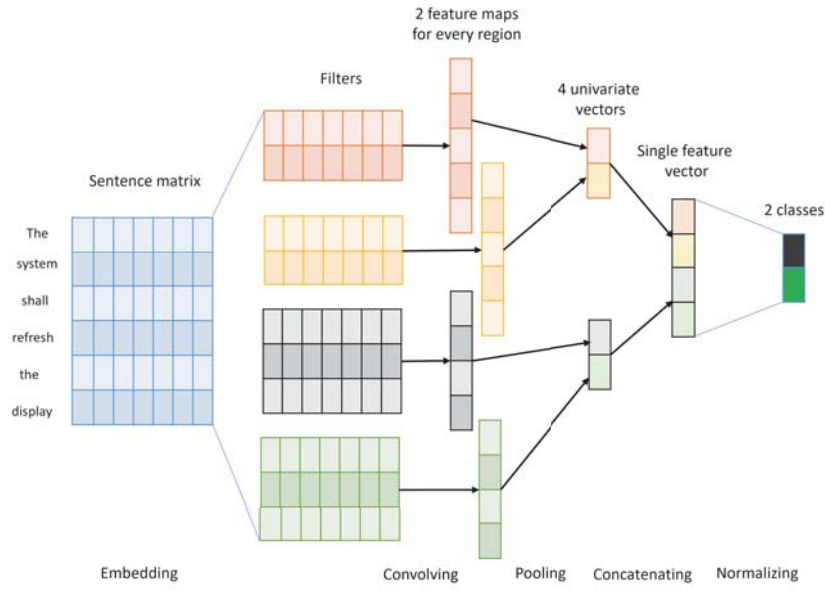


Figure 2.2: Conventional Neural Networks

are two recurrent neural networks (RNN) variants Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). The GRU architecture introduces two gates: a reset gate r that regulates the integration of new input with the prior memory and an update gate z that controls the preservation of prior memory.

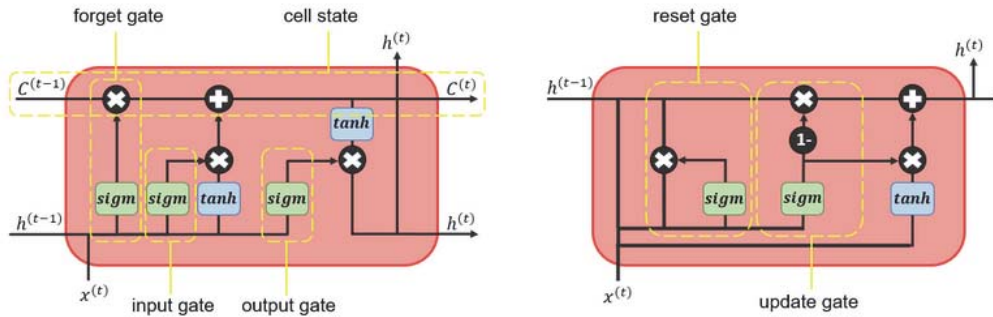


Figure 2.3: Conventional Neural Networks [1]

2.3 Natural Language Processing

The natural language processing is a field of computer science, artificial intelligence, and linguistics that studies the interactions between computers and human language. It is the processing of a text, dividing them into parts, adding grammatical elements and identifying interesting elements. The following are some of the common techniques of NLP:

- Sentence Segmentation: The first step in the pipeline is to break the text apart into separate sentences.
- Word Tokenization: This divides a sentence into words.
- POS Tagging: It has also been called labeling of parts of speech and assigns labels for nouns, pronouns, verbs, adjectives, adverbs, etc.
- Stemming and lemmatization: similar to the previous one, but you can get the basic form of a word based on its root, for example; better will be good.
- Stop word deletion: Word removal often occurs with articles such as the, and, a, etc. which do not contribute greatly to understanding the text and, therefore, can be deleted.
- Dependency Parsing: The next step is to figure out how all the words in our sentence relate to each other.
- Finding Noun Phrases: weve treated every word in our sentence as a separate entity. But sometimes it makes more sense to group together the words that

represent a single idea or thing.

- Named Entity Recognition: Identify and label words that represent real-world entities, such as people, organizations, places, dates, etc.

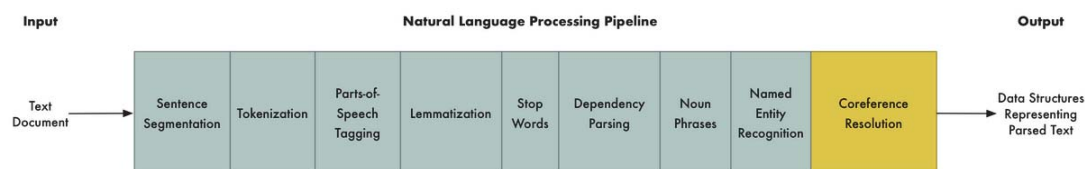


Figure 2.4: Natural Language Processing Pipeline

Distributional vectors or word embeddings essentially follow the distributional hypothesis, according to which words with similar meanings tend to occur in similar context. The example is shown in Figure 2.5.



Figure 2.5: Example of Word Embedding

CHAPTER 3

LITERATURE REVIEW

In this chapter, the systematic literature review about software requirements classification using deep learning techniques is illustrated in section 1. In section 2, the related work about the English requirements classification is described. In section 3, the related work about the Arabic requirements classification is described. Finally, augmentation-based technique

3.1 Systematic Literature Review: Software Requirements Classification using Deep Learning

3.1.1 Existing SLR in Software Requirements Classification

In this section, the contributions of several published reviews related to software requirements classification are summarized. The most relevant review reports 16 different conventional ML approaches for identification and classification of non-functional requirements only [8]. The primary studies of this review are 24 papers published

between 2007 to 2017.

Another systematic review that considered the identification and classification of software requirements in stack overflow using ML techniques [38]. Two types of requirements that are identified in stack overflow are functional and non-functional requirements. The 12 primary studies used the unsupervised learning technique Latent Dirichlet Allocation (LDA) and supervised learning technique Support Vector Machines (SVM).

The previous two reviews focused on dealing with only non-functional requirements classification using conventional ML techniques. A systematic mapping study with 404 primary studies that covered broadly the NLP technologies in the requirements engineering area [39]. The comprehensive mapping review covers the area of requirement engineering to provide in-width insight or map about the state of the art tools, technologies, and approaches in the requirement engineering field.

A systematic literature review conducted on using deep learning in different software processes or activities such as software requirement, software design, software maintenance, and testing [40]. Only two papers related to requirement classification have been part of the software requirement analysis process. Based on their 38 selected primary studies, the researchers conclude that software testing and maintenance processes are extensively investigated in the literature.

Based on the literature, there is a lack of recent in-depth research that reviews classifying software requirements from Software Requirement Specification (SRS) documents using DL techniques. First, commonly used datasets for software requirement classification are presented. Then, text preprocessing steps and feature representa-

tions that are required before the classification are investigated. The frequency of using DL models is reported with their optimal hyperparameter values and performance measures. The comparison between the is illustrated in Table 3.1.

Reference	Scope	Type	Studies	Time	Technique
[8]	Identification and classification of NFRs	Review	24	2007-2017	Conventional ML
[38]	Identification and classification of FR and NFRs in stack overflow	SLR	12	2013-2019	Conventional ML
[39]	NLP technologies in the requirements engineering	SMR	404	Until 2019	All
[40]	Deep learning in different software processes	SLR	38	2009-2019	DL
Our work	Deep learning in requirements classification	SLR	21	Until 2021	DL

Table 3.1: Comparison between the different reviews

3.1.2 Methodology

In the literature, a Systematic Literature Review (SLR) is considered one of the most well-known methods that has been applied to summarize the evidence and answer the formulated research questions. The procedures and guidelines in [41], [42] for conducting the SLR have been followed to identify and evaluate all the studies that address implementing DL-classifier of software requirements from SRS documents. We conduct SLR to answer our research questions because of the comprehensive and methodical steps of SLR to mitigate the biased conclusion and provide evidence for researchers about current studies and identify gaps in literature.

The SLR starts with identifying the aim of the systematic review and the search strategy. Then, the selection process needs to be specified by presenting the inclusion and exclusion criteria. Assessing the quality of the papers should be conducted to mitigate any bias. The next stage is extracting the required data from the selected

primary studies. Synthesizing and summarizing the findings by qualitative or quantitative methods is the last stage in the SLR protocol. The main stages of the SLR protocol are shown in Figure 3.1.

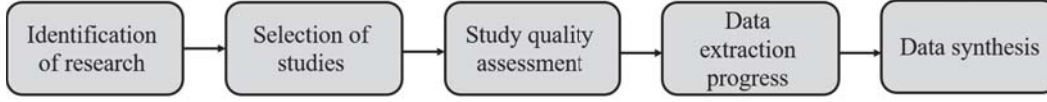


Figure 3.1: General stages of SLR protocol

To answer the formulated research questions, electronic search indexing (Scopus and Web of Science) and different online libraries (IEEE, ACM, Springer, Science Direct, and John Wiley) have been used to collect the primary studies. We have selected these well-known search engines and libraries to make our search comprehensive and unbiased. The formulated search string that contains the keywords and their appropriate synonyms has been validated to make sure of its feasibility to access the related studies. The star sign (*) is used to facilitate capturing similar forms of the word such as (classify, classifying, and classification). The boolean operator ORs are used between the synonyms and the boolean operator AND is used between the two sets of synonyms as in Table 3.2.

Keywords	Synonyms
Requirement classification	"classif* requirement" OR "classif* of requirement" OR "requirement classif*" OR "requirement identif*" OR "requirement categor*"
Deep learning	"deep learning" OR "deep neural network" OR "convolutional neural networks" OR "CNN" OR "recurrent Neural network" OR "RNN" OR "bert"

Table 3.2: Keywords and their synonyms

The retrieved papers have been filtered by applying the inclusion and exclusion criteria at three different stages. At the first stage, the papers were filtered by the

title only. The second stage requires reading the abstract of the papers to ensure their relevance to our scope. The selection of the primary studies was guided by inclusion and exclusion criteria. The output of the third stage is the final selected primary studies that have been assessed by the quality assessment in Table 3.3. The paper has to score 3 in the quality assessment test to be selected.

The inclusion criteria are defined as follows:

- Papers written in English language.
- Papers discuss DL techniques in requirement classification from software requirements specifications.

The exclusion criteria that are used in filtering the papers are as follow:

- Grey publications or white papers.
- Papers published in other than software or computer science domains.
- Review or any secondary papers.
- Papers use conventional ML techniques.
- Papers discuss requirement classification in app reviews or blogs.

Both forward and backward snowballing approaches have been used after collecting the start set using the main keywords [43]. The forward snowballing is gathering papers from the citation list of the examined paper. The backward snowballing is gathering paper from the reference list in the examined paper. The Summary of papers identification and selection is shown in Figure 3.2.

Criteria	Value
The paper aims to address any type of SRS sentences classification.	True = 1 False = 0
The structure and the results of the paper clearly specified.	True = 1 False = 0
The paper is published in peer-reviewed conference or journal.	True = 1 False = 0

Table 3.3: Quality assessment template

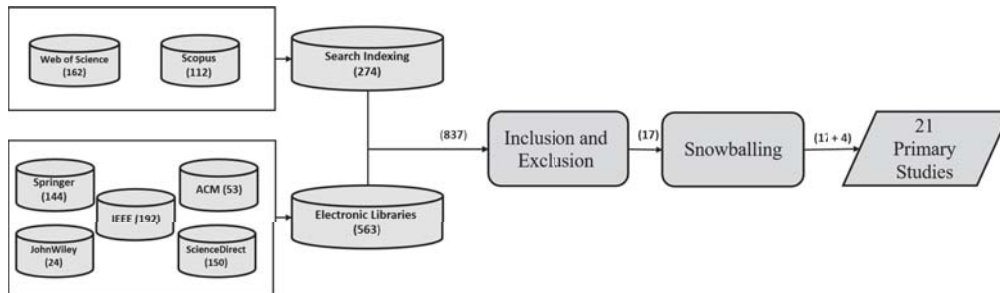


Figure 3.2: Summary of papers identification and selection

Section name	Column name
Paper details	Name of paper, publication date, authors name, name of journal or conference, publisher, and study objective.
Dataset	Name of dataset, no. of classes, and name of classes.
Preprocessing steps	Name of preprocessing steps.
Feature representation	Name of features and name of techniques.
DL algorithm	Name of algorithm, implementation tool or environment, and their hyperparameters.

Table 3.4: Headings of extraction form

To extract the required information from the papers, an excel sheet has been used with several columns to answer the first three questions such as dataset, preprocessing steps, word representation, DL model, hyperparameters, and performance measures as in Table 3.4. The data extracted from 21 primary studies that satisfied our inclusion and exclusion criteria has been synthesized and summarized. Most of the papers were published in the IEEE Xplore library. Six of the primary studies were published in the Springer Link, and two were published in the ACM digital library. The papers

distribution based on the publisher is presented in Table 3.5

Publisher	Papers count	Papers ID ¹
IEEE	13	S1, S2, S3, S4, S6, S7, S9, S12, S14, S15, S16, S18, S21
Springer	6	S8, S11, S13, S19, S17, S20
ACM	2	S5, S10

1) S refers to the selected primary study with unique number, the details appear in Appendix A

Table 3.5: Papers distribution based on the publisher

3.1.3 Results

In this section, the results and findings of the review are presented by answering the four research questions. First, the available datasets that are used in requirement classification are addressed. Next, the NLP preprocessing steps and feature representation are discussed. Then, the DL models are presented with their hyperparameters, and performance evaluation. Finally, the challenges and open research directions are presented.

The general pipeline for solving the problems of requirement classification using DL approaches is illustrated in Figure 3.3. The pipeline starts with preprocessing the annotated sentences extracted from SRS documents. Then, the input is represented by any vectorization methods that transform the raw data into a numerical vector before feeding these inputs to the DL models. Then, the classification measures are reported such as precision, recall, and f-score.



Figure 3.3: The pipeline for solving the problems of requirement classification using DL

The primary studies are classified based on their study goal into four categories as

in Table 3.6. The first category contains five studies that classify the sentences into requirement and non-requirement. In the second category, nine studies investigate the classification of requirements into functional and non-functional sub-categories as binary or multi-class problems. The availability of the public functional and non-functional datasets is one of the main reasons for a large number of publications.

Primary study objective	Study ID	No. of studies
To classify sentences into requirement and non-requirement.	S7, S11, S16, S17, S19	5
To classify sentences into F and NF requirements.	S2, S3, S4, S5, S6, S8, S9, S14, S20	9
To classify sentences into security and non-security requirements.	S3, S10	2
To classify sentences into different classes.	S1, S12, S13, S15, S18, S21	6

Table 3.6: the objectives of the primary studies

The third category with only two papers related to classifying the requirements into security/non-security. The last category contains six papers to classify the sentences into requirement/information (S1), architecturally significant functional requirements (ASFR) (S12), integration/software requirement (S13), business rules (S15), conditional sentences (S18), or implicit requirement (S21). The granularity of software requirements classification is shown in Figure 3.4.

3.1.3.1 Dataset Used and Modality (*RQ1*)

This subsection presents a comprehensive review of the datasets that are used in several papers related to requirements classification to answer the first research question. We used the SLR to identify the available datasets and how the researchers use it in their studies. After that, we investigate the

In the literature, the PROMISE dataset is the most well-known dataset for clas-

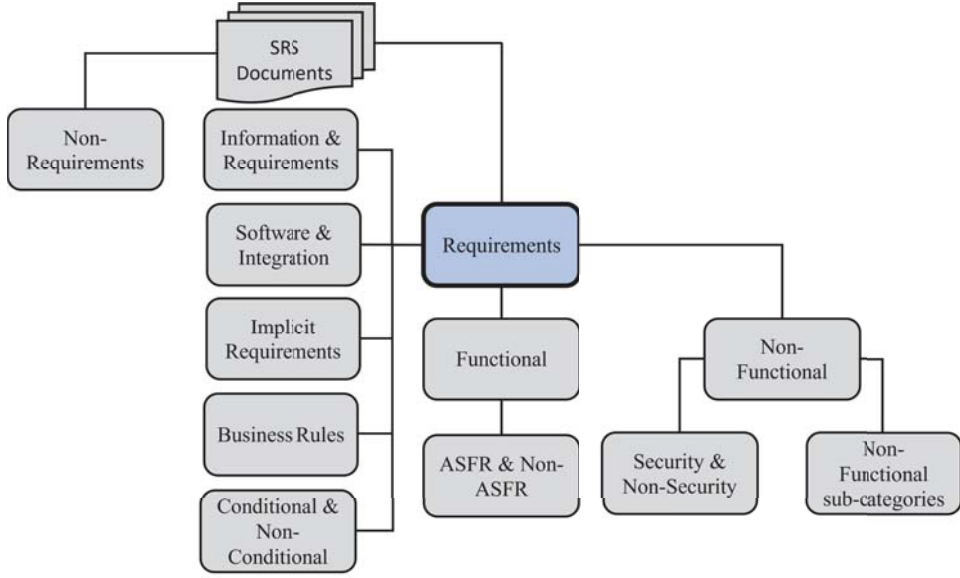


Figure 3.4: Different granularity in classifying software requirements

sifying requirements [8]. The dataset consists of 15 software projects written by MS University students in the Requirement Engineering course at DePaul University [5]. The requirements are labeled as either functional or non-functional sub-categories. The PROMISE dataset is used in the literature with different modalities (two, four, five, or ten classes) that complicate the comparison of their classification results. Several versions and modalities of the PROMISE dataset have been used as binary, multi-class with one label, or multi-class with multi-labels problems.

The SeqReq dataset consists of three industrial documents that are labeled with only one label either security-related or non-security-related requirements [44]. The dataset contains 177 labeled sentences from the Common Electronic Purse (ePurse), 124 labeled sentences from the Customer Premises Network (CPN), and 210 labeled sentences from the Global Platform Specification (GPS).

The CONCORDIA RE dataset is annotated with multi-labels, based on a requirements ontology modeled using the Web Ontology Language (OWL) in [4]. The dataset

is used in requirements classification tasks and contains not a requirement, functional, and six NFR classes (constraints, security, usability, efficiency, functionality, and reliability).

Another dataset Electronic Health Records (EHR) that contains several projects from the health domain is presented in [45] were used for the evaluation of the proposed models. The EHR dataset consists of 12 different projects related to the health domain. The sentences are annotated with multi-label of 14 NFR, and one functional.

Six requirements artifacts written in natural language from the electronic health-care domain are annotated with multi-label that are categorized into six security classes: accountability, confidentiality, integrity, privacy, identification & authentication, availability, and none security class [46]. The dataset can be used for both security/non-security and security sub-categories classification tasks. Three software requirement specifications **SmartTrip**, **SafeDrink**, and **BlueWallet** from different domains are used for identifying the functional requirements [47].

The available datasets that are addressed mostly are annotated by functional or non-functional labels. Some of them were labeled by several categories related to the security domain. Two popular sources of academic and industrial software requirements specification **PURE** and **RE corpus** that need to be annotated are available in [48], [49]. In Table 3.7, the detailed information about the datasets is presented.

The PROMISE dataset is the most popular in the RE community for requirement classification task. Different versions and modalities have been used either binary (F and NFR) or multi-class (four, ten, and others). The details about the different versions and modalities of using the PROMISE dataset are shown in Table 3.8.

Dataset used	No. of documents	No. of samples	Source	Type	Labels	Modality *	Studies ID
PROMISE	15	625	Academic	SRS	One label NFR	F, A, LF, L, MN, O, PE, SC, SE, US, FT	[50], [51]
SeqReq (ePurse + CPN + GP)	3	510	Industry	SRS	One label security	Security-relevant and non-security	[44]
Concordia RE	6	3064	-	SRS, supplementary specification and use case	Multi label NFR	NR, F, CO, SE, US, EF, FU, RE	[4], [51]
EHR (CCHIT, iTrust, Promise, others)	12	11876	Both	SRS, RFP, IM, DUA, CFR	Multi label NFR	F, AC, AU, A, L, LF, MN, O, PR, PC, RE, RC, SE, US, OT	[45]
Health artifacts	6	10963	Industry	Requirement artifacts	Multi label security	C, I, IA, A, AY, PR, None	[46]
SmartTrip + SafeDrink + BlueWallet	3	568	Industry	SRS	One label NFR	F	[47]
PURE	79	34,268	Both	SRS	Not labeled	-	[48]
RE corpus	463	212,186	Both	SRS	Not labeled	-	[49]
* Functional (F), non-functional (NF), availability (A), legal (L), look and feel (LF), maintainability (MN), operational (O), performance (PE), scalability (SC), security (SE), usability (US), fault tolerance (FT), performance and capacity (PC), reliability (RE), recoverability (RC), privacy (PR), access control (AC), audit (AU), constraints(CO), efficiency (EF), functionality (FU), not-requirement (NR), confidentiality (C), integrity (I), accountability (AY), identification and authentication (IA), and Others (OT).							

Table 3.7: Available software requirement datasets

Every dataset contains a set of classes such as functional or non-functional classes as illustrated in Table 3.9. The number of samples of classes in every dataset is shown in Table 3.10 describes mainly the classes that are related to security only. Some classes are more frequent in the dataset than others such as functional, security, and usability. On the contrary, some classes have a limited number of samples among the different datasets such as portability, efficiency, recoverability, scalability, and reliability.

3.1.3.2 Text Preprocessing & Feature Representation (*RQ2*)

To facilitate the classification task, several preprocessing techniques are used on natural language texts to minimize the noise and improve the performance of the DL

Dataset used	No. of documents	No. of samples	Source	Type	Labels	Modality **	Studies ID
PROMISE	15	684	Academic	SRS	One label NFR	F, A, L, LF, MN, O, PE, SC, SE, US	[52]
	15	765	Academic	SRS	One label NFR	F, NF	[9]
	15	625	Academic	SRS	One label NFR	F, NF	[53], [54]
	15	625	Academic	SRS	One label NFR	US, SE, LF vs all	[55]
	15	625	Academic	SRS	One label NFR	US, SE, O, PE	[53]
	15	625	Academic	SRS	One label NFR	F, A, LF, L, MN, O, PE, SC, SE, US, FT	[50], [51]
	15	326	Academic	SRS	One label NFR	A, L, LF, MN, O, PE, SC, SE, US	[4]
	15	370	Academic	SRS	One label NFR	A, L, LF, MN, O, PE, SC, SE, US, FT	[30]
	15	370	Academic	SRS	One label NFR	A, L, LF, MN, O, PE, SC, SE, US, FT, PO	[56]
	-	58	Academic	SRS	One label NFR	AA, AC, CE, DI	[57], [58]
	15	792	Academic	SRS	Multi label NFR	F, AC, AU, A, L, LF, MN, O, PR, PC, RE, RL, SE, US, OT	[45]
** Functional (F), non-functional (NF), availability (A), legal (L), look and feel (LF), maintainability (MN), operational (O), performance (PE), scalability (SC), security (SE), usability (US), fault tolerance (FT), access control (AC), audit (AU), privacy (PR), performance and capacity (PC), reliability (RE), recoverability (RC), authentication & authorization (AA), cryptography - encryption (CE), data integrity (DI), and Others (OT).							

Table 3.8: Versions and modalities of PROMISE dataset

models. Using the word/feature representation can capture the syntax and semantic of the words. The following subsections present findings to answer the second research question.

3.1.3.3 Text Preprocessing

Before classifying the requirements, there are a number of NLP preprocessing steps that can be utilized. The main advantages of using the preprocessing steps are minimizing the noise and meaningless tokens in the dataset, and improving the performance and efficiency of the model. In addition, the preprocessing steps decrease the vocabulary size by unifying the structure of the words. Below, some of the common techniques of NLP preprocessing:-

Dataset	Attribute																							Total
	Access Control (AC)	Audit (AU)	Availability (A)	Look and feel (LF)	Legal (L)	Maintainability (M)	Operational (O)	Privacy (PR)	Performance (PE)	Scalability (SC)	Recoverability (RC)	Security (SE)	Usability (US)	Portability (PO)	Features (FE)	Constraint (CO)	Efficiency (EF)	Functionality (FU)	Reliability (RE)	Non-Functional (NF)	Functional (F)	Others (OT)	Not-Requirement (NR)	
PROMISE v1 [51]	-	-	18	35	10	16	61	-	48	18	-	58	62	-	-	-	-	-	-	-	475	358	-	684
PROMISE v1 [9]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	270	745	-	-
PROMISE v2 [50]	-	-	21	38	13	17	62	-	54	21	-	66	67	1	10	-	-	-	-	-	255	-	-	625
Concordia RE [4]	-	-	-	-	-	-	-	-	-	-	-	57	28	-	-	110	14	16	9	-	787	-	2122	3140
PROMISE ML [45]	164	20	36	50	10	26	89	7	75	4	71	101	-	-	-	-	-	12	-	340	19	0	792	
EHR [45]	979	276	57	68	152	413	207	300	100	-	50	563	148	-	-	-	-	-	43	-	3568	82	6076	11876
SmartTrip + SafeDrink + BlueWallet [47]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	568	-	-	568

Table 3.9: Distribution of attributes in every dataset for functional and non-functional requirements

	Total											
	Confidentiality (C)	Integrity (I)	Identification & Authentication (IA)	Availability (A)	Accountability (AT)	Privacy (PR)	Cryptography - Encryption (CE)	Access Control (AC)	Authentication & Authorization (AA)	Security (SE)	Non-security (NSE)	510
												10963
												58
ePurse + CPN + GP [59]	-	-	-	-	-	-	-	-	-	187	323	510
Health [46]	3016	3289	258	135	3787	204	-	-	-	5050	5913	10963
SecPROMISE [57]	-	15	-	-	-	-	13	12	18	-	-	58

Table 3.10: Distribution of classes in every dataset for security requirements

- **Tokenization:** dividing a sentence into a sequence of tokens.
- **Punctuation removal:** removing the punctuation symbols such as comma and question mark.
- **Stop word removal:** removing the repeated words that have no distinct meaning such as auxiliary verbs, conjunctions, or articles.
- **Case folding:** changing all the characters to be lowercase to treat similar words with different character cases equally.
- **Normalization:** converting the word to its standard form.
- **Lemmatization:** converting the word to its lemma form by using the morphological analysis of the words.
- **Stemming:** converting the word to its stem form by removing the prefix and suffix.

The frequency of using NLP techniques in the primary studies is shown in Table 3.11. The highest frequency of usage is the tokenization and stop word removal steps. Then, removing the punctuation to minimize the tokens without unique meaning is applied four times in the primary studies. Case-folding will be used because the upper case and lower-case words have the same meaning. Without using case-folding, the similar words will be treated separately and vocabulary size will be increased.

Lemmatization is used four times in the primary studies (S5, S12, S14, and S16) compared with stemming which is used only once in (S6). Both stemming and lemmatization can improve the classifier performance and minimize the number of similar

words with different forms in the vocabulary. Comparing stammer with lemmatizer, stemmer is simpler and faster than lemmatizer which requires extensive linguistics knowledge to build the lookup table that is used by the algorithm to generate the lemma of the word.

NLP technique	Study ID	No. studies
Tokenization	S5, S6, S9, S13, S12, S14, S16, S17	8
Punctuation removal	S3, S12, S14, S16	4
Stop word removal	S1, S3, S5, S6, S12, S14, S16, S17	8
Case folding	S1, S3, S5, S6, S14, S16	6
Normalization	S17	1
Lemmatization	S5, S12, S14, S16	4
Stemming	S6	1

Table 3.11: NLP techniques in the primary studies

3.1.3.4 Feature Representation

Since the input of DL models is the SRS documents that contain natural language words, the feature representation is required by converting the words to a form that can be processed by the classifiers. Capturing implicit linguistic rules and common-sense knowledge that are hidden in the text are the main characteristics of a good representation of the features (words) [60]. There are some variations between feature representation techniques in their capability to capture syntax, semantic, and context-based semantics.

The traditional features representation depends on the frequency of the words such as term frequency (TF) or bag of words (BOW). Instead of using the number of word occurrences, the weight of every word is calculated where the unique words have a higher weight than the frequent words as in term frequency-inverse document

frequency (TF-IDF). The aforementioned simple traditional feature representations can capture neither the position nor the semantic of the words based on their context.

Word representation	Study ID	No. studies
TF	S4, S14	2
TF-IDF	S14	1
Word2vec	S1, S2, S3, S5, S8, S14, S15, S16, S17	9
Glove	S12, S13, S17	3
BERT	S9, S14, S18	3

Table 3.12: Feature representation in the primary studies

In recent years, the text of natural language sentences has been transformed using DL models into low-dimensional real-valued vectors which are called distributional vectors (word embeddings). The word embedding has been proposed to capture the similarity between the words. The pre-trained models Word2Vec [17] and Glove [61] are used to generate a vector for every word. The issues with these models are the lack of handling out of vocabulary (OOV) problem and their representations of the words are not considering different meanings or positions for a specific word.

To tackle the OOV problem, the fastText [62] model has been proposed that is based on n-grams to generate the embeddings of unknown words. Considering the context of the word, BERT [18] and ELMo [63] generate different representations for the same word based on their context. As shown in Table 3.12, there is limited usage of the traditional features representation. The Word2vec model is the most used word representation and three recent papers (S9, S14, and S18) used the BERT model. The weights of the embedding can be initialized randomly or with a pre-trained model that can assign the right weight to the embedding of the vocabulary. Based on the work in S3, the random word embedding scored less than Wor2Vec embedding which boosts

the scores and improves the overall performance of CNN. Sometimes the embedding is trained on the dataset as in S12 and the pre-trained models can be trained on an external dataset if the domain is not different as in S13. The advantage of using random weight initialization is the robustness and actual weights computed from the local dataset, but it jeopardizes the generality of the representation and requires more iterations to be trained. Pre-trained models are capable of convergence faster than the models from scratch but do not necessarily improve the classification results [64]. The tool TensorFlow supports and facilitates the implementation of the CNN and RNN as in S1, S3, and S12 which is "an open-source software library for numerical computation using data flow graphs" and Keras as in S5, S12, S13, and S16. The Google Colab environment has been used in S14.

3.1.3.5 DL Algorithms (RQ3)

Deep learning is a deep neural network with multiple-layered that contains neurons to learn [11]. There are main approaches to implement deep learning networks using convolutional neural networks (CNN), recurrent neural networks (RNN), recursive neural networks, or transformers. In the following, we will answer our third research question related to DL algorithms, their hyperparameters, and performance measures.

Based on Table 3.13, the CNN approach is the most commonly used among the other deep learning techniques. The two variants of RNN are used in three studies S5, S10, and S17 with one direction. The bidirectional RNN is used in three studies of primary studies. Only one study used the tree recursive neural networks and two studies used the transformers BERT.

DL algorithm	Study ID	No. of studies
CNN	S1, S2, S4, S5, S6, S7, S8, S10, S11, S13, S14, S17, S20, S21	14
RNN-LSTM	S5, S10, S17	3
RNN-GRU	S5	1
RNN-BiLSTM	S12, S15, S16	3
Tree recursive NN	S18	1
BERT	S9, S16	2

Table 3.13: DL algorithms in the primary studies

3.1.3.6 DL algorithms & their hyperparameters

In the following three subsections, a brief description about every DL approach is presented and optimal values for the hyperparameters are reported with different ways to search for them.

3.1.3.6.1 *Convolutional Neural Network (CNN):-*

3.1.3.6.2 Since convolutional neural network (CNN) showed effective and competitive results in the computer vision field, CNN architecture was adopted for classifying natural language sentences [14]. The sentences are converted to a matrix that represents the tokenized words by using word embedding representations such as word2vec or GloVe. Then the filters will perform the convolution process to produce a single value for each specific area in the sentence matrix. This process is called convolving which produces the convolved features of one feature map. Then the pooling step is implemented which subsamples the feature map to produce one value either by maximum or mean criteria. The pooled values are concatenated to form a single feature vector which is correlated with the probability distribution by using the softmax layer. The main drawback of CNN is the lack of capability to preserve sequential information and the dependency between the tokens in long sentences.

The best values of the hyperparameters that are tuned in CNN models are shown in Table 3.14. The maximum value of embedding size is 400 as in S2, the minimum value is 50 as in S13, and the remaining studies are between 128 and 300. The size of the vocabulary is limited in both studies S13 and S14. Only two studies report the sequence length which is 100 as in S13 or 50 as in S14. The common filter size that is used is 2 and 3 as in S1, S14, and S17. The number of filters varies from one study to another based on the complex nature of inputs. All studies used max-pooling in choosing the value to be in the feature map. The number of epochs depends on the problem and the dataset for example in S1 the model reached 100% after only 10 epochs. The common dropout value is 0.5 in most primary studies. The batch size is different in all reported primary studies.

ID	Embedding size	Vocabulary	Sequence length	Filter size	No. of filters	Pooling	No. of epochs	Dropout	Learning rate	Batch size	Parameters
S1	128	NA	NA	1, 2, 3	192	NA	10	NA	NA	NA	NA
S2	400	NA	NA	NA	NA	max	100	0.5	0.001	NA	NA
S3	128, 300	NA	NA	3, 4, 5	30, 50	max	20, 40, 60, 80, 100	0.5	NA	64	NA
S4	128, 300	NA	NA	4	128	max	NA	NA	NA	NA	NA
S6	150	NA	NA	3	32	max	250	0.5	NA	20	NA
S13	50	1878, 1720	100	NA	NA	max	20, 50	NA	NA	10	18801, 86521
S14	300	1247	50	2, 3, 4	200	NA	20	0.05	NA	32	NA
S17	100, 300	8404	20	2, 3, 4	NA	max	NA	0.5	NA	NA	NA

Table 3.14: CNN hyperparameters

3.1.3.6.3 *Recurrent Neural Network (RNN):-*

3.1.3.6.4 For handling sequential information, the RNN has been used where the recurrent unit is implemented to do its task over the whole sequence of inputs [65].

The results of the second input depend on the output of the previous input. In the literature, the RNN architecture has been widely used in several NLP tasks [66]. The issue of the vanishing gradient hardens the learning and tuning of the RNN models. The most used variants of RNN are long short-term memory (LSTM) [67] and gated recurrent units (GRU) [68] have been proposed to overcome the vanishing gradient problem.

The complexity of the GRU is less than the LSTM variant by implementing only two gates (reset and update) instead of three gates (input, output and forget) in LSTM to compute the hidden state. The two variants outperform the RNN, but the performance of GRU and LSTM is comparably the same with less computation power in GRU models. Processing the input in forward and backward directions can improve the performance by using Bidirectional Recurrent Neural Network (Bi-RNN) [69]. An attention-based RNN architecture used in S12 can maximize the efficiency by utilizing the essential parts of the inputs [70]. The details of RNN architecture and their variants can be found in [71].

The hyperparameters are tuned in RNN based on the type of the problem and the size of the dataset. Table 3.15 illustrates the best values of hyperparameters of several RNN models. The best value for an embedding size is either 300 or 50 and the number of tokens in the studies is not reported except S12 which is 4200. The longest sequence of input is 100 tokens as in S16. The other studies reported input in the range between 30 and 50. Generally, the number of epochs for training the RNN is high if the problem is multi-class classification or the model is unidirectional. The dropout value is 0.5 in most studies to prevent overfitting due to the limited number

of samples in the datasets. The model checkpoint and early stopping condition are used only in S12. Three studies used a 0.002 value as a learning value and two used 0.001 in their experiments. The batch size and the number of parameters are reported only in two different studies.

ID	Embedding size	Vocabulary	Sequence length	No. of epochs	Dropout	Model checkpoint	Early stopping	Learning rate	Batch size	Parameters
S5	300	NA	NA	200	0.5	NA	NA	0.001	NA	1,144,810
S12	300	4200	34, 30	30, 200	0.2, 0.1	Yes	Yes	0.001	NA	NA
S15	50	NA	36, 40	50, 300	0.5	NA	NA	0.002	NA	NA
S16	50	NA	100	50	0.5	NA	NA	0.002	NA	NA

Table 3.15: RNN hyperparameters

3.1.3.6.5 *Transformers:-*

3.1.3.6.6 A new model architecture has been proposed in [16] which is called a transformer. The new architecture can compute the representation of the inputs and the output without relying on RNN or CNN. The architecture is based on stacking self-attention which is mainly dependent on it for both encoder and decoder with a fully connected layer between them. Bidirectional Encoder Representations from Transformers (BERT) is one of the transformers that use Masked Language Model (MLM) to overcome the unidirectional limitation in other models [18].

The hyperparameters of BERT-based models are tuned as shown in Table 3.16. The sequence length value is 128 in S9 for base models and S16. The complexity of the transformers and several layers reduce the number of epochs in the training to be

between 10 and 64. The learning rate is 2E-5 in both two primary studies S9 and S16.

Early stopping condition is used in S9 to prevent the models' overfitting.

ID	Sequence length	No. of epochs	Learning rate	Early stopping	Batch size	Parameters
S9	128 base models, 50 large models	10 -32 binary 10-64 mult-iclass	2.00E-05	Yes	16	110M base model and 340M large model cased version
S16	128	25	2.00E-05	NA	20	NA

Table 3.16: Transformers hyperparameters

The performance of DL models depends on selecting suitable values of hyperparameters. Usually, the hyperparameters are tuned using the iterative process as in S1 where the tuning of hyperparameters starts with small values until the performance of the network stops to improve. Another approach is used for tuning is using Scikit-Learns inbuilt RandomizedSearch function [72] as in S12. The Grid search method for tuning the hyperparameters is used in S16.

In primary study S4, the meta-heuristic Binary Particle Swarm Optimization (BPSO) algorithm is used to find the optimal parameters of the DL models. Generally, the DL models use the loss function which quantifies how close the predicted distribution is to the true distribution. In binary or multi-label classification, the loss function is binary cross-entropy while in the multi-class problem, the categorical cross-entropy is used. All the primary studies used Adam optimizer in their models.

3.1.3.7 Performance Measures

In general, the performance of DL models outperforms the conventional ML models and achieves state-of-the-art results in different domains. Table 3.17 illustrates the details of the several experiments in the primary studies such as performance measures, number of classes, validation technique, and the used dataset. What is interesting about the data in this table is the high variation between them that maximizes the difficulty of comparison. Similar to conventional ML, the accuracy, precision, recall, and f-score metrics are used in both one-label and multi-label classification problems. For multi-label classification, hamming, zero-one, and label ranking losses are computed as in S12.

The confusion matrix is one of the ways to visualize the performance of the classifiers. It allows showing the types of successes and errors of models in the validation stage. The performance of the classifier is affected by the number of classes and the difficulty of classifying the sentences into several classes. For example, the minimum f-score is achieved when the model classifies the inputs into 14 classes as in S16 while the maximum f-score is achieved when the problem is binary classification as in S3, S9, and S17.

Generally, the performance of classification problems is evaluated using precision, recall, and f-score. There is a limited number of studies using the confusion matrix in reporting their results as in Figure 3.5(b). From the data in Figure 3.5(a), it is apparent that classifying the inputs into binary classes has been investigated more than the multi-class classification problem. The reasons might be increasing the number

ID	DL	Accuracy	Precision	Recall	F1	CM	Type	Validation	Dataset
S1	CNN	0.81	0.82	0.82	0.81	No	Binary	90 -10	89 requirements specifications (10000 content elements)
S2	CNN	-	0.81	0.79	0.77	Yes	12 classes	10-Fold	PROMISE
S3	CNN	0.9105	0.9152	0.9138	0.9134	No	Binary	10-Fold	SeqReq
		0.9259	0.9268	0.9187	0.9216	No	Binary	10-Fold	PROMISE
S4	CNN	0.83	-	-	-	No	Binary	90-10	PROMISE
S5	LSTM	0.715	0.717	0.715	0.7	No	10 classes	10-Fold	PROMISE
S6	CNN	-	0.82 0.94	0.76 -0.97	0.82 -0.92	No	5 classes	Stratified 10-Fold	PROMISE & SeqReq
S7	CNN	-	0.81	0.83	-	No	Binary	10-Fold	20000 requirements, 15000 non-requirements
S9	BERT	-	0.92	0.92	0.92	No	Binary	Stratified 10-Fold	PROMISE
		-	0.79	0.77	0.82	No	10 classes	Stratified 10-Fold	PROMISE
S10	LSTM	0.84	-	-	-	No	Binary	10-Fold	SeqReq
S12	Bi-LSTM	-	0.85	0.88	0.86	No	Binary	Stratified 4-fold 70:30	124 SRS, 9233 sentences
		-	0.88	0.79	0.83	No	12 classes	Stratified 5-fold 70:30	1782 sentences
S13	CNN	0.86	-	-	-	No	Binary	75-25	2000 requirements
S14	CNN	0.91	0.92	0.91	0.92	Yes	6 classes	70-30	1247 requirement
S15	Bi-LSTM	-	0.88	0.87	-	-	Binary	5-fold	40 SRS, 4043 sentences
		-	0.77	0.67	-	-	10 classes	5-fold	1676 sentences
S16	Bi-LSTM	-	0.9	0.89	0.89	No	Binary	Stratified 5-fold	18,614 sentences
		-	0.74	0.61	0.66	No	14 classes	Stratified 5-fold	5,472 sentences
	BERT	-	0.82	0.82	0.81	No	14 classes	Stratified 5-fold	5,472 sentences
S17	CNN	-	0.93	0.93	0.93	No	Binary	10-Fold	11060 sentences
S20	CNN	-	0.81	0.77	0.79	No	3 classes	-	5000 samples

Table 3.17: performance results of DL models

of the classes will increase the complexity of the models and produces poorer results [73]. In addition, multi-class classification of small datasets with a limited number of samples in every class will complicate the task.

Due to the limited number of samples in the existing datasets, all of the primary studies split the dataset into training and validation only without using the independent testing part. The 10-fold cross-validation approach is frequently used with the stratification improves the performance of the classifiers as shown in Figure 3.6(b). Nine of the reported different datasets are private which prevents other researchers from using their dataset for the sake of improvement and comparison. The PROMISE dataset that contains functional and non-functional requirements is used in 6 primary studies as in Figure 3.6(a).

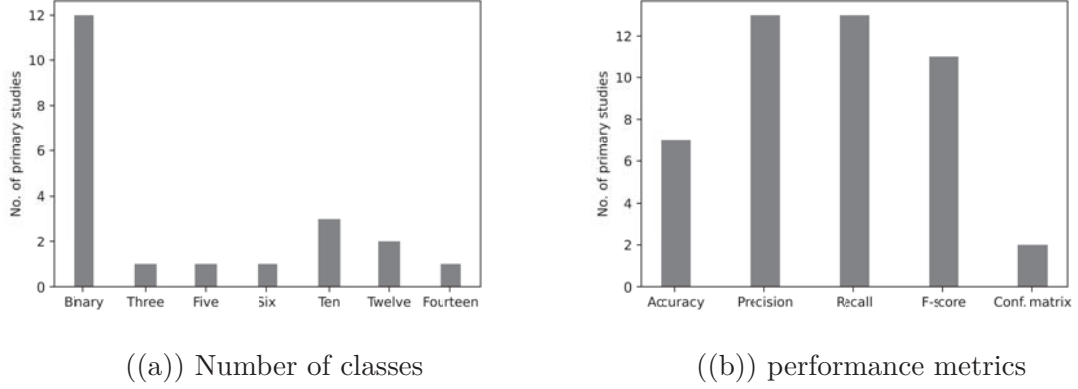


Figure 3.5: Number of classes and used performance metrics in the primary studies

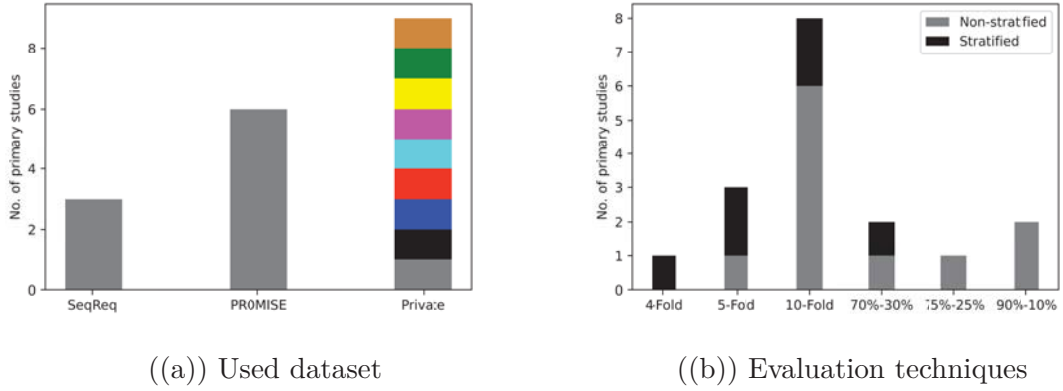


Figure 3.6: Used dataset and evaluation techniques in the primary studies

3.1.4 Open Challenges for Future Research

In this section, we address the limitations and issues to answer the fourth question.

The limitations and issues are illustrated and categorized into several parts as follows:

3.1.4.1 Issues related to the dataset

DL techniques can process large datasets perfectly, however, all publicly available datasets are considered as small datasets. Annotating different SRS from different domains by domain experts will advance the research in the requirement classification field. The main challenge is that labeling the dataset manually is time and effort-

consuming.

Since labeling manually is a challenging task, several techniques have been proposed to replace manual annotations with automatic labeling such as unsupervised, semi-supervised, or active learning. Unsupervised techniques can be used to find hidden patterns in the unlabelled dataset. The active learning technique can interact with the user to label the unlabelled samples.

Few studies have investigated the multi-label requirements classification problem. The requirement can have more than one label at the same time. The requirements can be separated into fine-grained sentences to minimize the number of labels in one sentence. Labeling the requirement with multi-labels is one way to mitigate the issue of labeling difficulty in some classification problems as functional and non-functional classification.

The publicly available datasets mainly target functional and non-functional requirements only. The lack of a dataset that is annotated with different labels from different scopes. The requirements can be classified into levels: high level such as usability, low level such as buffering and caching, and intermediate-level such as logging and authentication [52]. Another requirements classification that can be investigated are volatile/ stable requirements, goal-level/domain-level requirements, and product/process requirements.

All the available datasets consist of sentences only, but the SRS documents have plenty of extra information such as tables, headings, and figures. Manual processing is used to extract requirements from a document. Automating the extraction and classification of the requirements from the original SRS will minimize the burden

from the user side.

The most popular dataset is PROMISE which consists of 15 software requirements specifications written in the academic domain. Mitigating the external validity and increasing the validity, annotating SRS documents that represent the academic and industrial fields. In addition, the SRS documents are domain-specific that can represent health, finances, etc.

The difference between the trained and tested dataset distribution has not been investigated. The generality of the model needs to be improved to tackle different sources of the datasets. The dataset shift could be the cause of the high error rate when the model is tested on unseen data. The variation between the training and testing dataset can be the source (academic-industrial), domain (health - electronic commerce). Three types of shifts that occur in the independent variable (input), target variable (output), or the relationship between the independent and the target variable (input to output) can be validated.

All the available datasets are imbalanced where the difference between the number of samples of the classes is high. Some techniques can improve the results and minimize the bias of the imbalanced dataset such as class weight as in S5 S15 S16, oversampling S9, and random undersampling S9 S12 or undersampling with using clustering algorithm as in S1. The aforementioned strategies are considered simple and use the same samples in the dataset. Another way of dealing with the imbalanced or limited size of the dataset is using data augmentation, which aims at creating novel and realistic-looking training data by applying a transformation to an example, without changing its label. Several augmentation strategies can be used on texts such as

back-translation or word replacing with TF-IDF.

3.1.4.2 Issues related to the preprocessing and feature representation

To process the text by the classifier, the input needs to be converted to numbers using a tokenizer. There are three types of tokenization methods: word-based, sub-word-based, and character-based tokenizers. Examples of tokenizers that can be empirically tested are Byte-Pair Encoding (BPE), WordPiece, and SentencePiece algorithms. The impact of different NLP preprocessing steps on the results of DL algorithms has not been investigated. There are different stemmer and lemmatizer tools that might affect the final results of the DL classifiers.

The pre-trained word embedding trained on SRS source language and applied in SRS with target language have not been researched in the literature. There are several word embedding (BERT, Glove, Word2Vec, ELMo, and FastText) and several DL techniques (CNN, RNN-GRU, RNN-LSTM, RNN-BiGRU, RNN-BiLSTM, and Pre-trained models). Limited research has been found that empirically investigates the right combination between the word embedding and DL algorithms [74].

No previous study has investigated character embedding in software requirement classification. There many advantages of using character embedding instead of words such as small vocabulary to store and the lack of OOV problem. In addition, the character embedding is capable of processing the sub-words especially with misspelled words that consider unknown words in a word embedding. Hybrid embedding using word and character embedding utilizes the advantages of both levels.

3.1.4.3 Issues related to DL models

Transferring learnt knowledge in task or domain to another task or domain is called transfer learning. There are many types of transfer learning in NLP, such as domain adaptation, cross-lingual learning, multi-task learning [60]. Since we are dealing with a lack of labeled data, transfer learning is an excellent technique that can be used to train our model. Transferring the weight from the pre-trained models in rich resource problems or language to initialize the weight of the model to be trained on low-resource problems or language. In other words, it means additionally training a model that was trained for the task which is slightly different from the current.

Due to the limited number of samples in the available datasets, overfitting is one of the main challenging threads. To minimize the impact of overfitting, several regularization approaches were used such as dropout and l2 norm regularizer as in [14], [75]. The dropout is the common approach in our primary studies where some of the units are dropped (ignored) in the network. Early stopping condition is another approach to prevent the overfitting issue by monitoring the specified metric and stopping the training after a patience number of iterations without improvement. One drawback of the early stopping condition is that the approach can not guarantee the returned model is the best one. The model checkpoint is more flexible than the early stopping condition by saving the best model on the validation dataset. The early stopping condition is used in S12 S9 while the model checkpoint is used only in S12. The l2 regularizer with 0.01 has been used in S14.

One of the main challenges in software requirement classification is the long se-

quence of the input that complicates the task by encoding irrelevant information. Unsimilar to the tweets or limited length of reviews, the sentences of SRS can be tens of words. To tackle this issue, an attention mechanism is used to assign different weights to the regions of the long variable input. Combining the LSTM and attention mechanism improves the f-score in identifying and classifying Architecturally Significant Functional Requirements as in S12 and S16.

Transformer models have taken the field of natural language processing, changing it significantly. New, bigger, and better models seem to be released quite frequently, setting new benchmarks in performance across a wide variety of tasks. The transformer is a model that uses attention to boost the speed with which these models can be trained. In several tasks, the transformers outperform the google neural machine translation model. The most significant advantage, though, stems from how the transformer lends itself to parallelization. There are several transformers such as Bidirectional Encoder Representations from Transformers (BERT) [18], Universal Language Model Fine-tuning (ULMFiT) [76].

Tracing the impact of training samples on DL models' performance on testing samples is difficult due to the complex models and a large number of features. Estimating the influence of the training examples enables us to find high-value or mislabelled training data [77].

Generally, DL models need a large dataset to perform well, but sometimes the labeled samples are not available or costly. Zero-shot or few-shot learning are mainly trying to handle the problem of limited or infrequent classes which occurs in most SRS documents that contain imbalanced classes. The few-shot and zero-shot aspects

with multi-label problems are explored in [78].

3.1.4.4 Issues related to experiments setup

Generally, there is a lack of detailed description of the important hyperparameters and the seeds which prevent reproducibility. The reproducibility or replicability of the experiments is considered one of the main factors in scientific fields which help in comparing different results accurately. Since DL models have several hyperparameters, it is important to clarify all of them by using one seed for all data splitting, data sampling, used framework, and randomness of dropout. Another way of dealing with the randomness is repeating the number of experiments for specific number of times as in S14.

3.1.5 Threats to Validity

Since our research is vulnerable to several threats to validity, we are using different mitigation procedures. In this section, the threats and mitigation are elaborated in detail. The bias of selecting the primary studies is minimized by using different keywords of search strings and database searches for collecting them such as the web of science and Scopus. In addition, we used forward and backward snowballing to guarantee the relevant papers are included in our primary studies.

Another potential limitation is the subjective data extraction to answer the main research question. The variation of expressing the research questions, objectives, results, and conclusions in the primary studies might hinder the extraction of the required information. To mitigate the impact of this limitation, we used the iterative

process and matching between the selected primary studies.

3.1.6 Conclusions

Recently, deep learning approaches for classifying natural language sentences gained more attention and achieved state-of-the-art results in different fields. In this section, we surveyed the papers that applied deep learning techniques in requirements classification. PROMISE dataset is the one that is highly used in the literature. The lack of a large labeled dataset is one of the main challenges in the RE community. The methodology of applying deep learning techniques starting from preprocessing, word representation, hyperparameter tuning, and evaluation reporting have been elaborated.

Representing features using common statistical text representation such as (BoW) have not been used widely compared with complex representation as (Word2Vec). The CNN models are the most frequent deep learning classifiers and achieved high results in different problems of requirements classification. Several gaps were addressed in the literature and new directions were highlighted regarding the structure of the dataset or the DL models.

In conclusion, this review has explored several opportunities to improve the quality, generality, and performance of the DL requirement classifiers. The findings suggest several courses of action for unifying the dataset usage and performance measures to facilitate reproducibility and comparison.

3.1.7 Primary Studies List:

- S1: J. Winkler and A. Vogelsang, Automatic Classification of Requirements Based on Convolutional Neural Networks, in 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW), Beijing, China, Sep. 2016, pp. 3945, doi: 10.1109/REW.2016.021.
- S2: R. Navarro-Almanza, R. Juarez-Ramirez and G. Licea, "Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification," 2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT), Merida, Mexico, 2017, pp. 116-120, doi: 10.1109/CONISOFT.2017.00021.
- S3: A. Dekhtyar and V. Fong, RE Data Challenge: Requirements Identification with Word2Vec and TensorFlow, in 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, Sep. 2017, pp. 484489, doi: 10.1109/RE.2017.26.
- S4: Bisi, Manjubala, and Kirti Keskar. "CNN-BPSO approach to Select Optimal Values of CNN Parameters for Software Requirements Classification." 2020 IEEE 17th India Council International Conference (INDICON). IEEE, 2020
- S5: Rahman, Md Abdur, et al. "Classifying non-functional requirements using RNN variants for quality software development." Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation. 2019.

- S6: Baker, Cody, et al. "Automatic multi-class non-functional software requirements classification using neural networks." 2019 IEEE 43rd annual computer software and applications conference (COMPSAC). Vol. 2. IEEE, 2019.
- S7: Winkler, Jonas Paul, Jannis Grönberg, and Andreas Vogelsang. "Optimizing for recall in automatic requirements classification: An empirical study." 2019 IEEE 27th International Requirements Engineering Conference (RE). IEEE, 2019.
- S8: Sabir, Maliha, Christos Chrysoulas, and Ebad Banissi. "Multi-label Classifier to Deal with Misclassification in Non-functional Requirements." World Conference on Information Systems and Technologies. Springer, Cham, 2020
- S9: Hey, Tobias, et al. "NoRBERT: Transfer learning for requirements classification." 2020 IEEE 28th International Requirements Engineering Conference (RE). IEEE, 2020
- S10: A. Kobilica, M. Ayub, and J. Hassine, Automated Identification of Security Requirements: A Machine Learning Approach, in Proceedings of the Evaluation and Assessment in Software Engineering, Trondheim Norway, Apr. 2020, pp. 475480, doi: 10.1145/3383219.3383288.
- S11: J. P. Winkler and A. Vogelsang, What Does My Classifier Learn? A Visual Approach to Understanding Natural Language Text Classifiers, in Natural Language Processing and Information Systems, vol. 10260, F. Frasincar, A. Ittoo, L. M. Nguyen, and E. Métais, Eds. Cham: Springer International Publishing,

2017, pp. 468479.

- S12: R. Chatterjee, A. Ahmed, and P. R. Anish, Identification and Classification of Architecturally Significant Functional Requirements, in 2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), Zurich, Switzerland, Sep. 2020, pp. 917, doi: 10.1109/AIRE51212.2020.00008.
- S13: F. Petcuin, L. Stnescu, and C. Bdic, An Experiment on Automated Requirements Mapping Using Deep Learning Methods, in Intelligent Distributed Computing XIII, vol. 868, I. Kotenko, C. Badica, V. Desnitsky, D. El Baz, and M. Ivanovic, Eds. Cham: Springer International Publishing, 2020, pp. 8695.
- S14: Shreda, Qais A., and Abualsoud A. Hanani. "Identifying Non-functional Requirements from Unconstrained Documents using Natural Language Processing and Machine Learning Approaches." IEEE Access (2021)
- S15: P. R. Anish, A. Sainani, A. Ahmed, and S. Ghaisas, Implementation-Centric Classification of Business Rules from Documents, in 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), Jeju Island, Korea (South), Sep. 2019, pp. 227233, doi: 10.1109/REW.2019.00047.
- S16: Sainani, Abhishek, et al. "Extracting and Classifying Requirements from Software Engineering Contracts." 2020 IEEE 28th International Requirements Engineering Conference (RE). IEEE, 2020.
- S17: Jeon, Kahyun, Ghang Lee, and H. David Jeong. "Classification of the Re-

quirement Sentences of the US DOT Standard Specification Using Deep Learning Algorithms." International Conference on Computing in Civil and Building Engineering. Springer, Cham, 2020

- S18: Fischbach, Jannik, et al. "Towards Causality Extraction from Requirements." 2020 IEEE 28th International Requirements Engineering Conference (RE). IEEE, 2020.
- S19: Winkler, Jonas Paul, and Andreas Vogelsang. "Using tools to assist identification of non-requirements in requirements specificationsa controlled experiment." International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, Cham, 2018.
- S20: Merugu, R. Raja Ramesh, and Satyananda Reddy Chinnam. "Automated cloud service based quality requirement classification for software requirement specification." Evolutionary Intelligence (2019): 1-6
- S21: Onyeka, Emebo, Vaibhav Anu, and Aparna S. Varde. "Identifying implicit requirements in SRS big data." 2019 IEEE International Conference on Big Data (Big Data). IEEE, 2019

3.2 Arabic Software Requirements Classification

The literature have been surveyed to explore the related work on automated software requirements classification, Arabic sentences classification, and different application that used automatic machine translation.

Previous studies have explored the capability of **rule-based methods** that based on linguistic analysis in software requirements classification. The rules that can capture different syntactic patterns to classify the sentences into groups or categories. In [5], [79], the researchers proposed rule-based approach that classify the requirements using terms indicators and their probability. They used soft-goal interdependency graphs to extract the list of indicator terms for each category. A threshold has been proposed to classify the requirements based on their calculated probability that indicates keyword density. They achieved an overall recall of 79.9% and poor precision of 42.5%.

An approach Text2Policy was proposed in [80] that used syntactic and semantic analysis to extract security sentences from software specification documents. The model used rules to validate the correctness and consistency between the access requests with access control policies in the document. The Text2Policy model score an F-score 0.89 in detecting access control sentences and accuracy 0.86 in extracting rules from iTrust and IBM projects. Although the rule-based approaches have several advantages but the capability to tackle large number of samples is limited.

The vast majority of recent studies on software requirements classification have applied a **machine learning approaches** due to their capability of evolving and adopting statistical rules. Recently, a review based on 24 primary studies was conducted in identification and classification of NFR using traditional ML techniques [8]. The review found that SVM and naive bayes (NB) are the most classifiers had been used in the literature. Stemming and stop words removal were the highest usage among NLP steps for text preprocessing.

In classifying the requirements into functional and non-functional requirements, a supervised machine learning using different features such as meta-data, lexical, and syntactical features scored high precision and recall of up to 92.5% in [10]. The SVM classifier was received preprocessed phrases that had been stripped of punctuation and stop words, as well as lemmatized words. The sampling techniques have been used along with feature selection to minimize the large number of features. The binary SVM classifier achieves an F-score between 0.72 and 0.90 while the multi-class SVM classifier achieves an F-score between 0.70 and 0.82.

The well-known PROMISE dataset had been annotated to be four categories instead of only functional and non-functional requirements in [27]. The four categories are functional, quality, both functional and quality, or neither functional nor quality. The classifier that similar to [10] had been assessed in different experimental settings to be able to interpret the classification results by exploring the classifier's inner workings. by using the simple and general rules of linguistic features, the researchers proposed interpretable classifier that enables to derive 17 linguistic features.

A comparison between supervised and unsupervised machine learning in classifying the non-functional requirements have been presented in [81]. The results showed that supervised classifier Binarized Naïve Bayes (BNB) performed better than unsupervised classifiers. The impact of advanced preprocessing had been investigated by comparing the performance of BNB classifier on processed and unprocessed inputs. The advanced preprocessing improved the precision from 45% to 90%. The classifier in [9] was utilized after the features were extracted from an unprocessed and processed dataset. The classifier was able to classify a processed dataset 95.04% with a 4 percent better

F-score than an unprocessed one 89.92% in binary classification. The goal of this preprocessing was to reduce inconsistencies across requirements and enhance phrases with related properties.

Recently, researchers have shown an increased interest in applying **deep learning** techniques. The Artificial Neural Network (ANN) and CNN have been proposed to classify five non-functional categories: operability, performance, maintainability, usability, and security in SeqReq and PROMISE datasets [82]. The CNN outperformed the ANN with precision ranging from 82% to 94%, recall ranging from 76% to 97%, and F-score ranging from 82% to 92% according to their findings.

The CNN model was proposed to classify 10,000 sentences extracted from 89 software requirement specifications as requirement or information in [32]. The words were converted to numerical vectors using word2vec and the model achieved an F-score of up to 92%. They used the known terms retrieved from the dataset to interpret the classification of the model. A comparison between different artificial neural networks such as convolutional neural network (CNN) and two recurrent neural networks (RNN) variants, Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) have been used to classify nonfunctional requirements [13]. To increase the performance of the classifier, the researcher employed the word2vec method and pre-processing techniques such as lemmatization and stop words removal. The LSTM outperformed CNN and LSTM with an average accuracy, precision, recall, and F1-score of 71.7%, 71.5%, 70%, and 71.5% respectively.

Since most of the datasets in software requirements classification are small and the proposed models are not generalizable, the researchers proposed transformers-based

approaches in [30]. The proposed models were evaluated in classifying requirements into function and non-functional and classifying the non-functional requirements into several sub-categories. The transformers-based models outperform all other models that do not rely on manual preprocessing. Using transfer learning techniques, the result has been improved by 15% in some tasks. Apparently, all the previous reported papers have been targeting different SRS documents written in English language except the research in [32] which used SRS documents written in German language. There has been no work done in the requirements classification written in Arabic language.

Since there is a lack of research in classifying Arabic software requirements, we have reviewed main directions in classifying Arabic texts such as sentiment analysis. The majority of the researches that classify Arabic sentences are mainly related to sentiment analysis. Arabic sentiment analysis has long been overlooked by scholars. Early Arabic sentiment research, such as those published in [83], other papers proposed and compared different techniques for Arabic sentiment as in [84]–[86].

Recently, comprehensive research presented comparative review of the various methodologies on sentiment analysis [87]. They compared a wide range of models on three benchmark datasets in their research. Their findings reveal that deep learning models paired with word embeddings outperform traditional machine learning models such as SVMs significantly. However, their results conclude that using a transformer-based language model produces superior outcomes than using a word-embedding-based deep learning model architecture. On all three benchmark datasets studied, they show that using a fine-tuned AraBERT [88] outperforms all current conventional and deep

learning models.

In [89], Mihalcea et.al. employed an English-Romanian dictionary to mechanically build a Romanian subjectivity lexicon. The vocabulary that created was then utilized to categorize Romanian text. The translation was conducted by employing two bilingual dictionaries. Another research was performed on machine-translated texts to evaluate the efficacy of using statistical sentiment analysis methodologies [90].

Another study examines the influence of translation on sentiment using source text from the Arabic language, manual and automatic translations, and both manual and automatic sentiment assignments [91]. They use Arabic social posts dataset to show that sentiment analysis of English translations of Arabic documents provides comparable findings. In addition, automatic translated English sentiment lexicons benefits Arabic sentiment analysis systems. There is no study in the literature that addresses the topic of Arabic requirements classification. Most of the studies mainly targeting the requirements classification written in English.

3.3 English Software Requirements Classification

In this section, we present the related work on automated software requirements classification. Both the traditional and deep learning techniques are discussed. We first introduce the works that are related to the binary classification task. This is followed by discussions on multiclass and multilabel classification tasks.

3.3.1 Binary Classification

Among the earliest works on automatic requirements classification were presented by Cleland-Huang et al. [5], [79]. The classification of text into functional and non-functional requirements using 'term indicators' and their probability was proposed in [5], [79]. They extracted some keywords from soft-goal interdependency graphs to be used in requirements classification. Then, the probability value, which represents keyword density, was used to measure the likelihood that the requirement belonged to one of the NFR classes. Before the requirements were classified, the sentences were preprocessed to decrease the number of keywords by stemming and stopword removal.

Hussain et al. [9] used the C4.5 decision tree learning algorithm for requirements classification based on combining keyword features such as adjective keywords and syntactic features such as the number of adjectives and adverbs using the Stanford Parser to extract features and stem the words. Support vector machines (SVMs) classifier with metadata, lexical, and syntactical features have shown higher precision and recall of up to 92.5% in classifying functional and non-functional requirements [10]. The random undersampling technique was used in binary classification to balance the two-class samples. The supervised classifier receives the sentences after they are preprocessed by removing punctuation and stopwords and lemmatizing the words. Because of the large number of features, the researchers used a scoring classifier to measure feature importance before selecting the best features.

The preprocessing impact on binary requirements classification was investigated in [81]. The researchers used the classifier in [9] after extracting the features from an

unprocessed and processed dataset. In binary classification, the authors reported an increase of approximately 5% compared to using unprocessed text. The objective of the dataset processing was to minimize the inconsistency between the requirements specifications and enrich the sentences with similar features.

The impact of using vectorization methods was investigated with different classifiers on only four categories of the PROMISE dataset [92]. Several preprocessing techniques were applied, such as lemmatization, case folding, and non-alphabetical word removal. Four classifiers and six vectorization methods were evaluated in the binary classification task. In [30], the researchers used the bidirectional encoder representations from transformers (BERT) language model to maximize the generalization. Their proposed model NoRBERT minimizes the required number of epochs to train the models. For binary classification, their results were lower than SVM, as presented in [10], but better in terms of preprocessing and feature engineering.

Several experiments were conducted in [44] using naïve Bayes (NB) and convolutional neural networks (CNNs) to classify requirements into functional and non-functional categories. Word count and TF-IDF representations were used with naïve Bayes, whereas random and pre-trained embeddings such as Word2Vec were used with CNNs. CNN with word2vec outperformed the other classifiers. The deep learning approach was utilized to classify the sentences into requirements or 'information' by CNNs [32]. They compute the difference between the two classes with respect to the known words collected from the dataset.

3.3.2 Multiclass and Multilabel Classification

The approach described in [10] for binary classification was used for multiclass classification. Four NFR classes: usability, security, operational, and performance were selected to evaluate the technique. The approach requires removing punctuation, stopwords, and word lammetization.

A rule-based approach using thematic rules was proposed to classify NFRs into sub-classes and prioritize them based on their occurrence in multiple NFR classes [93]. The 10 NFR classes in the PROMISE dataset were classified by unsupervised and supervised classifier in [81]. Binarized Naïve Bayes (BNB), which is a supervised classifier, outperformed the unsupervised classifiers. The preprocessing on the dataset doubled the result of BNB in terms of precision and recall.

For multilabel classification, Dalpiaz et al. [27] relabeled the PROMISE dataset into four categories: functional, quality, both functional and quality, or neither. Their classifier resembles the classifier in [10] and was evaluated in different experimental settings. Due to the large number of features required to classify the requirements, it is arduous for human analysts to interpret the classification results by exploring the classifier’s inner workings. This motivated the researchers to derive simple and general linguistic-based rules for feature extraction.

A novel information retrieval-based solution for classifying the requirements was proposed in [94]. The Latent Dirichlet Allocation for Requirements Classification (LDARC) approach uses an information retrieval-based solution to classify and map software requirements [94]. The four categories of NFRs which are operational, per-

formance, security, and usability were selected to evaluate several classifiers and vectorization methods in [92]. The results showed that some types of NFRs, such as 'operational' category, are more difficult than other types. The authors recommended the usage of part-of-speech tags to enhance a classifier's performance.

Recently, deep learning systems have shown promising results in requirements classification tasks. A CNN-based deep learning system for requirements classification was presented in [15]. Pre-computed word embeddings using word2vec were used along with CNNs to automatically classify 12 functional and non-functional requirements categories. Another paper discusses the Artificial Neural Network (ANN) and CNN to classify only five non-functional categories: maintainability, operability, performance, security, and usability using two well-known datasets [82]. Their results showed that CNN achieved higher precision, recall, and F1-score. NoRBERT, proposed in [30], which is based on the BERT language model, was evaluated in multiclass with four categories and all NFRs. The approach outperforms all the approaches that do not use manual preprocessing in their methodology. Using transfer learning improved the result by 15% in some tasks.

CNN and two variants of RNN—gated recurrent unit (GRU) and long short-term memory (LSTM)—were compared in classifying 10 NFR categories in [13]. The researcher used the word2vec algorithm and pre-processing steps such as lemmatization and stopword removal to improve the performance of the classifier. The LSTM outperformed CNN and GRU models. In [95], the researchers investigated the identification of all the classes in the PROMISE dataset in requirements specification documents by using the word2vec model and common keywords. Before identification, several

preprocessing steps were conducted using stopwords removal and lemmatization. To minimize human manual effort, the word2vec model was pre-trained on Wikipedia text data before it was used to compute the similarity distance between popular NFR keywords and requirements statements.

There are limited studies that tackle the issue of multilabel classification, i.e., having more than one label for a given requirement text. Dalpiaz et al. [27] adopted a quality-oriented approach in [96] to model NFRs as quality goals and relabeled the PROMISE dataset into four categories, one of which had both functional and quality annotation for the specific requirement. Another dataset called CONCORDIA RE is annotated with multiple labels based on a requirements ontology modeled using the Web Ontology Language (OWL) in [97]. They used an SVM classifier to classify functional and six NFR classes.

The PROMISE dataset was labeled to represent the multilabel requirements classification problem in [45] and several classifiers were proposed, such as k-nearest neighbor (KNN), NB, and SVMs. Their approach targets requirements identification in unstructured texts and classifies the extracted sentences into functional and NFR statements. Ramadhani et al. proposed a fuzzy similarity approach with KNN (FSKNN) with a semantic factor by measuring the correlation between the hypernyms and synonyms with NFR categories [98]. Semantic-FSKNN improves the performance in terms of accuracy and Hamming loss compared with the original FSKNN method.

3.4 Data Augmentation in Software Requirements

Several augmentation techniques have been proposed in the area of image processing to solve the problem of scarcity of images that labeled for training the machine learning models. different techniques such as rotation, mirroring, contrast adjustment can be used to generate images from the originals. Advanced image augmentation technique use Generative Adversarial Networks (GAN) to generate augmented images that improve the performance of the deep learning models [99]

Unfortunately, applying the image augmentation in text is not an option because of the semantic of the features (words). The main challenge in augmenting the text is preserving the semantic of the text while changing the words. Different text augmentation techniques have been introduced such as padding, noise injection in training RNN and CNN models [100]. Another technique used four operations: random insertion, random deletion, random replacement and random swapping to improve the classification and prevent the model from overfitting [2].

A domain-specific technique use thesaurus to increase training performance by replacing words with their synonyms [101]. These techniques can not be used to different domains that rely on different terminologies. Recently , a new augmentation technique (CUSTOM) in software requirement domain have been proposed that based on concatenate two sentence from the same category after the sentences are sorted. The technique is preserving the main keywords in the same class but the augmented sentences might lose their meaningful meaning. Most of the studies in the literature augmenting text without targeting the requirements classification written in English.

CHAPTER 4

AN END-TO-END DEEP LEARNING SYSTEM FOR REQUIREMENTS CLASSIFICATION USING RECURRENT NEURAL NETWORKS

4.1 Introduction

In this chapter, we are going to design a BiGRU-based deep learning system that uses raw word or character sequences as input. This design alleviates the need for

complex NLP-based feature engineering. BiGRU-based deep learning systems have shown promising results in other NLP and machine learning tasks but have not yet been adequately researched in relation to requirements classification. Moreover, we also present multilabel classification, in addition to both binary and multiclass classification, which has also been inadequately researched thus far. The BiGRU-based deep learning system will be designed with consideration of data scarcity issues that impact the performance of the classifier.

4.2 Research Design

In this section, we present the research questions. Next, we discuss the dataset used in the study. Finally, we discuss the research methodology used in the study. The details are as follows:

4.2.1 Motivation

Existing studies use different feature extraction and feature selection techniques to classify natural language requirements into functional and non-functional requirements. In this study, minimizing the usage of NLP processing is one of our goals in classifying the requirements. As a result, we investigate using raw texts as features for requirements classification. Raw text can be fed to the model in many ways. In the simplest form, raw text can be seen as a sequence of characters or words. Using words as tokens has a benefit over character tokens from the point of view that words have inherent meaning, whereas characters do not have such meaning. However, us-

ing the word sequence as an input requires building the input vocabulary which can lead to the Out-of-Vocabulary (OOV) problem, i.e., words in the test set that were not seen during the training. It leads to a large vocabulary size with thousands of unique words. Additionally, misspelled or unlemmatized words will not be handled appropriately.

Using character sequences as input, on the other hand, does not suffer from the OOV problem. In the case of character sequence, the vocabulary consists only of unique characters of a language and digits in addition to few special symbols (e.g., in English, there are 26 characters and 10 digits). Moreover, it may be possible for a character sequence to deal with misspelled words. For raw text to be used as input to a machine learning model, we may need a deeper system that can learn the features automatically from the raw text. BiGRU-based deep learning systems can be used in such sequence classification tasks where we can utilize the context from both the directions.

In this chapter, we also investigate the multilabel requirements classification problem. One requirement may be labeled by more than one type of non-functional or functional requirement. For example, a requirement can be separated into fine-grained sentences that have one, two, or three labels, as shown in Figure 4.1. Annotating the sentences with multiple labels is considered one of the ways to mitigate the issues associated with single labeling in requirements classification.

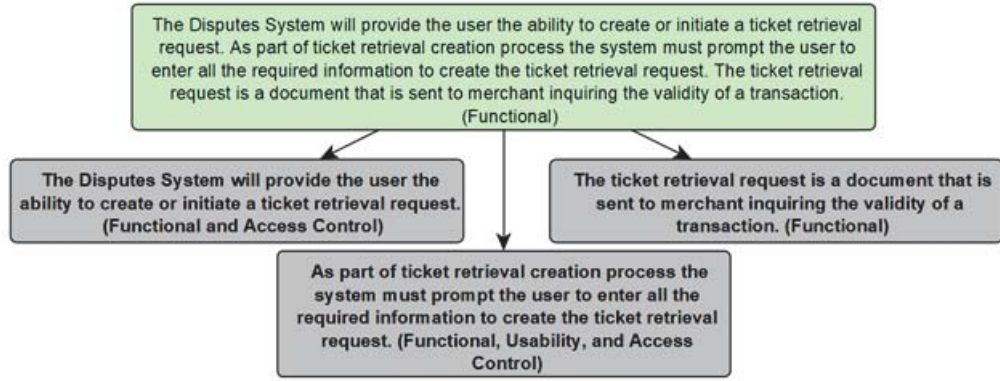


Figure 4.1: A sample requirement from the PROMISE dataset is separated into three sentences with multiple labels.

4.2.2 Research Methodology

In this subsection, we present the main steps that will be followed to achieve the study objectives. First, the characteristics of the datasets used for experiments are presented before explaining the deep learning architecture. Then, the training and evaluation approaches for all the experiments are detailed.

We performed four different tasks as listed below:

- Task 1: Binary classification of requirements into functional and non-functional.
- Task 2: Multiclass classification of requirements into different non-functional categories. We performed experiments classifying non-functional requirements into four, five, and ten classes.
- Task 3: Multilabel binary classification of requirements into functional and non-functional.
- Task 4: Multilabel multiclass classification of requirements into functional and different non-functional categories.

The original sentence is tokenized by dividing the sentence into individual words. Then the punctuation tokens, such as dots, are removed from the sentences because they do not affect the classification task. Similarly, the letter case should not affect the class of the requirements. For example, word 'the' written as 'The' in the beginning of the sentence should not influence the different requirement classes. Accordingly, case-folding is applied after punctuation removal. The ultimate goal of using case-folding is minimizing the number of unique words that need to be embedded. Since the data scarcity has affected the requirement classification, lowering the case of the letters mitigate the overhead of the trained words and minimize the number of the unknown words. Although the process of case-folding can add ambiguity in some cases but these cases can be considered as rare cases.

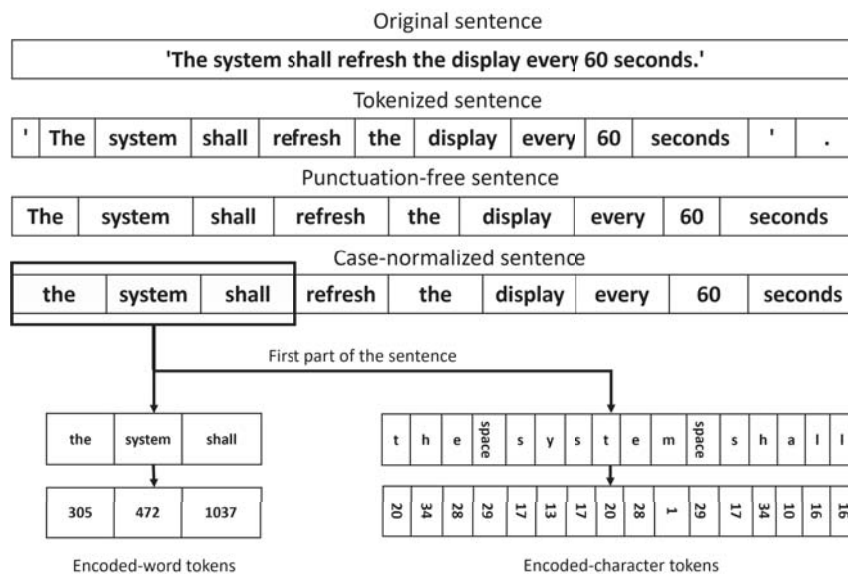


Figure 4.2: Steps of minimal pre-processing using word and character sequences.

After normalizing the letter case, dictionaries of the words and characters are built. Every word in the word sequences has a unique integer number. The size of the word

dictionary depends on the number of unique words in the corpus and the size of the character dictionary is only 38 tokens. The 38 tokens are 26 alphabetic tokens, 10 numbers from 0 to 9, a space and zero for padding. The words or characters are encoded as a sequence of integers that represent the original sentence. The preprocessing steps for words as tokens or characters as tokens are illustrated in Figure 4.2.

In contrast to CNN, which is well-known in the image processing area, RNNs are very popular for sequential tasks such as text classification, spam filtering, document-level sentiment classification, and speech tasks [102]. RNNs provides a way of treating a sequence of data by preserving their contextual information [65]. RNNs are suitable for text of variable length and is a widely used architecture for NLP tasks [66].

The vanishing gradient problem complicates RNN model learning and tuning. To solve the vanishing gradient problem, the most often used RNN variants are Long Short-Term Memory (LSTM) [103] and Gated Recurrent Units (GRU) [68]. The GRU variant has fewer parameters, is less complex, and involves fewer computations than the LSTM architecture. This is especially useful on smaller datasets, as in our case. In addition, GRUs can effectively record and reset the context information in the text, require fewer epochs and has shown to outperform LSTMs in many NLP tasks [104].

The GRU architecture introduces two gates: a reset gate r that regulates the integration of new input with the prior memory and an update gate z that controls the preservation of prior memory. The following four Eqs. 4.1, 4.2, 4.3, and 4.4 are the transition functions in GRU's hidden state h :

$$r_t = \wp(W^r x_t + V^r h_{t-1} + b^r) \quad (4.1)$$

$$z_t = \wp(W^z x_t + V^z h_{t-1} + b^z) \quad (4.2)$$

$$\hat{h}_t = \tanh(W^c x_t + V^c (r_t \odot h_{t-1})) \quad (4.3)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \quad (4.4)$$

where the parameters of the model are W , V and b , and are learnt during the training of the model at every step of t .

A bidirectional recurrent neural network (Bi-RRN) utilizes the information from the input in forward and backward directions which helps in the training stage [69]. Moreover, BiGRU is widely used in many fields such as text analysis, especially in the context of semantics [105]. The forward and backward GRU produces the token representation that is concatenated. The number of BiGRU layers and the number of neurons are tuned in all experiments using the trial and error approach.

After this, the **embedding layer** will be the next layer. The embedding represents the encoded words or characters in dense vector representation. The encoded positions of the words or characters are learned from the dataset instead of using pre-trained models such as word2vec [106] or GloVe [61]. Next, **BiGRU** receives the sequence of word or character embedding. The fully connected layer or **dense layer** is used after the BiGRU layers to add non-linearity to the model. The ReLU activation function is used in the dense layer to help the model converge within a few epochs. Then, **dropout** is used to prevent overfitting. Dropout is a common approach in the

literature where some of the units are randomly deactivated in the network during training. The **output layer** is the last layer in the proposed models which has one unit with sigmoid activation for binary classification, multiple units with sigmoid activations for multilabel classification, and multiple units with softmax activation for multiclass classification.

During training, we use the cross-entropy loss function. This means that we quantify how close the predicted distribution is to the true distribution using formula 4.5:

$$H(p, q) = \sum_x p(x) \log q(x) \quad (4.5)$$

In binary and multilabel classification, we use binary cross-entropy but in multiclass classification, we use categorical cross-entropy. The Adam optimizer is used as recommended in the literature [107]. After this, the training and evaluation will be conducted using a 10-fold cross-validation strategy. Finally, the evaluation measures will be computed based on the classification results.

4.2.3 Datasets

In this subsection, we present the datasets used in the experiments. We experimented with three publicly available datasets for software requirements classification: the PROMISE dataset, the multi-label PROMISE dataset, and the EHR dataset. Accessibility to these datasets by different researchers facilitates a degree of objectivity when comparing different techniques reported in the literature. The variations between the PROMISE and EHR datasets in terms of the number of labels, size, and

domain type enable a comprehensive evaluation and the possibility to assessing the generalization capabilities of the system. The details are as follows:

4.2.3.1 PROMISE

The PROMISE dataset is the most well-known dataset related to requirements classification in the literature [8]. It contains 625 functional and non-functional requirements. In these sentences, 255 are functional and 370 are non-functional requirements which are further sub-divided into various categories.

We perform three different classification tasks using the three datasets: binary, multiclass, or multilabel. The multilabel setup was experimented with Multilabel PROMISE and EHR datasets as discussed in the next sections. In binary classification we classify requirements into functional and non-functional. In the multiclass modality, we have three sets of NFR subcategories: four, five, and ten classes. In the literature, some studies select the most frequent classes that have the highest number of samples, either four classes [10], [30] or five classes [82]. Most studies use ten classes of NFRs after removing the portability class because it has only one sample [13], [15], [81]. Throughout this chapter, the abbreviations PROMISE NFR4, PROMISE NFR5, and PROMISE NFR10 are used to denote the number of classes of non-functional requirements. Table 4.1 illustrates the details of the modalities and classes using the PROMISE dataset. The third column of the table indicates the number of samples in a specific category. The number of non-functional requirements is greater than the number of functional requirements. The fourth and fifth columns refer to the average of the sentence length in word and character count. The 'maintainability' class has

Modality Name	Class Name	No. of samples	Avg. count of words	Avg. count of characters
Binary	Functional (F)	255	12.55	87.82
	Non-functional (NF)	370	13.45	91.89
Multiclass (Four)	Operational (O)	62	14.03	98.17
	Performance (PE)	54	14.76	96.2
	Security (SE)	66	12.67	91.83
	Usability (US)	67	14.78	100.95
Multiclass (Five)	Operational (O)	62	14.03	98.17
	Performance (PE)	54	14.76	96.2
	Security (SE)	66	12.67	91.83
	Usability (US)	67	14.78	100.95
	Others (OT)	120	13.32	92.06
Multiclass (Ten)	Availability (A)	21	13.66	82.14
	Fault Tolerance (FT)	10	12.7	92.7
	Legal (L)	13	11.92	88.3
	Look and Feel (LF)	38	12.61	88.5
	Maintainability (MN)	17	18.52	128.88
	Operational (O)	62	14.03	98.17
	Performance (PE)	54	14.76	96.2
	Scalability (SC)	21	13.28	86.05
	Security (SE)	66	12.67	91.83
	Usability (US)	67	14.78	100.95

Table 4.1: Distribution of the classes in the PROMISE dataset along with the average number of words and characters counts per sample.

the longest average words and characters while the 'legal' class has the shortest word and character counts.

4.2.3.2 Multilabel PROMISE

The PROMISE dataset was labeled in [45] with a multilabel annotation that consists of 792 sentences. The sentences are categorized into functional (F) or one of 14 non-functional requirements. The sentences are labeled with one, two, or three categories. The original sentences in PROMISE are separated into fine-grained sentences in multilabel PROMISE to reflect the exact categories. This is the reason for having a lower mean, median, maximum, and minimum length of requirements text in multilabel PROMISE compared to the original PROMISE dataset, as summarized in Table 6.3. Throughout this chapter, the abbreviations PROMISE ML is used to denote the multilabel requirements.

Dataset		Mean length	Median length	Maximum length	Minimum length
PROMISE	Word	19.89	17	84	6
	Character	117.95	103	494	40
PROMISE ML	Word	15.67	15	47	5
	Character	93.04	87	275	27
EHR	Word	21.12	18	162	3
	Character	131.38	111	943	13

Table 4.2: Descriptive statistics for the PROMISE, PROMISE ML and EHR datasets.

Table 4.3 illustrates the details of the multilabel classes where the third column indicates the number of the samples in the specific category. The fourth and fifth columns refer to the average of the sentence length in word and character representation. The functional class has the largest number of samples.

Modality Name	Class Name	No. of samples	Avg. count of words	Avg. count of characters
Multilabels (Fifteen)	Functional (F)	340	17.19	102.37
	Availability (A)	36	14.56	80.22
	Legal (L)	10	14.2	95.7
	Look and Feel (LF)	50	13.36	78.16
	Maintainability (MN)	26	14.76	91.35
	Operational (O)	89	14.25	86.90
	Performance and Capacity (PC)	75	15.54	87.53
	Reliability (RE)	12	13.17	84.17
	Security (SE)	71	14.29	88.1
	Usability (US)	101	15.62	89.78
	Recoverability (RC)	4	15.25	84.0
	Privacy (PR)	7	15.43	100.29
	Access Control (AC)	164	17.13	102.77
	Audit (AU)	20	16.05	94.6
	Others (OT)	19	16.37	98.05

Table 4.3: Distribution of the classes in the PROMISE ML dataset along with the average number of words and characters counts per sample.

The PROMISE ML sentences are labeled to have one, two, or three labels at the same time. Examples of the sentences and the number of samples in every group are shown in Table 4.4. The majority of the samples in the dataset have only one label, and only a few samples have 3 labels at the same time.

4.2.3.3 EHR

Another dataset, called electronic health records (EHR), which contains several projects from the health domain is labeled in [45] and was used for the evaluation

Number of labels	Number of samples	Example	Belong to
One	589	The system shall refresh the display every 6 seconds	Availability
		The application shall match the color of the schema set forth by Department of Homeland Security	Look and Feel
Two	174	The system shall provide a history report of changes made to the Activity or Event data	Functional and Audit
		The system shall protect private information in accordance with the organizations information policy	Privacy & Security
		The system shall be able to handle all of the user requests or usage during business hours	Performance and Availability
Three	29	Users shall be able to reach the "add" "drop and "trade player" pages with one click from the team owner's page	Functional, Usability, and Access control
		At the DePaul TeleServices Center when a supervisor uses the system to view a caller schedule the system response time must be no more than 2 seconds 90 % of the time and no more than 5 seconds the rest of the time	Functional, Availability, and Access control
		Vendor will submit a credit validation record to be applied to the enterprise billing system	Functional, Operational, and Access control

Table 4.4: PROMISE ML dataset examples

of the proposed models. In Table 6.3, the statistics of the dataset are elaborated. The EHR dataset consists of 12 projects related to the health domain. The labels of the sentences in the dataset are 14 NFR and one functional. Since our goal is to classify the requirements into functional and non-functional, we exclude the non-requirement sentences from the dataset.

In the EHR dataset, we will experiment with two modalities: binary and multi-class. In binary multilabel classification, the sentences can be labeled as functional, non-functional, or both. In multiclass multilabel classification, the sentences are categorized into functional or various NFRs. The number of non-functional requirements is roughly equal to the number of functional requirements. Table 4.5 summarizes the details about the modalities and classes, where the third column indicates the number of samples in a specific category. The fourth and fifth columns refer to the average sentence length in word and character representation, respectively.

Modality Name	Class Name	No. of samples	Avg. of word count	Avg. of character count
Multilabels (Binary)	Functional (F)	3889	20.47	130.28
	Non-functional (NF)	3015	21.24	136.64
Multilabels (Fifteen)	Functional (F)	3889	20.47	130.28
	Availability (A)	57	15.96	100.47
	Legal (L)	152	28.52	192.32
	Look and Feel (LF)	68	15.15	90.32
	Maintainability (MN)	412	21.08	148.52
	Operational (O)	207	16.61	111.91
	Performance and Capacity (PC)	100	15.8	94.64
	Reliability (RE)	41	18.76	123.02
	Security (SE)	563	19.65	133.08
	Usability (US)	148	14.35	106.97
	Recoverability (RC)	50	18.4	118.7
	Privacy (PR)	300	35.57	230.07
	Access Control (AC)	979	20.01	127.86
	Audit (AU)	276	20.3	130.01
	Others (OT)	83	22.52	151.12

Table 4.5: Distribution of the classes in the EHR dataset along with the average number of words and characters counts per sample.

4.3 Experiments, Results and Discussions

In this section, we elaborate on the metrics that are reported in the conducted experiments and their definitions.

4.3.1 Evaluation Measures

Based on the literature, we select four measures to compare the different classifiers. In the review of requirements classification [8], it was reported that 75% of primary studies used precision and recall to measure the effectiveness of the model. Since the dataset is imbalanced, we cannot rely only on accuracy. The F1-score is a harmonic mean of precision and recall and was used for evaluation. In summary, four metrics are used to report the results: accuracy, precision, recall, and F1-score. For the sake of comparison, we use the weighted average to compute precision, recall, and F1-score in multiclass classification as in [30], [97].

In multilabel classification, we compute the sample, micro, macro and weighted

average for precision, recall, and F1-score. The sample average expresses the average of specific metrics for every instance. The micro-average reports the overall true positives, false negatives, and false positives without any weight considerations for the labels. The weighted average computes the metrics with weighted consideration for every label in the dataset. The macro-average computes the metrics for every label and returns the average without considering the imbalance in samples for different classes. In addition, Hamming loss is used to compute the number of misclassified labels in multilabel classification problems [108]. The confusion matrix is presented to analyse the results and to describe the performance of the model.

4.3.2 Experiment Setup

All experiments were conducted on Google Colab using the Keras library on the TensorFlow platform. The inputs are raw data represented as sequences of words or characters with corresponding class labels. The hyper-parameters are tuned differently in the experiments to achieve higher scores in the validation data using an iterative process. Tenfold cross-validation was used as a training and validation division process. The embedding size is tuned to 500 for the word-level representation and 250 for the character-level representation. The three main layers of the model are the embedding layer, the BiGRU layers, and the fully connected layers.

The parameters of the embedding layer are the size of the vocabulary, the size of the embedding space, and the maximum sentence length. Table 4.6 summarizes the parameters in our experiments for the word and character embedding representations. The vocabulary size for the PROMISE and PROMISE ML datasets is 1672 words,

which represents the total number of unique words in the complete dataset. We have 1117 unique words for the five-class experiments, and the number increases slightly to 1318 for six-class and ten-class experiments. It should be noted here that for characters as tokens, the vocabulary size is 38, which includes the characters of the English language in addition to the digits, space, and decimal point. However, the input length to the system can differ depending on the maximum length parameter. The vocabulary size increases to 5579 unique words in EHR with a longer input length, which equals 89 words.

Dataset	Classification task	Tokens	Vocabulary Size	Vector space	Input length
PROMISE	Binary	Word	1672	500	50
	Multiclass (Four)	Word	1001	500	60
	Multiclass (Five)	Word	1318	500	63
	Multiclass (Ten)	Word	1318	500	63
	Multilabels (Fifteen)	Word	1672	500	47
	Binary	Character	38	250	200
	Multiclass (Four)	Character	38	250	200
	Multiclass (Five)	Character	38	250	300
	Multiclass (Ten)	Character	38	250	300
	Multilabels (Fifteen)	Character	38	250	150
EHR	Multilabels Binary	Word	5579	500	89
	Multilabels (Fifteen)	Word	5579	500	89
	Multilabels Binary	Character	38	250	250
	Multilabels (Fifteen)	Character	38	250	250

Table 4.6: Parameters of the embedding layer using different architectures.

One of the hyperparameters that needs to be tuned is the maximum length of the input texts to the models. The average numbers of words per sample in the original PROMISE and PROMISE ML datasets are approximately 19.9 and 15.7, and the average numbers of characters are 117.9 and 93.1, respectively. Selecting the maximum length sentence will increase the computations and raise several performance issues. We select 200 or 300 characters as the length for the PROMISE dataset and 150 characters for the PROMISE ML dataset. In most cases, the first few words of a

sentence will give enough intuition about the category. For example, the original sample the product must comply with the intranet page standards and requirements of ENET Securities. 95% of the product look and feel will be approved by the architecture group prior to implementation and the remaining 5% will be corrected and approved within one month of product release is 270 characters in length. The sample’s text was truncated by taking the first 150 characters as the product must comply with the intranet page standards and requirements of ENET Securities. 95% of the product look and feel will be approved by the architecture. The modified sample was still classified correctly as the ‘Look and Feel’ category.

We experimented with several values for the number of BiGRU layers and we tuned the value to be one BiGRU layer in the experiments using word sequences as input tokens and three stacked BiGRUs in the experiments using character sequences as input tokens. The parameters of the BiGRU layer are the number of units and the activation function for which we used ReLU. The parameters of the fully connected layer are the number of units and the activation function (ReLU) for the hidden layer, and we used the sigmoid function for the output layer if the problem was a binary classification or the softmax function if the problem was a multiclass classification.

There are other parameters that are tuned, such as learning rate decay and the number of training epochs. We set appropriate values to prevent overfitting. We used a different number of epochs based on the type of model (word or character) and the number of samples. We tuned the number of epochs to be between 15 and 60 for word models and 50 and 100 for the character models for all binary and multiclass experiments. In general, the hyperparameters of the model were chosen using an iterative

process. We tested each classifier with stratified n-fold cross-validation and computed the precision, recall, and F1-measure. With stratified 10-fold cross-validation, data are randomly partitioned into n folds based upon each fold of approximately equal size and equal response classification. The details about the replication package are publicly available in the GitHub repository.¹

4.3.3 Results

In this section, the results of both the word and character-sequence models are presented. The first two sections present binary and multiclass classification results and the next three subsections present multilabel binary and multiclass classification results.

4.3.3.1 PROMISE Binary Classification Results

The architecture of the system for binary requirements classification using word and character sequence is shown in Table 4.7. For the character model, a deeper structure is required due to the simplicity of representation and the limited size of the vocabulary compared to the word model. A large vocabulary increases the number of trainable parameters in the word model to 1,116,981 and the character model only has 393,651 trainable parameters. The model summaries for the two approaches in Figure 4.3 explain the layers of the models and their corresponding parameters.

Model	Class	Batch size	No. of epochs	RNN variants	Dense layer	Dropout	Output layer units
word	Binary	8	15	1 BiGRU 60	1 Dense 20	0.4	1
Char	Binary	8	50	3 BiGRU 50, 75, 25	1 Dense 50	0.2	1

Table 4.7: Summary of important hyperparameters for binary classification using the word and character models on the PROMISE dataset

Layer (type)	Output Shape	Param #
embedding_9 (Embedding)	(None, 50, 500)	836000
bidirectional_9 (Bidirectional)	(None, 120)	202320
dense_18 (Dense)	(None, 20)	2420
dropout_9 (Dropout)	(None, 20)	0
dense_19 (Dense)	(None, 1)	21
Total params: 1,040,761		
Trainable params: 1,040,761		
Non-trainable params: 0		

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 199, 250)	9500
bidirectional_3 (Bidirectional)	(None, 199, 100)	90600
bidirectional_4 (Bidirectional)	(None, 199, 150)	79650
bidirectional_5 (Bidirectional)	(None, 50)	26550
dense_2 (Dense)	(None, 50)	2550
dropout_1 (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 1)	51
Total params: 208,901		
Trainable params: 208,901		
Non-trainable params: 0		

Figure 4.3: Binary word (*left*) and character (*right*) model summary

Table 4.8 summarizes the accuracy, precision, recall, and F1-score, in addition to the counts of true positives, false negatives, false positives, and false negatives, for binary classification using the word and character models. In binary classification, the word model achieved a 6% higher F1-score than the character model. With the word model, only 17 and 27 sentences are misclassified out of 370 NFR and 255 FR sentences, respectively. For the character model, 40 and 45 sentences are misclassified out of 370 NFR and 255 FR sentences, respectively.

The results demonstrate that the words are more representative and easier to distinguish than the character sequences. Some keywords might be considered indicators of functional and non-functional requirements. The character model needs deeper representations which in turn requires more training data to learn the relationships robustly.

Model Name	True positive	True negative	False positive	False negative	Accuracy	Precision	Recall	F-score
Word sequence	353	228	17	27	0.93	0.93	0.95	0.94
Character sequence	330	210	40	45	0.85	0.86	0.91	0.88

Table 4.8: Binary classification results for the word and character models on the PROMISE dataset.

4.3.3.2 PROMISE Multiclass Classification Results

In this subsection, we describe multiclass classification by experimenting with four, five, and ten categories using word and character-sequence models. Four, five, and ten categories of NFR word models are shown in Table 4.9. The table contains the details about the architectures of the proposed models such as embedding size, RNN layers with their units, dense layer, and dropout values.

Model	Class	Batch size	No. of epochs	RNN layers	Dense layer	Dropout	Output layer units
word	4 classes	8	20	1 BiGRU 80	1 Dense 60	0.4	4
character	4 classes	8	70	3 BiGRU 100, 150, 75	1 Dense 120	0.2	4
word	5 classes	8	50	1 BiGRU 100	1 Dense 70	0.2	5
character	5 classes	8	80	3 BiGRU 125, 150, 100	1 Dense 100	0.2	5
word	10 classes	8	50	1 BiGRU 100	1 Dense 70	0.5	10
character	10 classes	8	80	3 BiGRU 125, 150, 100	1 Dense 100	0.2	10

Table 4.9: Summary of important hyperparameters for multiclass classification using word and character models on the PROMISE dataset.

In the multiclass classification problem, the word-sequence model show stable results compared to the character model whose F1-score drops from 0.94 in binary classification to 0.75 in ten-class classification. The results of four, five, and ten categories of the word and character models are shown in Table 4.10. The confusion matrices of the word and character sequences classification for four and five classes are shown in Figures 4.4 and 4.5, respectively. The confusion matrices in Figure 4.6 show that some misclassified sentences are frequent between pairs of classes, such as 'security' and 'usability'. This further strengthens the arguments for multilabel classification.

Model Name	Accuracy	Precision	Recall	F-score
PROMISE NFR4 categories classification				
Word sequence	0.87	0.89	0.87	0.87
Character sequence	0.70	0.73	0.70	0.70
PROMISE NFR5 categories classification				
Word sequence	0.77	0.79	0.77	0.77
Character sequence	0.63	0.66	0.63	0.63
PROMISE NFR10 categories classification				
Word sequence	0.76	0.78	0.76	0.75
Character sequence	0.56	0.56	0.56	0.54

Table 4.10: Multiclass classification results for the word and character models on the PROMISE dataset.

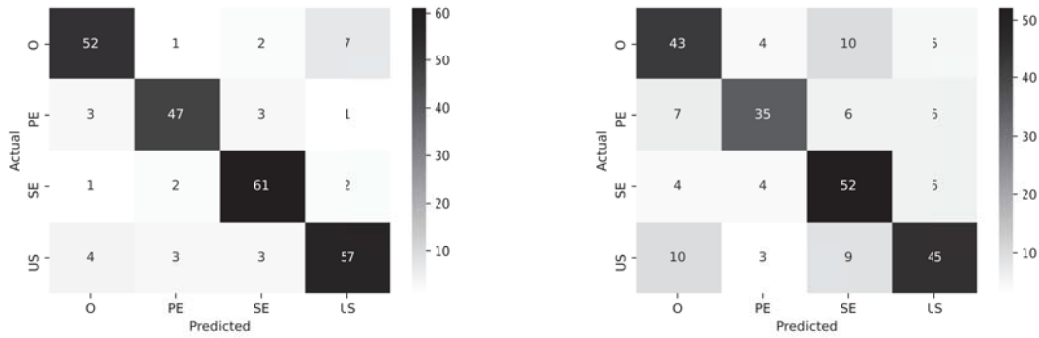


Figure 4.4: PROMISE NFR4 confusion matrices for word and character models

4.3.3.3 PROMISE Multilabel Multiclass Classification Results

The word and character-sequence models are used for the multilabel classification problem with stratified sampling in this subsection. The output layer uses the sigmoid function to determine membership for each class. Table 4.11 details the hyperparameters for the word and character models for multilabel classification.

Model	Class	Batch size	No. of epochs	RNN layers	Dense layer	Dropout	Output layer units
word	15 LABELS	8	60	1 BiGRU 256	1 Dense 125	0.5	15
character	15 LABELS	8	100	3 BiGRU 125, 150, 100	1 Dense 125	0.2	15

Table 4.11: Summary of the important hyperparameters for multiclass multilabel classification using the word and character models on the PROMISE ML dataset.

In multilabel classification, the difference in accuracy between the word-sequence model and character-sequence model is 0.19. The weighted average of the F-score is

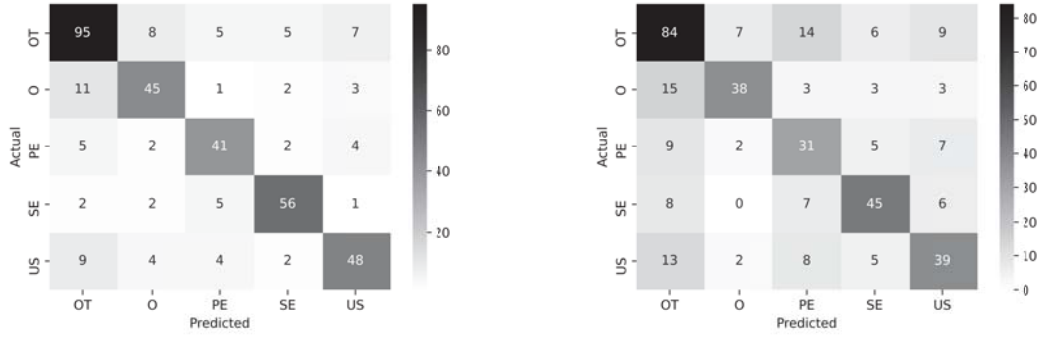


Figure 4.5: PROMISE NFR5 confusion matrices for word and character models

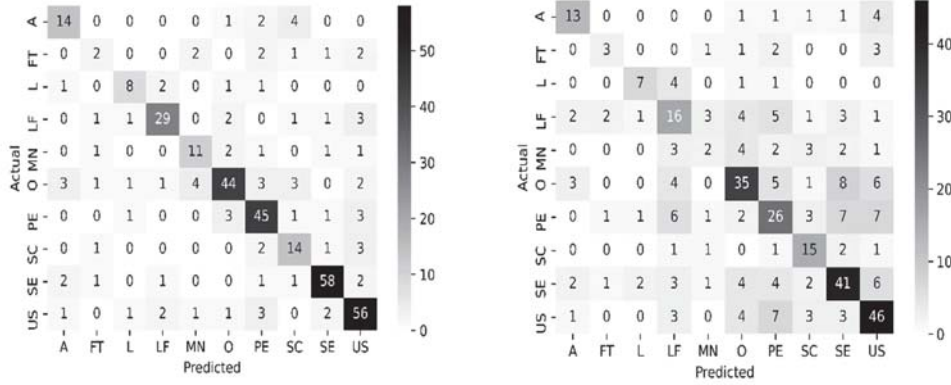


Figure 4.6: PROMISE NFR10 confusion matrices for word and character models

0.69 for the word-sequence model and 0.52 for the character-sequence model. The limited number of samples in most classes and the increasing number of categories are the main reasons for the misclassified sentences. The results of the multilabel word and character models are shown in Table 4.12.

Model Name	Accuracy	Precision	Recall	F-score	Hamming loss
Samples					
Word sequence	0.57	0.66	0.67	0.66	0.05
Character sequence	0.38	0.50	0.49	0.48	0.07
Weighted					
Word sequence	0.57	0.73	0.67	0.69	0.05
Character sequence	0.38	0.59	0.51	0.52	0.07
Macro					
Word sequence	0.57	0.49	0.42	0.44	0.05
Character sequence	0.38	0.39	0.28	0.31	0.07

Table 4.12: Multiclass multilabel classification results using the word and character models on the PROMISE ML dataset.

4.3.3.4 EHR Multilabel Binary Classification Results

The EHR dataset is used to evaluate the proposed word and sequence models. We evaluated the proposed models in binary classifications. The hyper-parameters for the models are tuned as shown in Table 4.13.

Model	Class	Batch size	No. of epochs	RNN layers	Dense layer	Dropout	Output layer units
word	2 LABELS	64	60	1 BiGRU 256	1 Dense 125	0.5	2
character	2 LABELS	64	100	3 BiGRU 150, 200, 125	1 Dense 125	0.2	2

Table 4.13: Summary of the important hyperparameters for binary multilabel classification using the word and character models on the EHR dataset.

The classification results are presented in Table 4.14. For binary classification, a higher accuracy of up to 0.8 is achieved by the word-based model and 0.74 by the character-sequence model. The difference in Hamming loss between the word and character models is limited to approximately 0.04 and only 0.03 in the F1-measures. Thus we can clearly see by comparing the results from the previous section that increasing the number of samples can significantly improves the results of the character-sequence model.

Model Name	Accuracy	Precision	Recall	F-score	Hamming loss
Samples					
Word sequence	0.80	0.90	0.91	0.89	0.17
Character sequence	0.74	0.87	0.88	0.86	0.17
Weighted					
Word sequence	0.80	0.89	0.89	0.89	0.13
Character sequence	0.74	0.85	0.87	0.86	0.17
Macro					
Word sequence	0.80	0.88	0.89	0.89	0.13
Character sequence	0.74	0.85	0.87	0.85	0.17
Micro					
Word sequence	0.80	0.88	0.89	0.89	0.13
Character sequence	0.74	0.85	0.87	0.86	0.17

Table 4.14: Binary multilabel classification results using the word and character models on the EHR dataset.

4.3.3.5 EHR Multilabel Multiclass Classification Results

In multiclass classification, the models are evaluated to classify the sentences with multilabels. The hyper-parameters of the models are similar to the models in binary classification except for the units of the output layer, as shown in Table 4.15.

Model	Class	Batch size	No. of epochs	RNN layers	Dense layer	Dropout	Output layer units
word	15 LABELS	64	60	1 BiGRU 256	1 Dense 125	0.5	15
character	15 LABELS	64	100	3 BiGRU 150, 200, 125	1 Dense 125	0.2	15

Table 4.15: Summary of the important hyperparameters for multiclass multilabel classification using the word and character models on the EHR dataset.

The models achieved lower accuracy in multiclass classification than in binary classification. The difference between the word and character models is 0.06 in terms of the weighted F1-measure. The detailed results of both the models are shown in Table 4.16.

Model Name	Accuracy	Precision	Recall	F-score	Hamming loss
Samples					
Word sequence	0.69	0.83	0.83	0.81	0.03
Character sequence	0.64	0.76	0.74	0.74	0.04
Weighted					
Word sequence	0.69	0.82	0.80	0.81	0.03
Character sequence	0.64	0.79	0.73	0.75	0.04
Macro					
Word sequence	0.69	0.68	0.56	0.60	0.03
Character sequence	0.64	0.61	0.42	0.48	0.04
Micro					
Word sequence	0.69	0.83	0.80	0.81	0.03
Character sequence	0.64	0.81	0.73	0.77	0.04

Table 4.16: Multiclass multilabel classification results using the word and character models on the EHR dataset.

4.3.4 Discussion

Based on the results in subsection 4.3, the word-based are more robust and stable. The results show that there are fewer misclassified NFR samples than misclassified functional samples in both the word and character-sequence models. We note that

some of the misclassified functional requirement samples contain numbers that might mislead the models because the numbers are mainly used in the NFR for specifying measurements and hard constraints. As an example, the sentence *Movies can only be streamed if the customer has purchased a movie and is within the **2-day time** period to stream the movie* is misclassified as an NFR.

As shown in Figures 4.4 and 4.5, the word-sequence models are capable of detecting the samples of two classes 'performance' (PE) and 'security' (SE) with a low number of false negatives compared with other classes in four and five-class classification problems. In the four-class classification problem, there are frequent misclassified samples between two classes 'usability' and 'operational' in both word and character-sequence models. In the five-class classification problem, the class 'others' (OT), which contains several characteristics of six categories, results in several errors for both the word and character models.

We note that one of the reasons for misclassification is the lack of clarity in the sentences. Some sentences are very short and abstract, which complicates the classification tasks. As an example, the sentence *"The product shall be used in office environments"* is related to the 'operational' subcategory. Another source of misclassification is the overlap between the categories. The intersection between the categories is clearly due to some common words between two classes that can be misleading during classification. An example of a misclassified sample that belongs to overlapping categories is the statement *Users shall **feel** satisfied using the product. 85% of all users will be satisfied with the product*, which belongs to the 'usability' class even though the first part of the statement gives some intuition about the 'look and feel'

class.

Identifying the relationship between 'usability' (US) and 'Look and Feel' (LF) is a challenging task because of the high similarities between them. Another example is that the 'operational' (O) and 'Look and Feel' (LF) classes have some common characteristics that make it complicated to differentiate between them. But in general, the proposed models are capable of distinguishing between the classes and are capable of classifying the samples correctly most of the time.

Based on the confusion matrix in Figure 4.6, the 'availability' and 'security' classes, that can belong to a single class at a higher level, result in a high number of misclassified samples. Four sentences of the 'availability' class are misclassified as security. In the character models, the 'usability' and 'performance' classes contain 14 misclassified sentences, where 7 sentences are misclassified as 'usability' instead of 'performance' and 7 are misclassified as 'performance' instead of 'usability'.

A limited number of samples in one category hinders the model from learning to make a correct prediction. One example is the 'fault tolerance' class that contains the minimum number of samples in the dataset. In both the word and character models, only a limited number of 'fault tolerance' samples are correctly classified, two samples are correctly classified in the word model and three samples are correctly classified in the character models. The same issue appears with the two classes 'legal' and 'availability'.

Some classes have shown significant similarity in their classification results in both the word and character models. The difference between the number of correctly classified samples using the word and character models is one for 'availability', 'legal'

and 'scalability'. The number of correctly classified requirements of the three previous classes are 14, 8, and 14 while the character model classified 13, 7, and 15 samples correctly. The difference is approximately ten samples in the 'operational' and 'usability' classes.

Using deep learning for multilabel classification is proven to outperform the other techniques. In addition, BiGRU models for word and character sequences are a more general and usable approach. The number of labels is specified as the number of nodes in the output layer that contain the sigmoid function to predict the probability of class membership for the label, a value between 0 and 1 before the value is rounded.

The word-sequence models outperform the character-sequence models in all the experiments. Both word and character models are affected negatively by increasing the number of classes, but the word model tends to be more robust to the increasing number of classes than the character model. The character models are promising, and we can ascertain which one is the best because of the tradeoffs between them. The character models are robust to misspelled words where one character is affected, such as Avialable, and the sub-word is utilized, but with the word model, the misspelled words are treated differently in the vocabulary.

The differences between the performance of word and character sequences models on ten folds are highlighted in Figure 4.7. The standard deviation of the binary word-sequence model is ± 2.2 , and standard deviation of the binary character-sequence model is ± 4.1 . The difference in standard deviation is not significant between word and character sequences models in the four and five-class classification tasks. Clearly, the standard deviation of character-sequence models are higher than the word-sequence

models.

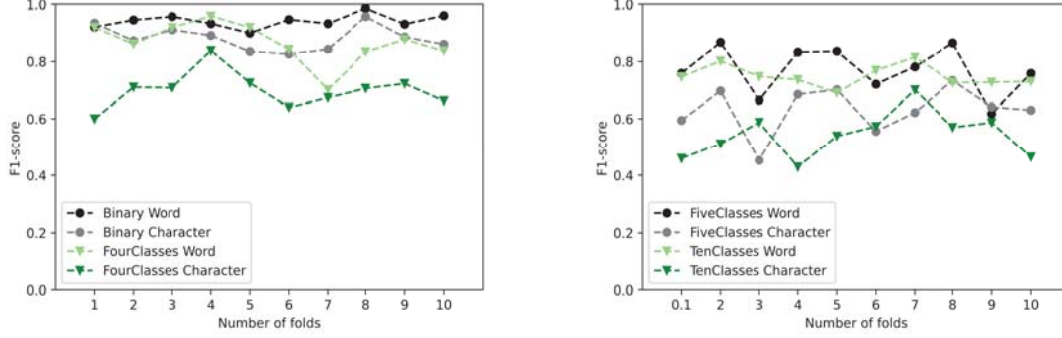
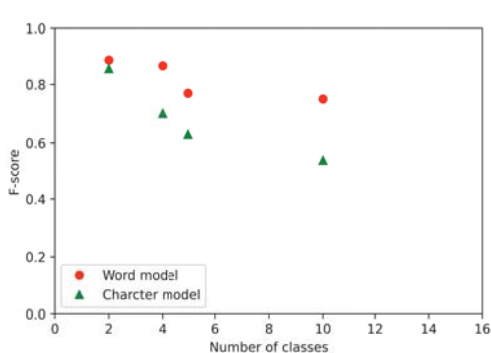


Figure 4.7: Performance of the word and character models on the ten folds of the PROMISE dataset.

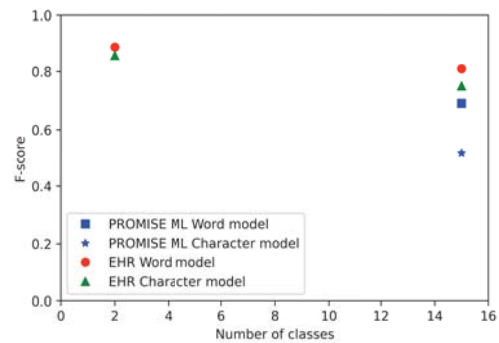
Another advantage of using character models is that similar words such as Available and Availability have limited impact compared to the word models that consider them as two different words. Character models are more useful if we have constraints on memory and computation. The F-score for both the word and character models with single label classification and multilabel classification are shown in Figure 6.9(a) where the red circles are the word models and the green triangles are the character models. In Figure 6.9(b), the blue squares represent the word model on the PROMISE ML and the stars represent the character model on the PROMISE dataset. The red circles and green triangles represent the word and character models on the EHR dataset.

4.4 Comparison with Published Works

The literature was surveyed to select relevant studies to compare with our results. The criteria for selecting the studies are that the study should discuss functional and non-functional classification and the used datasets must be part of the work. The



((a)) PROMISE single-label classification



((b)) PROMISE ML and EHR multilabel classification

Figure 4.8: F1-score of the word and character models on single-label and multilabel classifications.

reported metrics are accuracy, precision, recall, and F-score. The macro average of the multiclass is used in this comparison.

4.4.1 Binary Classification

In binary classification, several studies used either traditional machine learning or deep learning. Different traditional supervised learning methods rely on different types of features, as in [9], [10], [81], [92]. The models proposed to solve binary classification achieve good performance either by using syntactic features and keywords as in [9], meta-data, lexical, and syntactical features as in [10], or vectorization methods as in [92]. Another study that is based on the CNN model with fastText embedding scored 0.93 for accuracy and 0.94 for F-score as in [109] or with Word2vec as in [44]. In addition to feature engineering, preprocessing has been implemented in previous papers such as stopword removal as in [10], lemmatization as in [10], [92] and different temporal and grammatical preprocessing as in [81]. One paper that used BERT to

increase generalization achieved 0.92 for precision, recall and F-score [30]. Our models outperform the previous models except for the research that relies on syntactic features and keywords [9]. A comparison between our results and the results reported in the literature on binary classification is shown in Table 6.7.

Reference	No. of Classes	Classifier type	Names of classes ¹	Classifier ²	Features	Precision	Recall	F1-score
[9]	Binary	Traditional Machine learning	F, NF	Decision tree C4.5	Syntactic Features and keywords	0.97	1.00	0.98
[10]			F, NF	SVM	meta-data, lexical, and syntactical	0.92	0.93	0.93
[92]			F, NF	SVM, NB, LR, RF	Six different vectors	-	-	≈ 0.8
[81]			F, NF	Decision tree C4.5	Ten features from un-processed dataset	0.90	0.90	0.90
[81]			F, NF	Decision tree C4.5	Ten features from processed dataset	0.95	0.94	0.94
[109]		Deep Learning	F, NF	CNN	fastText	0.92	0.96	0.94
[44]			F, NF	CNN	Word2vec	0.92	0.91	0.92
[30]			F,NF	BERT	-	0.92	0.92	0.92
Our word model			F, NF	BiGRU	-	0.93	0.95	0.94

1) Functional (F), non-functional (NF)

2) SVM: Support Vector Machine, NB: Naïve Bayes, LR: Logistic Regression,RF: random forest,CNN: convolutional neural network BERT: Bidirectional Encoder Representations from Transformers, and BiGRU: Bidirectional Gated Recurrent Unit

Table 4.17: Comparison with published works on binary requirements classification tasks.

4.4.2 Multiclass Classification

In multiclass classification, the first two papers [79], [95] that used TF-IDF keywords and indicator terms for classification achieve a good F-score between 0.36 and 0.42. The studies in [10], [27], [92] use an SVM for the different modalities of multiclass

classification, achieving an F-score in the range between 0.68 and 0.79. Using binarized Naïve Bayes (BNB) with processed and unprocessed datasets doubled precision and recall to 90% and 91% respectively, as in [81]. For multiclass classification, the reported papers used preprocessing methods such as stopwords removal [10], stemming [97] or lemmatization [10], [92].

Applying the deep learning techniques ANN [82], CNN [13], [15], [82], [109], or RNN [13] with the word2vec pre-trained models or bag of words achieved an F-score ranging between 0.77 and 0.85. For the reported papers, preprocessing methods have been used on the dataset before applying DL models such as stopwords removal [13], [82], [109], stemming [82], [109], or lemmatization [13]. Overall, the performance of BiGRU with raw words showed promising results by capturing several characteristics of functional and non-functional subcategories. A comparison between our results and the results reported in the literature on multiclass classification is shown in Table 6.10.

4.4.3 Multilabel Classification

There are a limited number of research studies in the literature which investigate multilabel requirements classification. SVM was evaluated on the PROMISE ML dataset, and the macro-F-score was reported as 0.382 in [45]. The same classifier achieved a macro-F-score of 0.62 on the EHR dataset. The SVM classifier in [97] was evaluated on the CONCORDIA dataset and achieved a weighted-F-score of 0.67 for classifying the sentences into 7 classes. Our word model achieved a 0.44 macro-F-score on PROMISE ML and 0.6 on the EHR dataset. In binary classification, the word model achieved an F-score of 0.89, and the character model achieved an F-score

Reference	No. of Classes	Classifier type	Names of classes ³	Classifier ⁴	Features	Precision	Recall	F1- score	
[79]	Multiclass	Rule-based	A, L, LF, MN, O, PE, SC, SE, US	-	Indicator terms	0.24	0.76	0.36	
[95]			F, A, FT, L, LF, MN, O, PE, PO, SC, SE, US	-	Word2vec and common keywords	0.51	0.41	0.42	
[81]		Traditional Machine learning	A, FT, L, LF, MN, O, PE, SC, SE, US	BNB	Ten features from unprocessed dataset	0.45	0.45	0.45	
[81]			A, FT, L, LF, MN, O, PE, SC, SE, US	BNB	Ten features from processed dataset	0.90	0.91	0.9	
[97]			A, L, LF, MN, O, PE, SC, SE, US	SVM	Unigram tokens with ontology	0.77	0.6	0.67	
[10]			O, PE, SE, US	SVM	meta-data, lexical, and syntactical	0.78	0.81	0.77	
[27]			F, NF, Only F, Only NF	SVM	100 best features	0.81	0.78	0.79	
[27]			F, NF, Only F, Only NF	SVM	500 best features	0.77	0.75	0.75	
[92]			O, PE, SE, US	SVM, NB, LR, RF	Six different vectors	≈ 0.77	≈ 0.67	≈ 0.68	
[82]			Deep Learning	MN, O, PE, SE, US	ANN, CNN	bag of words	0.86	0.83	0.85
[15]				F, A, FT, L, LF, MN, O, PE, PO, SC, SE, US	CNN	word2vec using skip-gram model	0.81	0.785	0.77
[13]				A, FT, L, LF, MN, O, PE, SC, SE, US	CNN, RNN variants	Word2vec	0.72	0.72	0.7
[109]		A, L, LF, MN, O, PE, SC, SE, US		CNN	Word2vec	0.79	0.75	0.76	
[30]		O, PE, SE, US		BERT	-	0.86	0.88	0.87	
[30]		A, FT, L, LF, MN, O, PE, SC, SE, US		BERT	-	0.79	-	0.82	
Our word model		LF, O, PE, SE, US		BiGRU	-	0.89	0.87	0.87	
Our word model		LF, O, PE, SE, US, OT		BiGRU	-	0.79	0.77	0.77	
Our word model		A, FT, L, LF, MN, O, PE, SC, SE, US		BiGRU	-	0.78	0.76	0.75	
3) Functional (F), non-functional (NF), availability (A), legal (L), look and feel (LF), maintainability (MN), operational (O), performance (PE), scalability (SC), security (SE), usability (US), fault tolerance (FT), and Others (OT).									
4) BNB: Binarized Naïve Bayes, SVM: Support Vector Machine, NB: Naïve Bayes, LR: Logistic Regression, RF: random forest, ANN: Artificial Neural Network, CNN: convolutional neural network, RNN: recurrent neural network, BERT: Bidirectional Encoder Representations from Transformers, and BiGRU: Bidirectional Gated Recurrent Unit.									

Table 4.18: Comparison with published works on multiclass requirements classification tasks

of 0.85. A comparison between our results and the results reported in the literature on multilabel multiclass classification is shown in Table 4.19.

Obviously, the capability of GRU to capture the long-term dependencies improves the performance of the proposed models in word and character sequences but only in a forward sequence. The bidirectional version of GRU can capture more information about the context and improve the performance of the classifier by using two independent hidden layers to combine the forward and backward processing of data. These characteristics help the proposed BiGRU models learn the required dependencies and

Reference	No. of Classes	Classifier type	Dataset	Names of classes ⁵	Classifier ⁶	Features	Precision	Recall	F1- score
[45]	Multilabel	Traditional Machine Learning	PROMISE	F, A, L, LF, MN, O, PC, RE, SE, US, RC, PR, AC, AU, OT	SVM	Similarity distance	-	-	0.382
[45]			EHR	AC, AU, AV, LG, LF, MT, OP, PR, PS, RC, RL, SC, US, OT, FN, NA	SVM	TF-IDF Keywords	0.72	0.54	0.62
[97]			CONCORDIA	F, CO, SE, US, EF, FN, RE	SVM	Unigram tokens with ontology	0.77	0.6	0.67
Our word model		Deep Learning	PROMISE	F, A, L, LF, MN, O, PC, RE, SE, US, RC, PR, AC, AU, OT	BiGRU	-	0.49	0.42	0.44
Our word model			EHR	F, NF	BiGRU	-	0.88	0.89	0.89
Our word model			EHR	F, A, L, LF, MN, O, PC, RE, SE, US, RC, PR, AC, AU, OT	BiGRU	-	0.68	0.56	0.60
5) Functional (F), availability (A), legal (L), look and feel (LF), maintainability (MN), operational (O), performance and capacity (PC), reliability (RE), security (SE), usability (US), recoverability (RC), privacy (PR), access control (AC), audit (AU), and Others (OT)									
6) SVM: Support Vector Machine, and BiGRU: Bidirectional Gated Recurrent Unit.									

Table 4.19: Comparison with published works on multilabel requirements classification tasks.

relationships effectively.

4.5 Threats to Validity

Since our research is vulnerable to several threats to validity, we use different mitigation procedures. In this section, the threats and mitigation are elaborated in detail.

Construct Validity: The lack of reporting and evaluation standards in the research studies in the literature complicates the analysis and performance comparison. To mitigate this, we conducted several experiments that contain a variable number of categories to compare their work. Additionally, we used four metrics that provide in-

sight into classifier performance. To facilitate research reproducibility, we fix the seed of the random number generators to be equal to 42, as in [27], in all the conducted experiments.

External Validity: The PROMISE projects written by students are not representative of all projects especially projects from the industry. We used the PROMISE dataset because it is the most frequently used dataset in the literature and, thus, enables comparison with other approaches. In addition, the PROMISE ML dataset is labeled by one author, and 30 out of 792 requirements sentences were validated by other people, and the inter-rater agreement was 0.71. An inter-rater value of 0.71 indicates moderate agreement. We also used the EHR dataset, which contains several SRSs written by experts from the industry. To prevent overfitting, we used the minimum number of epochs, neurons, and layers and utilized learning rate decay and model checkpoints.

Conclusion Validity: The imbalanced data set might have some impact on our findings, especially in the NFR subcategories which suffer from variation in the number of samples. Using under-sampling is not a useful approach due to the limited size of the dataset. Conducting several experiments with different NFR subcategories mitigates the issues of the existence of a small number of samples of a specific subcategory. Moreover, we used stratified 10-fold cross-validation, which can maintain the same number of samples of all categories in one iteration.

4.6 Conclusions and Future Work

In this study, we presented an end-to-end approach to classify software requirements using raw word and character sequences employing BiGRUs. The presented systems achieved state-of-the-art results on publicly available datasets. The models are evaluated on binary and multiclass classifications on the original PROMISE dataset and multilabel multiclass classification on the PROMISE ML dataset. In addition, the EHR dataset is used to evaluate the proposed model on multilabel binary and multilabel multiclass classification tasks.

Two approaches were investigated in terms of input text tokenization: word sequences as tokens and character sequences as tokens. In general, the word sequence outperforms the character models in binary and multiclass classification. The performance of the character models is affected when the number of classes increases while the performance of the word models maintains its classification ability. Additionally, the models based on BiGRU outperform most of the other deep learning or traditional machine learning technique.

CHAPTER 5

AUGMENTATION-BASED TECHNIQUES FOR SOFTWARE REQUIREMENTS CLASSIFICATION

5.1 Introduction

In the previous chapter, we improve the classifier's performance by designing the BiGRU-based deep learning system considering data scarcity issues. However, a limited dataset size or an imbalanced dataset can affect the classifier's performance. Deep learning techniques can process large datasets effectively; however, all publicly available datasets in the software requirements domain are considered small. Annotating SRS documents from different disciplines by domain experts will advance the research

in the requirement classification field. But the main challenge is that labeling the dataset manually is time and effort-consuming.

Usually, requirements are collected from actual SRS documents, resulting in imbalanced datasets having highly skewed distribution of samples to different requirement classes. Since a given requirement can belong to multiple classes at the same time, augmenting multilabel requirements requires special attention to mitigate the impact of imbalanced datasets. Augmenting the imbalanced classes equally increase the gap between the majority and the minority classes. The imbalanced issue is much harder in multilabel classes due to the intersection between the majority and the minority classes. In this chapter, we propose different data augmentation techniques for dealing with the issues of limited and imbalanced training data.

5.2 Research Design

The research design to achieve this objective is divided into five subsections. First, we discuss the research methodology and the datasets used in the study. Then, we elaborate architecture of augmentation-based technique and classifier architecture used in the study. Finally, the classification measures are defined. The following subsections describe a summary of them.

5.2.1 Research Methodology

The research methodology of our third objective can be classified into three major parts. The first part is splitting the dataset into training and validation splits to be

used in the experiments. The split should be consistent with previous experiments to have a systematic comparison. The split of evaluation will be used to evaluate the proposed model and will not be part of any other parts. The second part is augmenting the training split using the augmentation technique. Several techniques can be used in the augmentation stage to improve the classification results. The validation split is used to evaluate the classifier trained on augmented and original training split. The augmentation process has been conducted on ten training splits using the well-known method of ten-folds cross-validation. The third part is building the deep learning classifier to be trained on the augmented dataset. After that, the classifier's performance is evaluated and reported. The main components of the research methodology of classifying requirements using augmentation-based techniques are shown in Figure 5.1.

5.2.2 Existing Datasets

In the reported experiments, two datasets with one and multilabel have been used and presented in the following subsection.

5.2.2.1 PROMISE Dataset

The dataset consists of 15 projects spanning a wide range of domains and was written by students as a term project [5]. The details about the PROMISE dataset have been introduced in section 4.2.3.1. Table 5.1 illustrates the details of the modalities and classes using the PROMISE dataset along with the average number of words counts per sample.

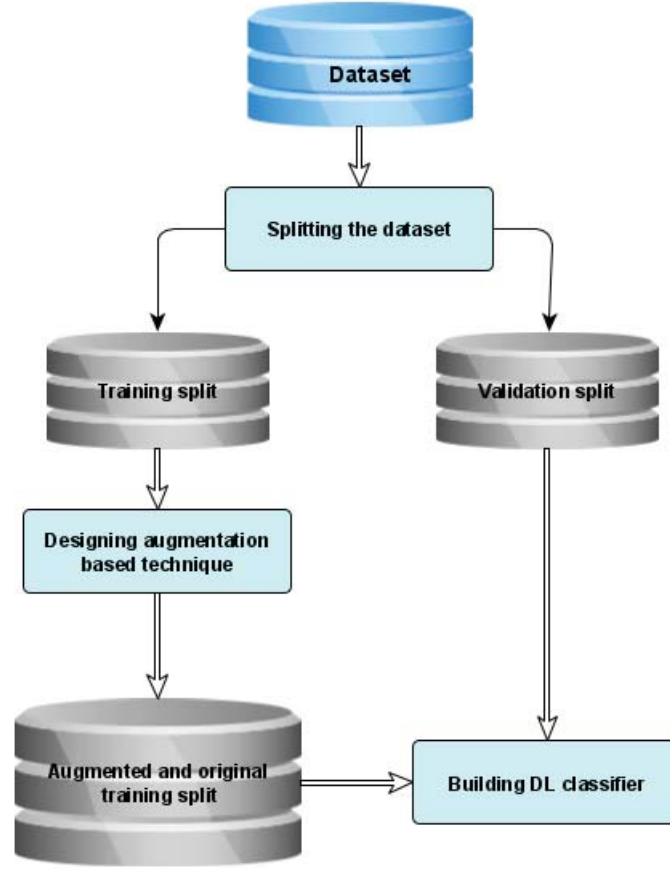


Figure 5.1: Overview of the research methodology in classifying software requirements using augmentation-based technique.

5.2.2.2 Multi-label EHR Dataset

The EHR dataset consists of 12 projects related to the health domain. The labels of the sentences in the dataset are 14 NFRs. Since our goal is to classify the requirements into nonfunctional categories, we exclude the functional and nonfunctional requirement sentences from the dataset. In the EHR dataset, the multiclass multilabel classification has been addressed, and the sentences are categorized into various NFRs. On average, the longest sentences belong to the privacy class, and the shortest sentences belong to the usability class. The details about the PROMISE dataset have been introduced

Modality Name	Class Name	No. of samples	Avg. count of words
Binary	Functional (F)	255	12.55
	Non-functional (NF)	370	13.45
Multiclass (Four)	Operational (O)	62	14.03
	Performance (PE)	54	14.76
	Security (SE)	66	12.67
	Usability (US)	67	14.78
Multiclass (Ten)	Availability (A)	21	13.66
	Fault Tolerance (FT)	10	12.7
	Legal (L)	13	11.92
	Look and Feel (LF)	38	12.61
	Maintainability (MN)	17	18.52
	Operational (O)	62	14.03
	Performance (PE)	54	14.76
	Scalability (SC)	21	13.28
	Security (SE)	66	12.67
	Usability (US)	67	14.78

Table 5.1: Distribution of the classes in the PROMISE dataset along with the average number of words counts per sample.

in section 4.2.3.2. Table 5.2 summarizes the details about the classes, where the third column indicates the number of samples in a specific category. The fourth column refers to the average sentence length in word representation.

Modality Name	Class Name	No. of samples	Avg. of word count
Multilabels (Fourteen)	Availability (A)	57	15.96
	Legal (L)	152	28.52
	Look and Feel (LF)	68	15.15
	Maintainability (MN)	412	21.08
	Operational (O)	207	16.61
	Performance and Capacity (PC)	100	15.8
	Reliability (RE)	41	18.76
	Security (SE)	563	19.65
	Usability (US)	148	14.35
	Recoverability (RC)	50	18.4
	Privacy (PR)	300	35.57
	Access Control (AC)	979	20.01
	Audit (AU)	276	20.3
	Others (OT)	83	22.52

Table 5.2: Distribution of the classes in the EHR dataset along with the average number of words counts per sample.

5.2.3 Architecture of Augmentation-based Software Requirement Classifier

In this subsection, we will elaborate on the architecture of the common data augmentation technique before discussing the shortcomings of the common data augmentation technique. Then, the proposed architecture that is supposed to tackle the shortcomings mentioned above is presented with the architecture of the DL-classifier.

5.2.3.1 Architecture of common data augmentation technique

In this subsection, the architecture of one of the most cited data augmentation technique in recent years is elaborated [2]. This common data augmentation technique has been applied generally in text classification used four operations: random insertion, random deletion, random replacement, and random swapping to improve the classification and prevent the model from overfitting. Before that, the user should set two parameters: the percent of change and the number of augmented sentences per sample. Some of these operations used WordNet to generate the synonyms. The WordNet is a semantic ontology that enumerates the relations between words and other elementary semantic units. It builds a unique semantic network that helps find word relations, synonyms, grammar, etc. WordNet [110] puts nouns into groups of cognitive synonyms called synsets. For instance, the words car and automobile belong to the same synset. WordNet constitutes a hierarchical structure by applying a super-subordinate relationship between synsets. The synset for chair, for example, is a child of the synset for furniture in the WordNet hierarchy. WordNet supports

Natural Language Processing (NLP) tasks such as sentiment analysis, text similarity, automatic language translation, and more. An example of the WordNet network of the word book is shown in Figure 5.2.

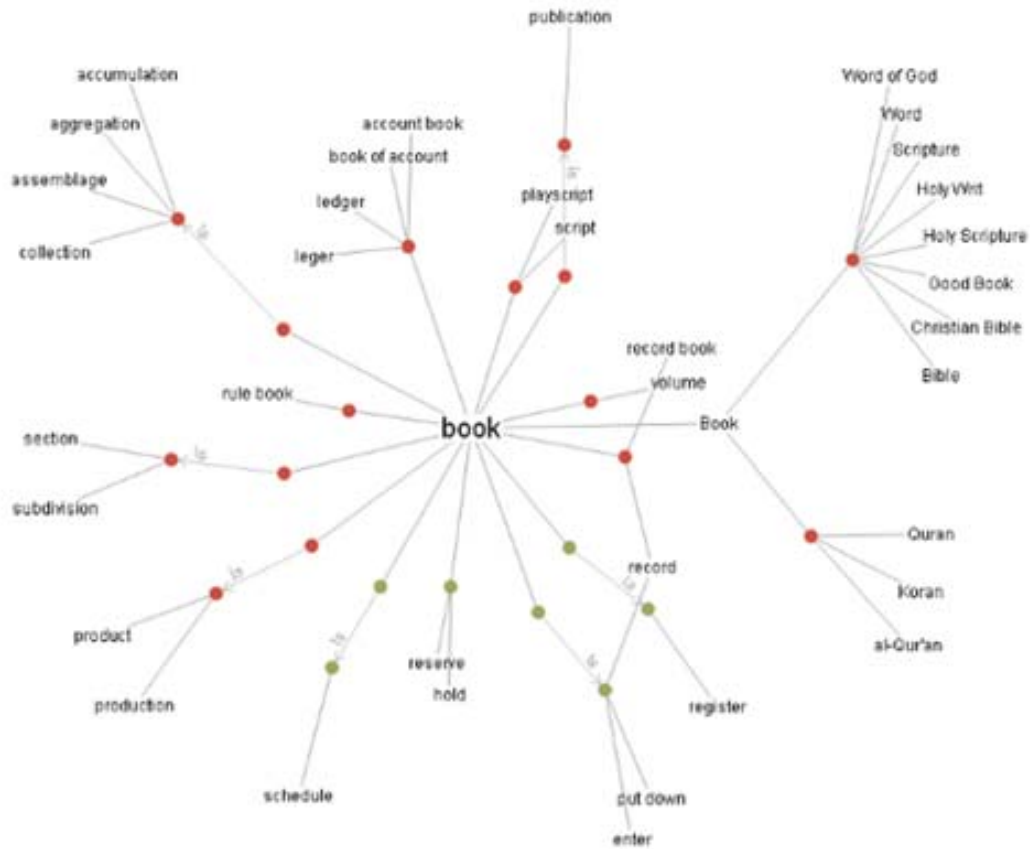


Figure 5.2: Example of WordNet semantic network of the word book.

The detailed explanation and different examples for every operation are as follows:

- **Synonym Replacement (SR):** in this operation n words are replaced with their synonyms using WordNet. For example, the original requirement: "the product shall allow for intuitive searching of available conference rooms" is augmented using SR to be "the **intersection** shall allow for intuitive searching of available conference rooms"

- Random Insertion (RI): randomly, n words have been selected and finding their synonyms using WordNet before inserting them in a random position. For example, the original requirement: "the product shall allow for intuitive searching of available conference rooms" is augmented using RI to be "the product shall allow **elbow room** for intuitive searching of available conference **rooms**"
- Random Swap (RS): swapping two randomly selected words in the sentence. For example, the original requirement: "the product shall allow for intuitive searching of available conference rooms" is augmented using RS to be "the product shall allow for intuitive **available** of **searching** conference rooms"
- Random Deletion (RD): deleting n random words from the sentence. For example, the original requirement: "the product shall allow for intuitive searching of available conference rooms" is augmented using RD to be "the product shall allow for **intuitive** searching of available conference rooms"

After doing the four augmented operations on the sentence separately, the augmented sentences are generated. The architecture of the common data augmentation technique is shown in Figure 5.3, and the numbers refer to the following shortcomings.

5.2.3.2 Shortcomings in the common data augmentation technique

Several shortcomings have been addressed in the existing technique. The main four shortcomings can be summarized in the following three points:

1. Static parameters that can be assigned regardless of the characteristics of the sentence is one of the main limitations in the existing technique. The class

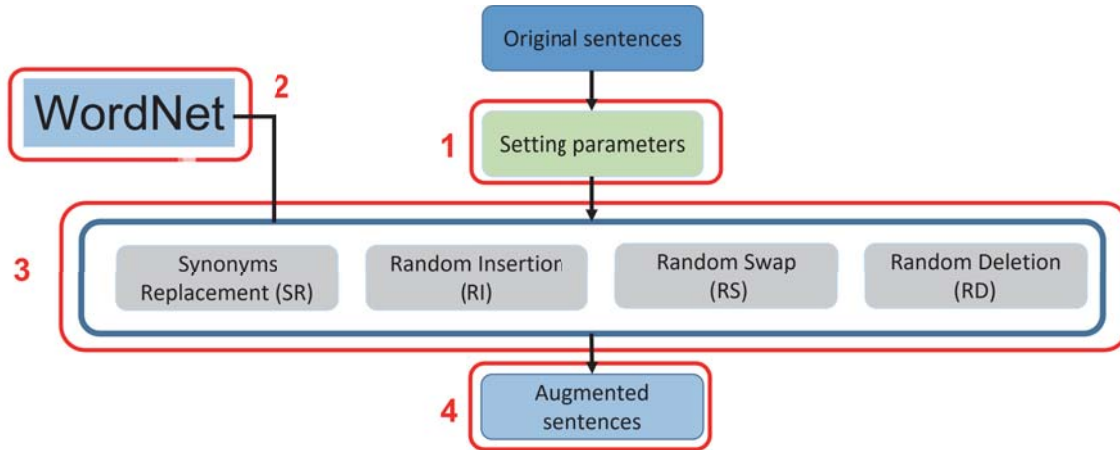


Figure 5.3: Architecture of common data augmentation that have been proposed in [2].

percentage will be maintained if we augment the samples without considering their size. The class imbalance will increase, affecting the classifiers' performance to bias toward the majority class. The length of a sentence can be used to indicate the capability of the sentence to generate more augmented samples compared with short sentences that might lose valuable features and generate redundant augmented samples. For example, the original sentence "the system shall achieve 95 up time" with a length equal to 7 can be augmented to be "the shall 95 up time" if we assign the number of augmentation to be 8. the maximum sentence length should give more chance to be augmented.

2. WordNet is used to find the synonyms of the words. The main limitation of this dictionary is the static replacement regardless of the context. For example, the word in the original sentence "the **product** shall allow for intuitive searching of available conference rooms" has several synonyms using WordNet, which are ('cartesian product', 'mathematical product', 'intersection', 'ware', 'merchandise', 'production'). One augmented sentence is "the **intersection** shall allow

for intuitive searching of available conference rooms".

3. High randomness by conducting three operations (deletion, swapping, insertion) that might increase the distortion of the augmented sentences.
4. Lack of validation stage is one of the main limitations of the common data augmentation technique. Augmentation operations might alter the sentence's meaning and add noisy features in the training split.

5.2.3.3 Architecture of proposed data augmentation technique

The proposed technique consists of four main parts:-

- Dynamic parameters setting: two parameters have been tuned to be set dynamically. The first parameter is the percent of change. In this parameter, we mainly used sentence length as the main factor in this setting. The percent of change of the sample depends primarily on its length and can be computed by the following equation where max is the length of the longest requirement and min is the length of the shortest requirement in the training part.

$$”PercentofChange” = ((\frac{(samplelength - min)}{(max - min)})/3) + 0.05$$

The second important factor is the number of augmentation that is required for every sample. We used two factors to decide the required number of augmentation: the class weight and the requirement length. The first equation is computing the number of required augmentation (N) by subtracting the num-

ber of samples in a class from the number of samples in the majority class as in equation 5.1. Then, the weight of every sample in a class is computed by dividing the length of the sample by the sum of the lengths of all samples as in equation 5.2. The computed sample weight will be multiplied by the number of augmentation (N) to get the number of augmented sentences that need to be generated from that sample as in equation 5.3.

$$Numberofaugmentation(N) = Max(class) - Class(X) \quad (5.1)$$

$$Weight(sample(length)) = \frac{SampleLength}{AllSamplesLength} \quad (5.2)$$

$$Numberofaugmentation(sample) = Weight(sample(length)) \times N \quad (5.3)$$

The pseudo-code of weighted class and length augmentation for one label requirement is shown in the algorithm 1.

The multilabel class needs to be augmented by considering the association between the classes in the samples. The first step is computing the weight of the class by dividing the number of samples in a specific class by the total number of labels in the dataset as in equation 5.4:

Algorithm 1 Pseudo-code of weighted class and length augmentation for one label requirement.

Input: Original requirements

- 1: Find the maximum (max) number of samples in majority class.
 - 2: Multiply max by the factor that need to be replicated
 - 3: **for** All classes in the dataset *iteration* = 1, 2, ... **do**
 - 4: Compute the total number of requirements in the class
 - 5: Subtract the total of the class by the total number of samples in the majority class to find the required number of augmentation.
 - 6: **for** All requirements *iteration* = 1, 2, ... **do**
 - 7: Compute the weights of the samples in the class based on their length
 - 8: Multiply the weights by the required number of augmentation.
 - 9: Print the augmented requirements
-

$$Weight(Class) = \frac{(ClassX)}{(AllClasses)} \quad (5.4)$$

The second step is computing the weight of the sample based on the association to the class as in equation 5.5:

$$weight(Sample(class)) = \frac{\sum \frac{1}{WeightOfClass}}{NoOfTrueLabels} \quad (5.5)$$

Then, the sample weight is based on the length of the requirement by dividing the length of the sample by the total length of all the samples as in equation 5.6:

$$weight(Sample(length)) = \frac{SampleLength}{AllSamplesLength} \quad (5.6)$$

The weight of the sample has been computed by using regularization factor α to balance between the weight of the sample by class and length as in equation 5.7:

$$weight(Sample) = (\alpha)weight(Sample(L)) + (1 - \alpha)weight(Sample(C)) \quad (5.7)$$

The pseudo-code of weighted length and one label class augmentation is shown in algorithm 2.

Algorithm 2 Pseudo-code of weighted class and length augmentation for multilabel requirement.

Input: Original requirements

- 1: **for** All classes in the dataset *iteration* = 1, 2, ... **do**
 - 2: Compute class weights by dividing the total number of samples in a class by the total number of true labels in the datasets.
 - 3: Sum the total weights of the sample based on class weights
 - 4: Compute class weights by dividing the inverse of the weights by the total number of true labels of the sample.
 - 5: Compute length weights by dividing the length of the sample by the total number of lengths in the datasets.
 - 6: Compute the sample weight using class and length by using alpha.
 - 7: Divide the computed sample weight by all samples weights and multiply by the total number of required augmentation
 - 8: Print the augmented requirements
-

- RoBERTa: In Natural Language Processing (NLP), BERT is a word embedding that uses a Transformer to learn contextual relations between words (or sub-words) in a text [18]. Google AI introduced BERT in 2019 to offer the capacity to compute bidirectional contextualized word representations. Transformer adopts a self-attention mechanism that includes two separate mechanisms (1) an encoder that reads the text input and (2) a decoder that produces a prediction for the task. As BERT is aimed at generating a language model, only the encoder mechanism is necessary. BERT differentially weights the significance of each part of the input data. On the one hand, there are directional language

models where the text input is read sequentially (i.e., left-to-right or right-to-left). The Transformer encoder reads the text input as an entire sequence of words at once. Therefore, BERT is considered bidirectional, or to be more accurate to classify it as non-directional. This is an advantage of BERT as it allows the model to learn the context of a word based on its surrounding text (i.e., left and right of the word).

- Dictionary-based: The domain dictionary has been collected by searching the main keywords and features of every class in nonfunctional requirements in different datasets and websites explaining the categories of nonfunctional requirements. After that, the minimum number of relevant synonyms of every word was collected using the WordNet and Thesaurus website. Three examples that represent three classes security, operational, and usability classes are shown in Figure 5.4.

Security class

```
["authorized", "accredited", "certified", "licensed",  
"approved", "legitimate"]
```

Operational class

```
["operate", "run", "work", "engage", "function"]
```

Usability class

```
["self explanatory", "straightforward", "distinguishable",  
"clear", "understandable"]
```

Figure 5.4: Examples of three words with their synonyms that belong to three non-functional classes.

- Validation stage: cosine similarity is often used to find the similarity between

two vectors. The cosine of the angle between two vectors is calculated to determine whether two vectors are pointing in the same direction. The two vectors represent the statement that we need to compare and is generated using BERT transformer. The cosine similarity is a widely-used concept in text analysis to measure document similarity. It helps find the similarity between two documents or text by calculating the cosine of the angle between the two vectors. The resulting cosine value indicates the degree to which these two vectors are distinct. Mathematically, cosine similarity is defined with the following equation:

$$\cos(\theta) = \frac{\mathbf{A}\mathbf{B}}{\|\mathbf{A}\|_2\|\mathbf{B}\|_2} \quad (5.8)$$

The numerator of the equation represents the inner product of the vectors $\mathbf{A}$ and $\mathbf{B}$ that represent two statements we need to compare, and it will be large if the vectors are very similar. The architecture of the proposed dictionary and transformer-based data augmentation technique in software requirement classification is shown in Figure 5.5

5.2.4 Architecture of deep learning Classifier

The RNN-based classifier consists of five consecutive layers: embedding layer, Bi-GRU layer, dense layer, dropout layer, and dense layer. As a variant of LSTM [103], GRUs [68] have fewer hyper-parameters and effectively solve the problem of the vanishing gradient in RNN. Furthermore, according to the literature [104], GRUs can effectively record and reset context information in the text, need fewer epochs, and outperform

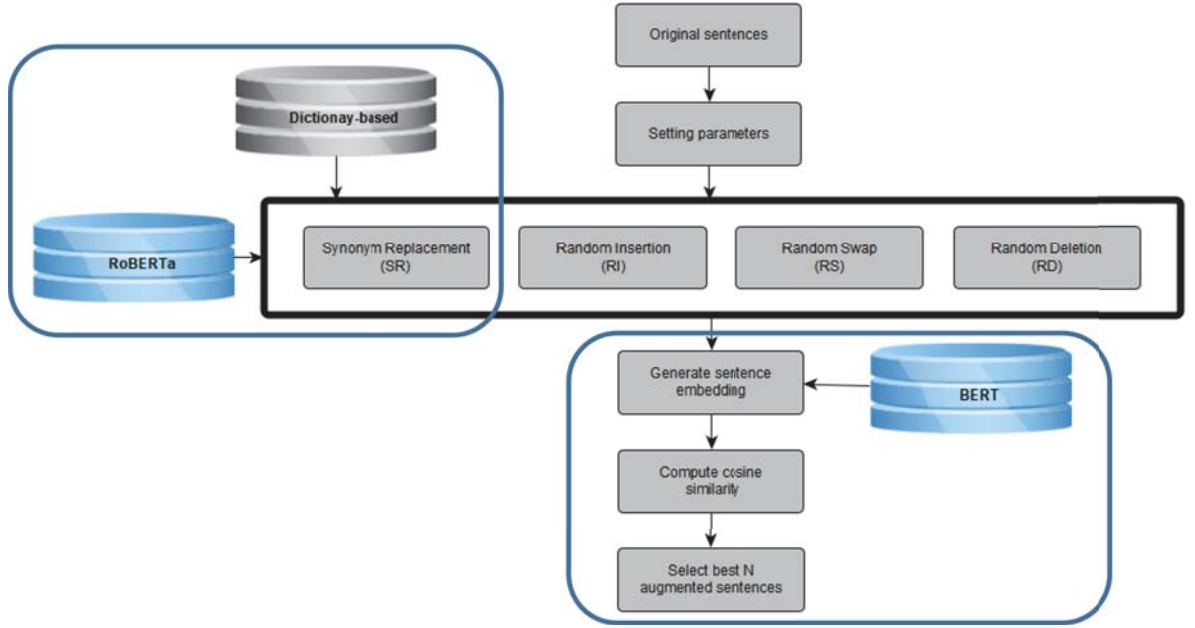


Figure 5.5: Architecture of the proposed dictionary and transformer-based data augmentation technique in software requirement classification.

LSTM. The details of this layers have been mentioned in section 4.2.2.

5.2.5 Evaluation Measures

The metrics provided in the conducted experiments, as well as their definitions, are discussed in this section. We chose four metrics to compare the different classifiers based on the literature. The confusion matrix is presented in binary and multi-class classification to describe the performance of the models classification. The example of the confusion matrix is shown in Figure 6.3. The definitions of accuracy, precision, recall, and F1-score using the confusion matrix are explained in section 4.3.1.

5.3 Experiment Setup

In this section, we illustrate the environments that are set to conduct the experiments. We used the Google Colab that supports GPU processors with the Keras library that was built on the TensorFlow platform to conduct all of our experiments. Using an iterative procedure, the experiment’s hyper-parameters are tuned differently to attain higher scores in the validation data. The five important hyperparameters tuned are the input length, embedding size, number of neurons in both bi-GRU and dense layer, the dropout rate, and the batch size. To assess the classifier, several validation splits were utilized; we calculated the precision, recall, and F1-measure for each classifier using stratified n-fold cross-validation. The details about the experiments can be accessed by the link. ¹

5.4 Results and Discussion

In this section, the results of existing techniques are evaluated on three modalities of the PROMISE dataset. In addition, the classifier’s performance is assessed on the EHR dataset. The weighted class and length software requirements augmentation and dictionary and transformer-based software requirements augmentation are presented in the following two subsections.

5.4.1 Results of Common Data Augmentation Technique

The common data augmentation technique is evaluated on the three modalities of the PROMISE dataset. The performance of common data augmentation on a limited

dataset shows an improvement compared with non-augmented training splits. The percentage of the selected training has been chosen randomly using the same seed in both without and with augmentation experiments. The less the dataset, the more improvement the common data augmentation has shown as can be seen from Table 5.3.

PROMISE (TEN)	Without Data Augmentation	With Common Data Augmentation
10%	38.36% (+/- 4.96%)	46.50% (+/- 6.74%)
20%	47.98% (+/- 4.28%)	57.57% (+/- 6.43%)
30%	55.50% (+/- 9.21%)	60.61% (+/- 6.42%)
40%	55.79% (+/- 6.63%)	63.31% (+/- 3.54%)
50%	63.13% (+/- 3.99%)	67.22% (+/- 5.19%)
60%	66.23% (+/- 9.29%)	67.62% (+/- 6.90%)
70%	67.26% (+/- 3.94%)	70.05% (+/- 5.14%)
80%	67.72% (+/- 7.03%)	77.23% (+/- 3.98%)
90%	71.49% (+/- 7.80%)	74.99% (+/- 5.75%)

Table 5.3: Results of using common data augmentation technique on several portions of PROMISE dataset.

The common data augmentation was evaluated on the PROMISE dataset. In binary and four-class problems, the common data augmentation technique improves the F-score only 1% to be 95% in binary classification and 1% to be 88% in four-class, respectively. In the ten-class problem, the common data augmentation achieved 78% that improves the classification with 3%. The comparison between the F-score of the common data augmentation and non-augmented datasets in three different modalities is shown in Table 5.4

PROMISE	Binary	Four	Ten
Without Augmentation	0.94	0.87	0.75
With Common Data Augmentation	0.95	0.88	0.78

Table 5.4: Comparison between the F-score of the augmented and non-augmented datasets in three different modalities.

5.4.2 Results of The Proposed Techniques

The following two subsections present the weighted class and length software requirements augmentation and dictionary and transformer-based software requirements augmentation. The details of these techniques are elaborated with the results of using these techniques.

5.4.2.1 Weighted Class and Length-based Augmentation Technique

The two parameters: percent of change and number of augmentation, have been incorporated in weighted class and length software requirements augmentation. The adaptive percent of change assigns value based on the requirement length. For example, the longest requirement in training is 70 words, and the shortest requirement is five words. The requirement "the system shall achieve 95 up time" with seven words will be assigned a percent of change value 0.08 by computing the following equation:

$$PercentofChange(sample) = ((\frac{7 - 5}{70 - 5})/3) + 0.05 = 0.08$$

The other parameter, the number of augmentation, can be set in different ways in one and multilabel requirements. In one label, the first step is assigning the number of augmentation that needs to be added to make the dataset balance. Then this number is distributed among the samples in the class based on their length as in Table 5.5. The last column is rounded to be an integer number.

The variations between the common and weighted one label techniques in tackling

Requirement	Length	weight	Number of augmentation
Requirement 1	10 words	0.15	$\text{round}(6.92) = 7$
Requirement 2	15 words	0.23	$\text{round}(10.38) = 11$
Requirement 3	25 words	0.38	$\text{round}(17.31) = 18$
Requirement 4	15 words	0.23	$\text{round}(10.38) = 11$
sum	65 words	1	45

Table 5.5: Computing the number of augmentation that need to be generated based on the requirement length in one label software requirements augmentation technique.

the imbalanced issue can be shown in the subsequent figures. Using the common augmentation technique the difference between functional and nonfunctional requirements can reach up to 400 samples as in Figure 5.6 while using the weighted augmentation technique the difference between them less than 40 samples as in Figure 5.7. The four classes in PROMISE dataset contain similar number of samples which minimize the difference between augmented dataset using common augmentation technique as in Figure 5.8 and weighted augmentation technique as in Figure 5.9. The variation can be noticed in binary and ten-class problems. For example, the majority class usability augmented to be up to 400 using the common technique and up to 250 augmented samples using the weighted augmentation technique as in Figures 5.10 and 5.11.

The common and weighted one-label augmentation techniques have been evaluated on binary, four, and ten PROMISE datasets. The weighted data augmentation technique did not improve the F-score in the binary classification problem. However, in four classification problem, the weighted data augmentation technique improves the F-score by only 1% to 89%. In the ten-class problem, the weighted data augmentation achieved 76%, which is a 3 % improvement compared with the common data augmentation. The comparison between the common and weighted data augmentation in

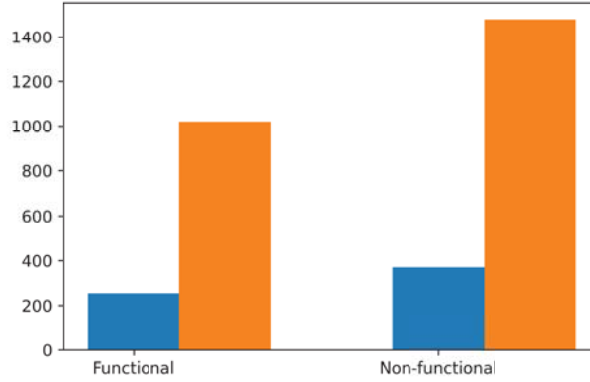


Figure 5.6: The statistics of classes using the common one label augmentation on binary PROMISE dataset.

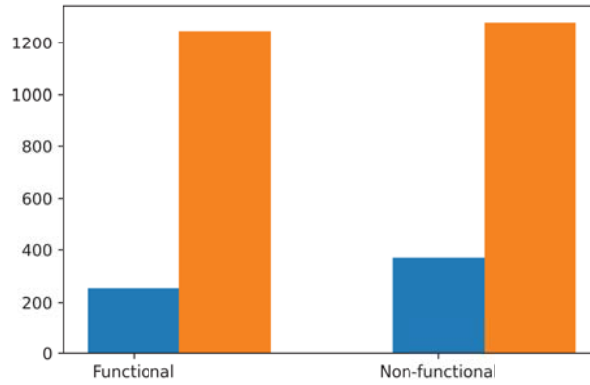


Figure 5.7: The statistics of classes using the weighted one label augmentation on binary PROMISE dataset.

three different modalities is shown in Table 5.6.

For multilabel classification, the common and weighted one-label augmentation techniques have been evaluated on multiclass multilabel EHR datasets. The binary classification has been excluded because of the balanced samples of functional and nonfunctional requirements in EHR dataset. The weighted class and length-based augmentation can differentiate the number of augmentations based on the class label and the requirement length. The value of α have been set by using trail and error,

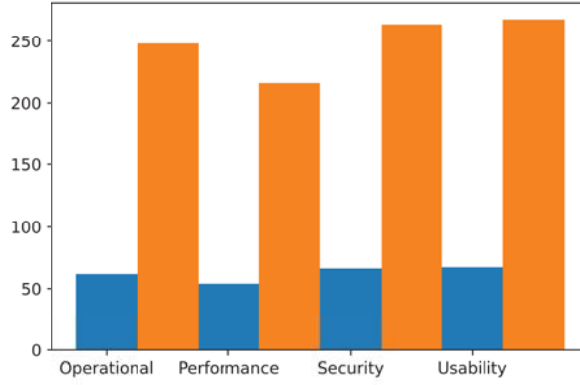


Figure 5.8: The statistics of classes using the common one label augmentation on four PROMISE dataset.

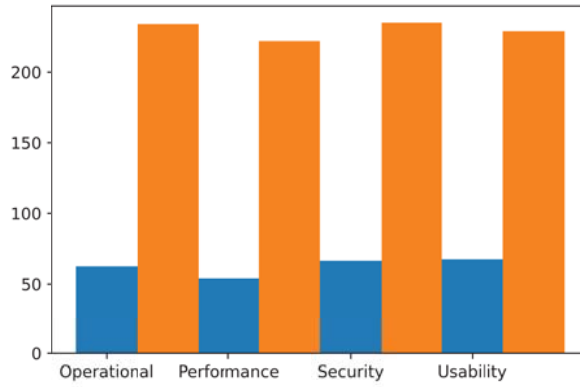


Figure 5.9: The statistics of classes using the weighted one label augmentation on four PROMISE dataset.

however we assign more weight to the class because we are trying to tackle the issue of the imbalance in the dataset. The weighted multilabel augmentation results are shown in Table 5.8. The best F-score has been achieved 0.61 with α value equal 0.15. The range of α values between 0.1 to 0.25 and the classifier scored F-score 0.58, 0.61, 0.60, 0.56 using α values equal to 0.1, 0.15, 0.2, 0.25. Before we use α , we scale the weight based on the length to have an equal sum similar to weight based on the class. The impact of α can be noticed in Table 5.7. when α equals 0.05, the length has a

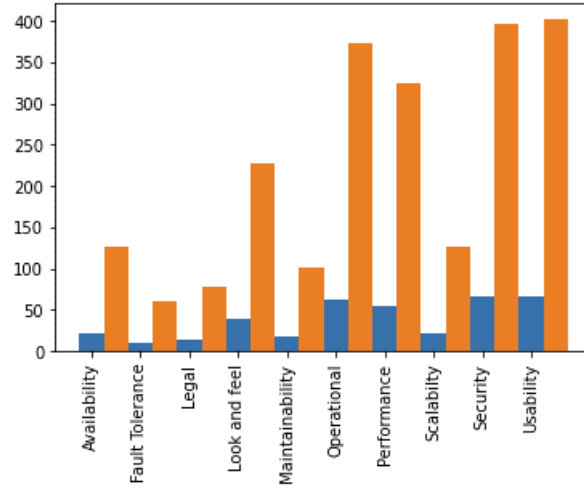


Figure 5.10: The statistics of classes using the common one label augmentation on ten PROMISE dataset.

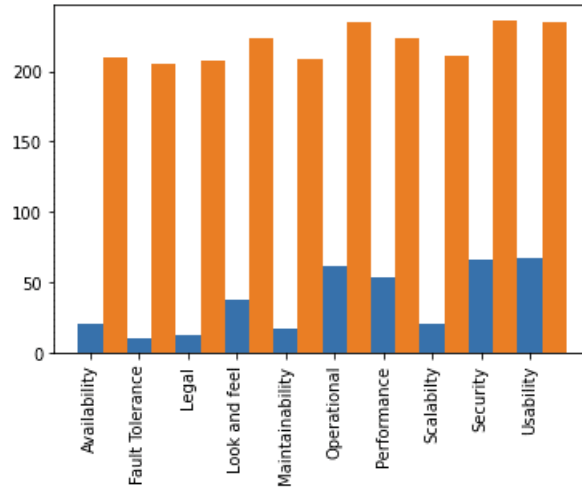


Figure 5.11: The statistics of classes using the weighted one label augmentation on ten PROMISE dataset.

limited impact on the number of augmentations, but when we increase α to be equal to 0.2, the length changes the number of augmentations between two requirements.

The variations between the common and weighted multilabel techniques in tackling the imbalanced issue can be shown in Figures 5.12 and 5.13. In the common technique, the access control augmented up to 5000, but the weighted technique augments the

PROMISE	Binary (F1 Macro)
With Common Data Augmentation	0.94
With Weighted Data Augmentation	0.92
PROMISE	Four (F1 Macro)
With Common Data Augmentation	0.87
With Weighted Data Augmentation	0.89
PROMISE	Ten (F1 Macro)
With Common Data Augmentation	0.73
With Weighted Data Augmentation	0.76

Table 5.6: Comparison between the results of common and weighted one label augmentation on the PROMISE dataset.

Requirement	Weight(class)	Weight(length)	No augmentation ($\alpha = \mathbf{0.05}$)
Requirement 1	7.9	33.28	9.2
Requirement 2	7.9	6.4	7.8
Requirement	Weight(class)	Weight(length)	No augmentation ($\alpha = \mathbf{0.2}$)
Requirement 1	7.9	33.28	13.1
Requirement 2	7.9	6.4	7.7

Table 5.7: Computing the number of augmentation that need to be generated based on the class and requirement length in multilabel software requirements augmentation technique.

samples of access control class up to 2000 samples, and the minority classes have a chance to be augmented.

For example, the association between two minority classes (Availability and others) in the sentence "the solution should provide the ability to conform to data retention availability and destruction to provide the ability to inactivate obsolete or destroy unstructured health record information" has been augmented 15 times while the association between minority and majority classes (Availability and access control) in the sentence "implement administrative physical and technical safeguards that reasonably and appropriately protect the confidentiality integrity and availability of the electronic protected health information that it creates receives maintains or transmits on behalf of the group health plan" has been augmented 7 times. The comparison between the F-score of the common and weighted data augmentation and in the EHR dataset is

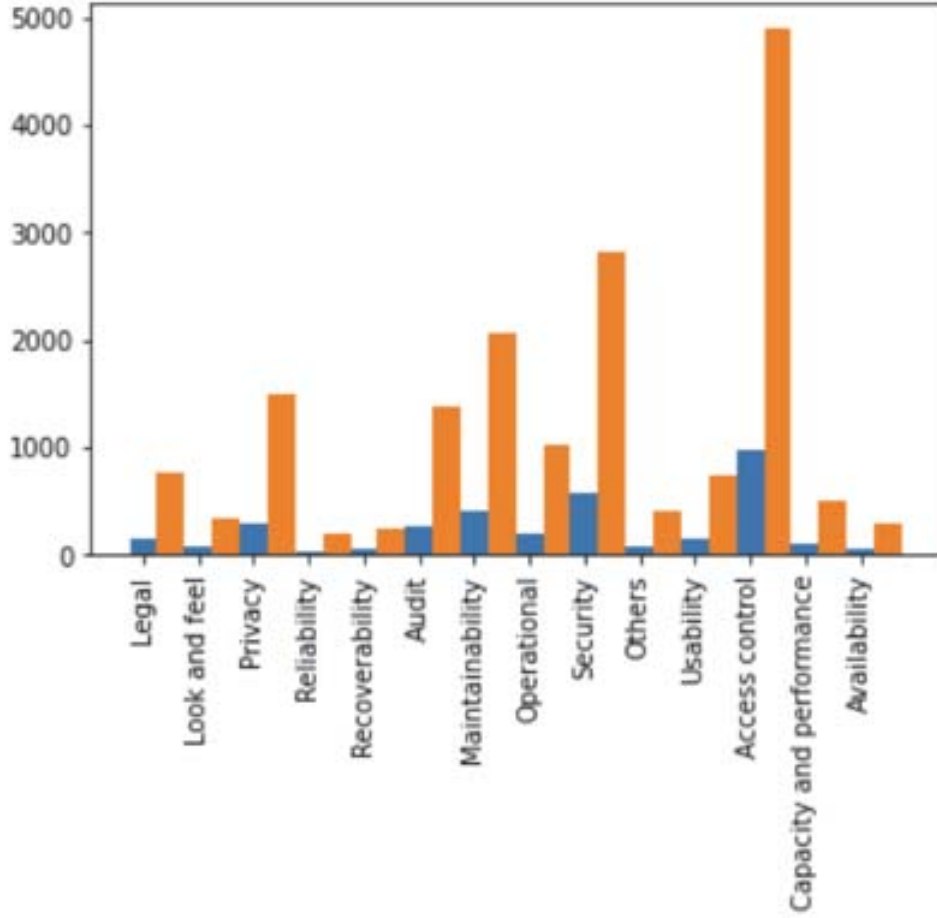


Figure 5.12: The statistics of classes using common multilabel augmentation on EHR dataset.

shown in Table 5.8.

5.4.2.2 Dictionary and Transformer-based Augmentation

In this proposed model, the dictionary can maintain the main keywords, and the transformer tackles the issue of replacing the remaining words. The RoBERTa transformer has been selected after several pilot studies to guarantee its suitability to the software requirement domain. The contextual transformer returns the best five synonyms, and randomly, one of them will be selected for the replacement. In the validation stage, we

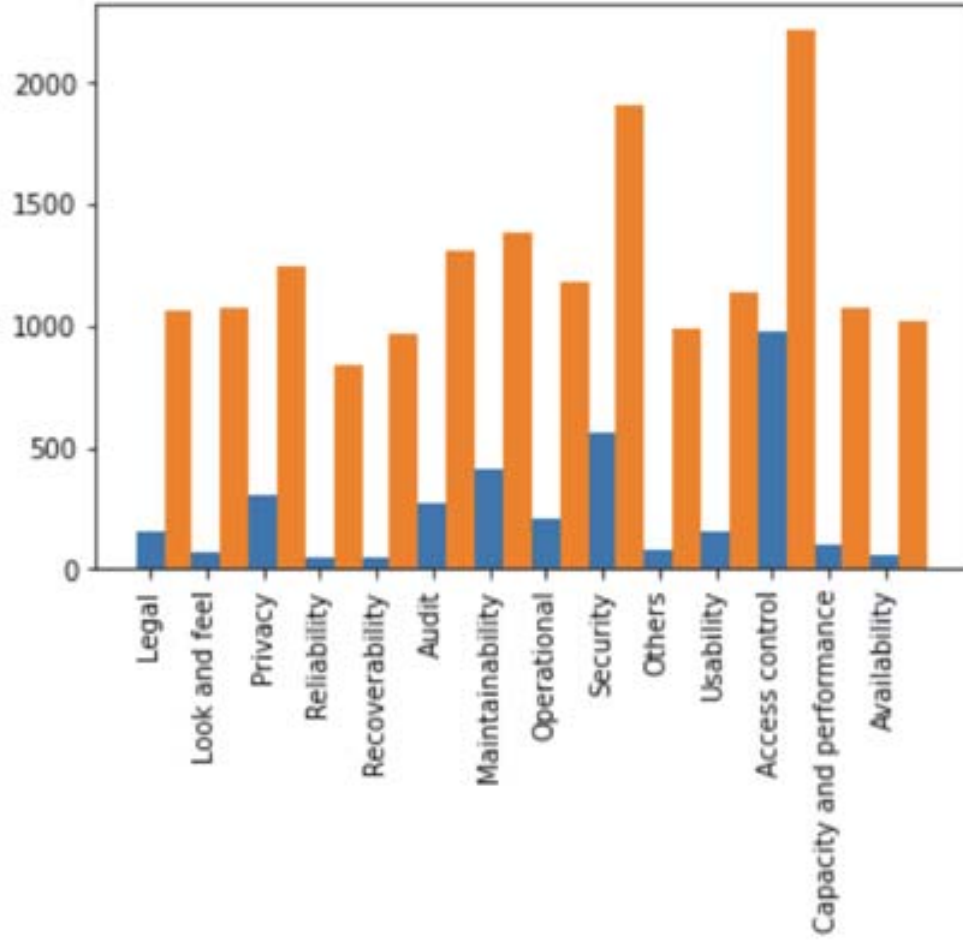


Figure 5.13: The statistics of classes using the weighted multilabel augmentation on EHR dataset.

used the BERT transformers to generate the embedding of the original and augmented sentences. Using trial and error, we used values (20, 40, 60) to be the number of generated sentences and selected the best eight augmented sentences recommended in the literature [2]. We notice that the more we generate samples, the higher percentage of similarity will be selected, which minimizes the variation between the augmented samples. The overall steps of the validation stage in the dictionary and transformer-based software requirements augmentation is shown in Figure 5.14.

The dictionary-based and transformer-based augmentation technique augmenta-

EHR	F1-Macro
With Common Data Augmentation	0.56
With Weighted Data Augmentation	0.61

Table 5.8: Comparison between the results of common and weighted multilabel augmentation on EHR dataset.

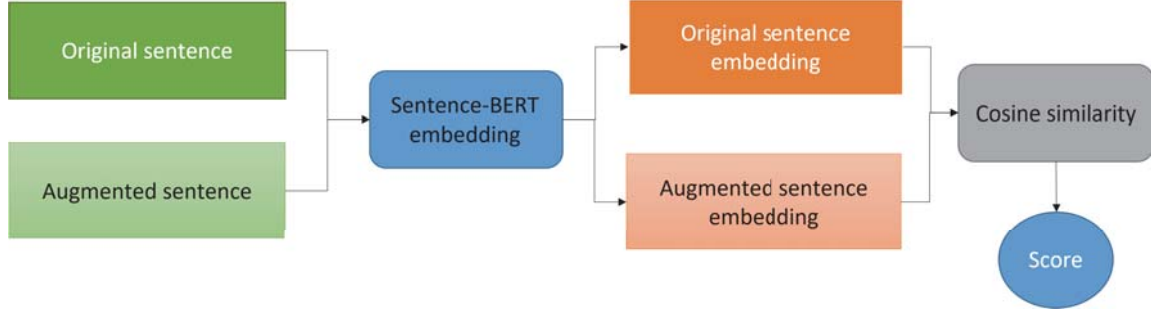


Figure 5.14: Steps of validation stage in dictionary and transformer-based software requirements augmentation.

tion techniques have been evaluated on four and ten PROMISE datasets. The binary classification have been excluded because of its simplicity and the capability to reach 95% with using common data augmentation technique. Classifying the requirements by using dictionary-based and transformer-based augmentation technique improves the F-score in four-class and ten-class problems. The common data augmentation evaluated on two modalities of PROMISE dataset. In four-class problem, the common technique improves the F-score only 1% to be 88%. However, the proposed technique improve the F-score and achieved 91%. In the ten-class problem, the common data augmentation achieved 78% and the proposed technique scored 80%. The results of different techniques are shown in Table 5.9 and Table 5.10.

PROMISE	PROMISE: four classification
Without Augmentation	0.87
With Common Data Augmentation	0.88
Dictionary and Transformer-based Augmentation	0.91

Table 5.9: Comparison between the results of common and dictionary and transformer-based software requirements augmentation on four classes PROMISE dataset.

PROMISE	PROMISE: Ten classification
Without Augmentation	0.75
With Common Augmentation	0.78
Dictionary and Transformer-based Augmentation	0.80

Table 5.10: Comparison between the results of common and dictionary and transformer-based software requirements augmentation on ten classes PROMISE dataset.

Generally, the weighted class and length augmentation techniques improve the classification compared with the technique that assign equal weights to the samples. State of the art techniques frequently choose to assess their multilabel classification techniques using metrics that might not represent the classification results of the minority classes. The proposed techniques improve the classification by assigning higher weights to the minority compared with the majority classes.

Based on the literature, using binarized Naïve Bayes (BNB) with processed and unprocessed datasets doubled precision and recall to 90% and 91% respectively, as in [81]. The processed dataset has been produced by applying different manual unification rules that improve the classification. The domain-vocabulary and transformer-based augmentation automate this process except the task of collecting the vocabulary. This threats has been mitigated by minimizing the number of words in the vocabulary.

5.5 Conclusion

Data scarcity and imbalanced datasets are two main challenges in software requirements classification. Two approaches based on class and length were proposed to assign dynamic weights to every sample. The presented technique achieved higher results compared with a popular text data augmentation technique. Using the com-

mon data augmentation techniques without considering the domain characteristics did not achieve excellent results. In addition, we presented augmentation techniques based on domain-vocabulary and transformer to address the domain-specific nature of the problem and prevent the distortion of the main features in the augmented sample. The proposed technique achieved a 3% improvement compared with the common data augmentation in four-class problem and 2% improvement in ten-class problem.

CHAPTER 6

SOFTWARE REQUIREMENT CLASSIFICATION IN A LOW RESOURCE LANGUAGE

6.1 Introduction

In the previous two chapters, the datasets that have been used is limited and imbalanced which are main issues of data scarcity. However, in some cases the data scarcity issues can be harder by the lack of labeled datasets. To address this issue in this chapter, different DL-based models have been proposed to improve the performance of requirements categorization in Arabic by utilizing resources accessible in rich-resource languages. The translation from low-resource to rich-resource language and vice versa have been experimented to check whether classifying directly the native better than translated SRS documents. In addition, two different tokenization

approaches have been used to find the the best tokenization that is suitable to Arabic requirements classification. The tokenized words and characters sequences have been used to classify binary and multi-class Arabic requirements. The impact of preprocessing on the performance of classifiers has been investigated

6.2 Research Design

The datasets used in the study are presented in this section. Then, we discuss the classifier architecture that used in the study. Finally, the classification measures are defined. The next subsections describe a brief summary about them.

6.2.1 Existing Datasets

In the experiments, three datasets have been used and presented in the following subsection .

6.2.1.1 PROMISE

The most well-known dataset in the requirement classification domain is PROMISE dataset [8]. The dataset comprises 15 projects that span a wide range of domains and had written by students as a term project [5]. The dataset had written in English language and used in binary and multi-class classification problems. The 625 requirements had been annotated as follows: functional (F): 255, non-functional(NF): 370, and these 370 non-functional requirements had been classified to ten classes as the following: operational (O) : 62, security (SE) : 66, performance (PE): 54, usability (US):

67, legal (L): 13, scalability (SC): 21, fault tolerance (FT): 10, look and feel (LF): 38, maintainability (MN): 17, availability (A): 21. We used the name PROMISE for the English version and En2ArPROMISE for the Arabic version throughout this chapter.

Dataset		Min. length	Max. length	Mean length	Median length
English PROMISE	Word	6	84	19.89	17
	Character	40	494	117.95	103
Arabic PROMISE	Word	4	70	17.73	16
	Character	30	403	102.69	90

Table 6.1: Descriptive statistics of English and Arabic PROMISE datasets

6.2.1.2 Arabic News

The dataset consists of an Arabic articles published in newspaper that used formal Arabic language. The text that has been used contains only Arabic alphabetic, numeric and symbolic words without any foreign terminologies. The dataset in [111] is collected by semi-automatic web crawling process from three online newspapers: Assabah (www.assabah.ma), Hespress (www.hespress.com) and Akhbarona (www.akhbarona.com). The dataset's documents are divided into five categories: sport, politics, culture, economics, and diversity. The total number of articles in the dataset is 111,728 documents structured in text files. We only select 5000 samples in our experiments in which 1000 represents every class. We used the name ArNews for the Arabic version and Ar2ENNews for the English version throughout this chapter.

Dataset		Min. length	Max. length	Mean length	Median length
Arabic News	Word	7	89	62.7	67
English News	Word	9	124	72.94	74

Table 6.2: Descriptive statistics of English and Arabic News dataset

6.2.1.3 Arabic software requirements

The dataset contains functional and non-functional requirements of electronic records management systems. The requirements starts by the creation and management of records, identification of records, data description, classification schemes, reliability and authenticity of records, store and removal of records, search, retrieval and presentation, and system management. The total number of requirements is 299 requirements. The functional requirements are 106 and non-functional requirements are 193. The requirements have been annotated by two master students in software engineering. The agreement between them has been computed using Krippendorff's Alpha [112] which equals to 0.769. The value is higher than the accepted value which is 0.66 [113]. The disagreement between the annotators has been discussed in one meeting to provide final consistent labeling. We used the name ArSRS for the Arabic version and Ar2ENSRS for the English version throughout this chapter.

Dataset		Min. length	Max. length	Mean length	Median length
Arabic Requirement	Word	3	67	15.39	13
English Requirement	Word	3	94	18.78	16

Table 6.3: Descriptive statistics of English and Arabic requirement dataset

6.2.2 Research Methodology

The main steps to achieve the study's goals will be illustrated in the next subsections. The Figure 6.1 illustrates the framework of the research and the variations between the conducted experiments. Based on the **dataset**, we have three different datasets, the first one is the translated PROMISE to Arabic, the second dataset is Arabic news dataset and the translated one to English, the third dataset is Arabic software

Dataset	No. of classes	Type of inputs	Level of preprocessing	Evaluation setting
ArPromise	2, 4 10	word and character	4	10 Folds
ArNews	5	word	3	10 Folds
EnNews	5	word	3	10 Folds
ArSRS	2	word	1	cross-project
EnSRS	2	word	1	cross-project

Table 6.4: Comparison between the experiments

requirements and the translated version to English. We used the Google translator in all the experiments.

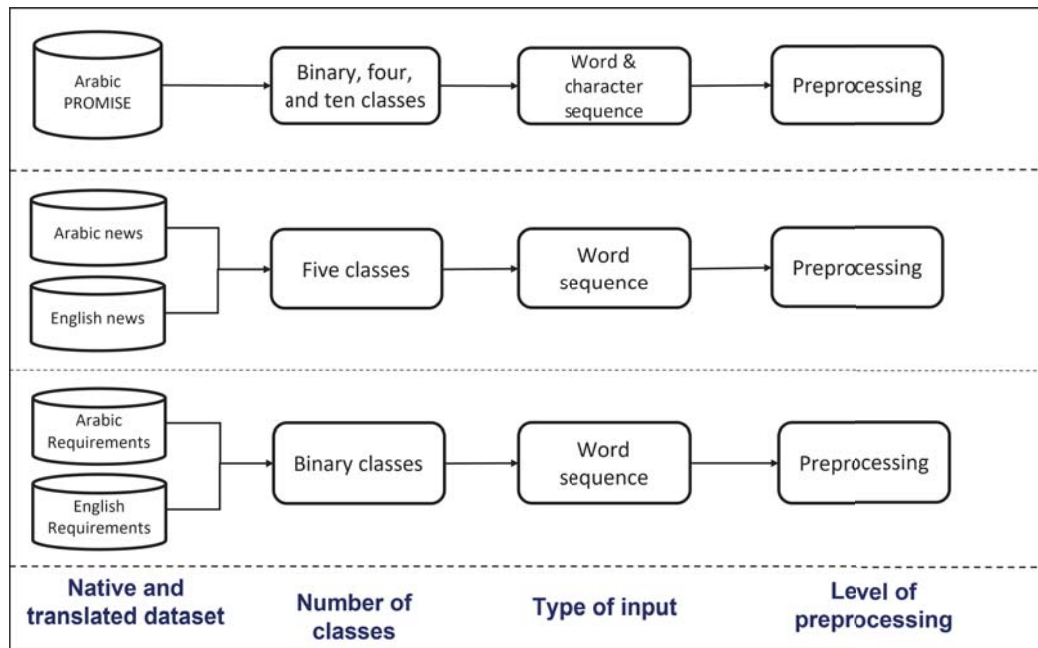


Figure 6.1: The research framework of the research

The impact of **number of classes** has been investigated by conducting several experiments. The experiments can be binary or multi-class classification. The performance of the classifiers based on the **type of inputs** either word or character sequence have been studied in Arabic PROMISE dataset. The **level of preprocessing** is the degree of using NLP preprocessing step such as stop words removal and lemmatization. A comparison between the experiments is shown in Table 6.4.

The research methodology in this study can be classified to three major parts. The first part is preparing the dataset to be used in the experiments. The second part is building the deep learning classifier to be trained on the labeled dataset. The last part is evaluating the performance of the classifier and reporting the results. The main components of research methodology are shown in Figure 6.2.

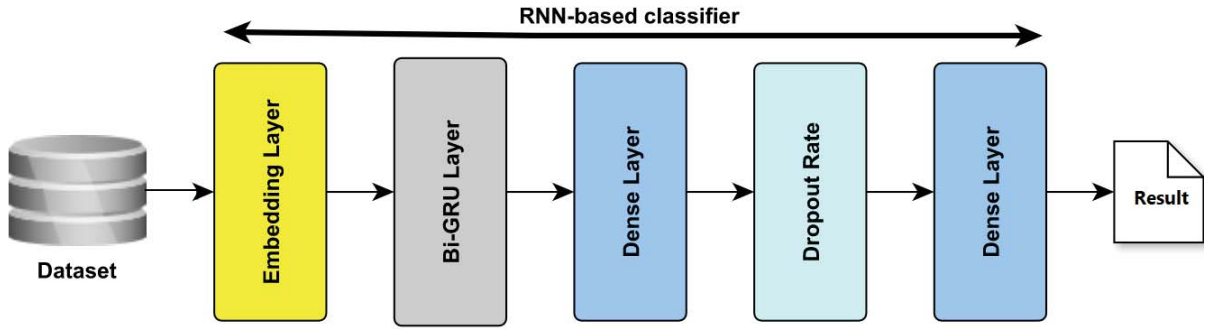


Figure 6.2: The main components of research methodology

The dataset have been selected from the community and prepared to be the input of the RNN classifier. Three datasets have been translated and preprocessed. By breaking the language into distinct words, the sentence is tokenized. Using an Arabic language package for Python **Araby** [114] tokenizer and punctuation removal functions, the punctuation tokens are then deleted from the phrase. Beside that araby library provides basic functions to manipulate Arabic letters and text such as removing the tashkeel from the words. The terminology or non-Arabic words have been removed to check their impact on the classification. The stop words list in Arabic and English language has limited capability to distinct between the classes. The lemmatization process has been used to minimize the size of the vocabulary by using **Farasa** lemmatizer [115]. the The degree of preprocessing has four levels:

1. Removing the punctuation
2. Removing non-Arabic words or terminology
3. Removing the stop words
4. Lemmatizing the words in the dictionary.

The dictionaries for the words and characters are constructed after preprocessing. Each word and character in the input sequence has a distinct integer number. The dictionary's size is determined by the number of unique words or characters in the corpus. For example, the dictionary's size can be 38 unique tokens. The 38 tokens are 28 Arabic alphabetic tokens, 7 tokens for some Arabic special letters such as () 26 English alphabetic, tokens 10 numbers from 0 to 9, a space and zero for padding. Since the length of the sentences is variable, the sentences will be padded to be the same length as the longest sentence to be fed to the proposed classifier.

As a variant of LSTM [103], GRUs [68] have fewer hyper-parameters and effectively solves the problem of the vanishing gradient in RNN. Furthermore, according to the literature [104], effectively GRUs can record and reset context information in the text, need less epochs, and outperform LSTM. The RNN-based classifier consists of five consecutive layers: embedding layer, Bi-GRU layer, dense layer, dropout layer, and dense layer.

The encoded words or characters are represented in dense vector in the embedding layer. Instead of employing pre-trained models, the encoded vectors of the words or characters are learnt from the dataset itself. The sequence of words or characters is then sent to Bi-GRU layer. After the Bi-GRU layer, the fully linked layer or dense

layer is utilized to introduce non-linearity to the model. In the dense layer, the ReLU activation function is utilized to assist the model converge in a few epochs. To avoid overfitting, the dropout is then utilized in all the models. The output layer, which is one **sigmoid** function in binary classification or the **SoftMax** function in multi-class classification, is the final layer in the models.

Binary cross-entropy is used in binary classification, whereas categorical cross-entropy is used in multi-class classification. The Adam optimizer is employed in accordance with the literature's recommendations. Following that, a split plan will be used to execute the training and testing stage. Finally, depending on the predicted labels, the evaluation measures will be computed. In the training and testing stages, the effectiveness of the proposed models is evaluated by training the model on 90% of the samples and validating the model on 10% and repeating this process for 10 iterations. The evaluation metrics are calculated to be compared with the studies in the literature.

6.2.3 Evaluation Measures

The metrics provided in the conducted experiments, as well as their definitions, are discussed in this section. We chose four metrics to compare the different classifiers based on the literature. 75% of main studies employed recall and precision metrics to measure the efficacy of the model according to the requirements classification review [8]. The imbalanced dataset requires metrics other than the accuracy such as F1-score. For the sake of optimization, the F1-score is a sum of two scores, recall and precision [116]. Using the Scikit-learn toolkit, four metrics are used to compare the

models throughout this experiment: accuracy, recall, precision, and F1-score. For the purpose of comparison, we compute precision, recall, and F1-score in multi-class classification using the weighted average, as described in [30], [97], [117].

		Actual	
		Positive	Negative
Predicted	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Figure 6.3: Confusion matrix

The confusion matrix is presented in binary and multi-class classification to describe the performance of the models classification. The example of the confusion matrix is shown in Figure 6.3. The definitions of accuracy, precision, recall, and F1-score using the confusion matrix are explained as follows:

- Accuracy: is calculated as the number of correct classifications divided by the total number of classifications. It is defined as in Eq. 6.1

$$Accuracy = \frac{(truepositives + truenegatives)}{(truepositives+truenegatives+falsepositives+falsenegatives)} \quad (6.1)$$

- Precision: estimates the total number of correct classifications with respect to

the number of classifications retrieved. It is defined as in Eq. 6.2

$$Precision = \frac{(truepositives)}{(truepositives + falsepositives)} \quad (6.2)$$

- Recall: estimates the percent of correct classifications. It is defined as in Eq.

6.3:

$$Recall = \frac{(truepositives)}{(truepositives + falsenegatives)} \quad (6.3)$$

- F-Score: is the harmonic mean of precision and recall. It is defined as in Eq.

6.4:

$$F1 - Score = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)} \quad (6.4)$$

6.3 Experiment Setup

In this section, we illustrate the environments that are set to conduct the experiments.

We used the Google Colab that supports GPU processors with Keras library that built on TensorFlow platform to conduct all of our experiments. Using an iterative procedure, the hyper-parameters are tuned differently in the experiments to attain higher scores in the validation data. The five important hyperparameters that have been tuned are the length of the input, embedding size, number of neurons in both bi-gru and dense layer, the dropout rate, and the batch size.

To assess the classifier, several validation splits were utilized, We calculated the precision, recall, and F1-measure for each classifier using stratified n-fold cross-validation. Another splits has been used to test the classifier performance on unknown

projects by splitting the dataset into training, validation and testing split. The details about the experiments can be accessed by the link. ¹

6.4 Results and Discussion

In this section, the results of the experiments in three datasets are elaborated in both either binary or multi-class classification. In the first subsection the experiments are conducted on ArPROMISE dataset with two, four, and ten classes. The next subsection is the results of five classes classification in the news dataset, and the results of binary requirements classification across projects.

6.4.1 Arabic Software Requirements Classification in En2ArPROMISE Dataset

The ultimate objective of this subsection is investigating the capability of classifying Arabic requirements. This investigation contains four dimensions that need to examine in Arabic requirements text: translation, preprocessing, number of classes, and input type as shown in Figure 6.4.

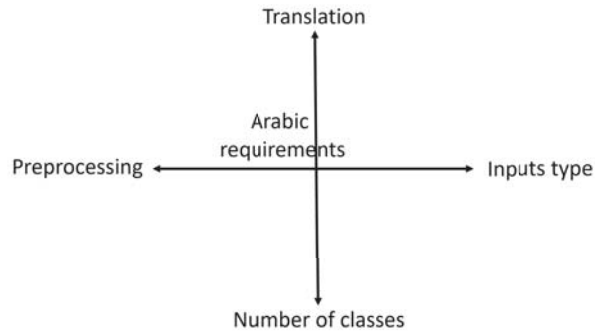


Figure 6.4: The four dimensions of conducting experiments on En2ArPROMISE

6.4.1.1 Arabic Binary Classification

The performance of the proposed models using word and character sequence have been evaluated using standard measures in Table 6.5. Clearly, a limited variations between precision and recall of the proposed model using word tokens compared with character tokens. The results of the binary models show that the word sequence model outperforms the character model in all the reported experiments. Different level of preprocessing have negligible impact on classification measures using the word sequence. In binary classification, the word sequence model achieved 8% higher than the character sequence model in F1-score measures.

Model Name	Preprocessing	Accuracy	Precision	Recall	F-score
Word sequence	Without Punctuation	90.71	89.67	88.22	88.61
	Terminology Free	91.03	90.14	88.20	88.97
	Without Stopwords	90.72	88.92	89.05	88.63
	With Lemmatization	91.04	90.86	87.43	88.78
Character sequence	Without Punctuation	82.73	81.39	76.05	77.72
	Terminology Free	79.82	74.35	77.65	75.56
	Without Stopwords	81.59	73.31	80.80	76.16
	With Lemmatization	85.12	84.50	79.68	81.14

Table 6.5: Word and character binary model results in En2ArPROMISE dataset

One technique to visualize the performance of classifiers is to use the confusion matrix. Clearly, the word model classifies the functional and non-functional requirements better than the character model. In word model, the difference between the misclassified functional and non-functional samples is 4 while the difference in character model is 11 samples which means the capability of the word model to mitigate the imbalanced classes is better than the character model. The values of the elements in confusion matrix are shown in Figure 6.5.

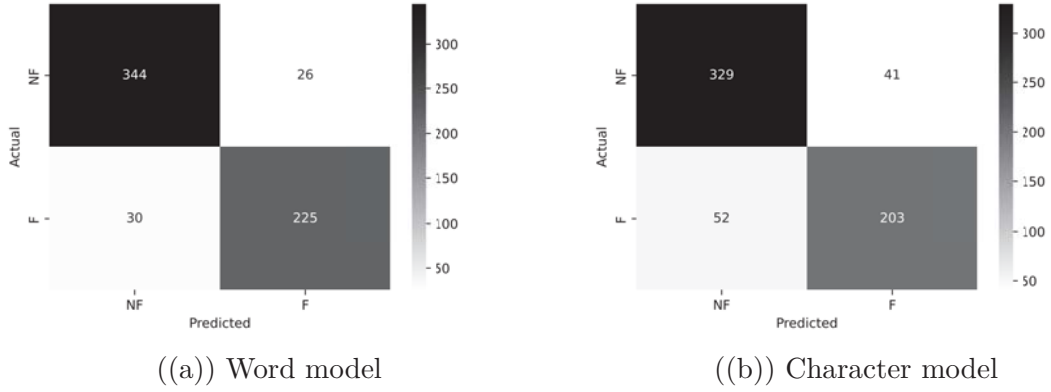


Figure 6.5: Word and character models in En2ArPROMISE classification

The classification of Arabic requirements has shown several challenges compared to English requirements classification. For example the maximum Arabic requirements length is 69 as in Table 6.6 compared to the English sentences which is 84 . The number of unique words in Arabic requirements is larger than the number of unique words in English requirements which is 1672 without any significant preprocessing. The removal of terminologies or stopwords has limited impact on the vocabulary size and input length. The lemmatization step reduce significantly the number of unique words in Arabic to the half (from 2549 to 1216).

Preprocessing level	Arabic length	Arabic unique words
Arabic Without Punctuation	69	2549
Arabic Terminology Free	69	2493
Arabic Without Stopwords	62	2478
Arabic With Lemmatization	62	1216
English Without Punctuation	84	1672

Table 6.6: Comparison between PROMISE and En2ArPROMISE datasets

There is a variation between the classification of functional and non-function requirements in word and character models. The capability of word model to minimize the impact of imbalance distribution of two classes has been shown in the previous

results. The character model might perform better if we have large dataset especially with rich morphology languages such as Arabic. The word-based sequence model outperformed the character-based sequence model even though the Arabic language is having rich morphology which imply a large vocabulary in word sequence compared with character sequence inputs. One of the main characteristics that has been noticed in Arabic sentence classification is increasing the size of vocabulary with small length compared with English language.

The rich morphology can be a challenge in Arabic requirements classification which plays an important role by increasing the percentage of unknown words in validating or testing split. For example the nonfunctional requirement that belongs to performance category: "The product shall **synchronize** with the office system every hour" is translated to Arabic to be "يجب أن يتزامن المنتج مع نظام المكتب كل ساعة" is misclassified to be functional requirements. The feature **synchronize** is part of two performance statements in English is translated to be two words in Arabic ترامن and مزامنة . The high probability of altering the features in Arabic can be on of the main source of misclassification. The lemmatization step fails to unified the two similar words to be encoded differently.

The lemmatization can be useful to increase the features that can be reliable in classifying the requirements. For example, statements that has word grid in representation constraints and in Arabic شبكة can be unified using the lemmatization especially from plural to singular forms. The existing of terminologies can be misleading in some cases because of their ability to maintain their shapes even after the applying the Arabic lemmatization tools. For example, the terminology in Windows Media Player "Movies shall be encoded and streamed via **Windows Media Player**

9.0" exists in two different requirements one is functional and another is nonfunctional. The translated statement of this requirement is misclassified by the Arabic model.

The literature was reviewed to find relevant researches that could be compared with our findings. The papers were chosen based on their discussion of functional and non-functional categorization, as well as their use of the PROMISE dataset. Since there is a lack of studies in Arabic requirements classification, the comparison is conducted with the English requirements classification to show how close the classification of translated to Arabic requirements. Similar to the literature, we used the recall, precision, and F-score in the binary classification comparison. The English requirements classification into functional and non-functional classes achieved higher F-score as in Table 6.7. The minimum value that has been scored is 0.8 in [92] and the maximum value is 0.98 in [9]. Most of the studies scored around 0.92 while the proposed model that classifies translated Arabic requirements scored 0.89 which is considered comparable results.

6.4.1.2 Arabic Multi-class Classification

The nonfunctional requirements in En2ArPROMISE dataset can be classified into four or ten classes by tokenizing the inputs into sequence of words or inputs. In the four classification problem, the preprocessing has improved the F-score by 6% in word model and 4% in character model. Table 6.8 shows the findings of four classification word and character based models.

Figure 6.6 shows that some misclassified sentences are more common in some classes. The lowest number of misclassified samples in both operational and security

Ref.	Type	category	Names of classes ¹	Features	Classifier ²	Recall	Precision	F1-score
[9]	Binary	Traditional Machine learning	FR, NFR	Syntactic Features and keywords	Decision tree	1.00	0.97	0.98
[10]			FR, NFR	meta-data, lexical, and syntactical	SVM	0.93	0.92	0.93
[81]			FR, NFR	Ten features from un-processed dataset	Decision tree	0.9	0.9	0.9
[81]			FR, NFR	Ten features from processed dataset	Decision tree	0.94	0.95	0.94
[109]		Deep Learning	FR, NFR	fastText	CNN	0.96	0.92	0.94
[44]			FR, NFR	Word2vec	CNN	0.91	0.92	0.92
[30]			FR,NFR	-	BERT	0.92	0.92	0.92
Arabic model			FR, NFR	-	Bi-GRU	0.88	0.90	0.89
1) Functional Requirement(FR), Non-Functional Requirement(NFR) 2) CNN: convolutional neural network, SVM: Support Vector Machine, Bi-GRU: Bidirectional Gated Recurrent Unit, and BERT: Bidirectional Encoder Representations from Transformers								

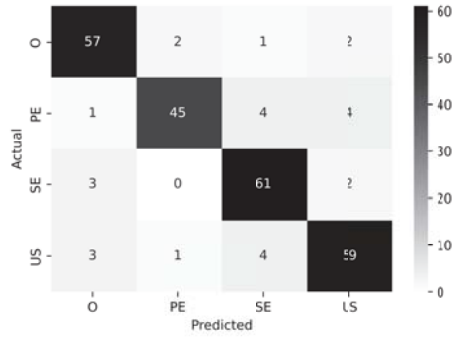
Table 6.7: Comparison of PROMISE binary classification

Model Name	Preprocessing	Accuracy	Precision	Recall	F-score
Word sequence	Without Punctuation	83.13	85.18	83.13	83.00
	Terminology Free	84.35	86.45	84.35	84.19
	Without Stopwords	85.53	88.72	85.53	85.55
	With Lemmatization	89.17	90.45	89.17	88.87
Character sequence	Without Punctuation	67.45	71.09	67.45	65.82
	Terminology Free	65.85	67.69	65.85	65.28
	Without Stopwords	65.82	68.62	65.82	65.23
	With Lemmatization	69.43	71.58	69.43	68.89

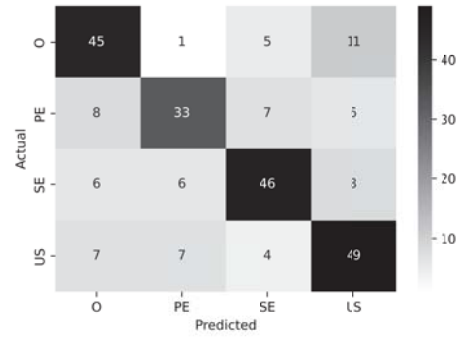
Table 6.8: Word and character four classes model results in En2ArPROMISE dataset

classes is 5 using word model, whereas the greatest number of misclassified samples in classifying performance requirements is 9.

Table 6.9 shows the results of ten different preprocessing on word and character models. The preprocessing tasks boost the F-score in word sequence by 5%. On the contrary, the character model has not been affected by the preprocessing steps.



((a)) Word model



((b)) Character model

Figure 6.6: Word and character Four classes models in En2ArPROMISE classification

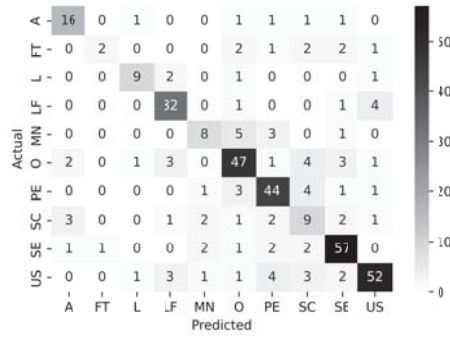
The high variation of classification results between word and character up to 17% in classifying multi-class problem.

Model Name	Preprocessing	Accuracy	Precision	Recall	F-score
Word sequence	Without Punctuation	67.46	69.04	67.46	65.89
	Terminology Free	66.91	70.67	66.91	66.45
	Without Stopwords	69.65	73.53	69.65	68.90
	With Lemmatization	74.80	75.28	74.80	73.18
Character sequence	Without Punctuation	55.27	57.84	55.27	54.32
	Terminology Free	55.55	56.72	55.55	53.86
	Without Stopwords	58.51	58.02	58.51	56.13
	With Lemmatization	56.61	57.13	56.61	55.21

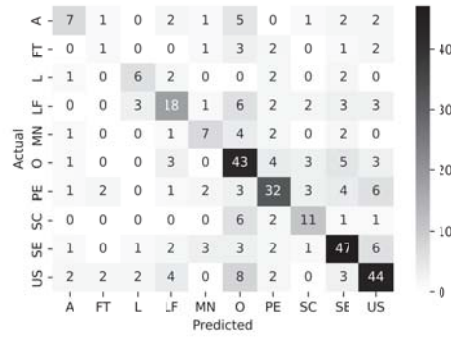
Table 6.9: Word and character Ten classes model results in En2ArPROMISE dataset

The confusion matrix of word and character models is shown in Figure 6.7. The samples of ten classes show different results and can be similar between word and character as maintainability class, or the character better than word as in scalability class, and the superiority of the word model in all remaining classes.

Classifying the requirements into several classes with limited number of samples is a challenging task. Writing the requirements in a very limited number of features complicates the extraction the features and classification process. For example, the operational requirement "The product shall be supported using the corporate sup-



((a)) Word model



((b)) Character model

Figure 6.7: Word and character ten classes models in En2ArPROMISE classification

port center" is misclassified to be usability requirement even after the translation and lemmatization. The using word استخدام has been used 25 times in the usability class while only 7 times in the operational class.

The proposed model is capable of detecting one of the requirement which is "The product shall protect the identity of the players. The product shall provide players no access to information that might reveal the identity of another player" that is translated to be "يجب أن يحمي المنتج هوية اللاعبين. يجب ألا يوفر المنتج للاعبين أي وصول إلى المعلومات التي قد تكشف عن هوية لاعب آخر". This is labeled in the original PROMISE dataset as performance but is more related to security requirement. This indicate the ability of the Arabic classifier to detect the outliers in the dataset after be trained.

The intersection between classes can be an issue in the annotation and the classification stages. For example, the "look and feel class" intersects with the usability class. Customizing the look means improving the usability of the application and sometimes improving the usability entitles some aspects of the system appearance. The look and feel requirement "The user interface shall have standard menus buttons for navigation" that is translated to be "يجب أن تحتوي واجهة المستخدم على أزرار قوائم قياسية للتنقل" is misclassified by the

model by label it as usability class.

Classifying the English PROMISE requirements into multi-classes either four or ten classes has been investigated in literature using traditional ML and deep learning techniques. Two studies in [10], [92] used SVM to classify the nonfunctional requirements into four classes (PE, US, SE, O). Their findings achieved an F-score up to 0.77. Applying traditional ML using Binarized Naïve Bayes to classify ten classes of nonfunctional requirements in a binarized way improved the F-score to be equal to 91% after processing and unifying some features in the dataset in [81].

Recently, several studies have proposed DL-based models to classify the requirements. The CNN technique have been used widely in [13], [15], [82], [109] to classify the requirements with excellent results. The RNN variants in [13] and ANN [82] had been applied with the bag of words or word2vec to represent the features and achieved results of F-score up to 0.85. Recent study have been published in [117] proposed model based on Bi-GRU approach to classify English requirements and scored 0.87 F-score in classifying the four classes. The highest F-score is 0.82 in classifying ten classes using BERT transformers in [30]. After translation, the model that are trained on EN2ArPROMISE scored comparable results in both four and ten classification. Table 6.10 shows a comparison of our findings in classifying En2ArPROMISE dataaet with studies in the literature classify PROMISE dataset.

Ref.	Type	Category	Names of classes ³	Features	Classifier ⁴	Recall	Precision	F1-score
[10]	Multiclass	Traditional ML	PE, US, SE, O	meta-data, lexical, and syntactical	SVM	0.81	0.78	0.77
[92]			PE, US, SE, O	Six different vectors	SVM, NB, LR, RF	≈ 0.67	≈ 0.77	≈ 0.68
[81]			SC, US, SE, A, MN, L, PE, FT, O, LF	Ten features from processed dataset	BNB	0.91	0.90	0.9
[82]		Deep Learning	MN, PE, US, SE, O	bag of words	ANN, CNN	0.83	0.86	0.85
[15]			F, SC, US, SE, A, MN, L, PE, FT, O, LF, PO	word2vec using skip-gram model	CNN	0.78	0.81	0.77
[13]			SC, US, SE, A, MN, L, PE, FT, O, LF	Word2vec	CNN, RNN variants	0.72	0.72	0.7
[109]			SC, US, SE, A, MN, L, PE, FT, O, LF	Word2vec	CNN	0.75	0.79	0.76
[30]			PE, US, SE, O	-	BERT	0.88	0.86	0.87
[30]			SC, US, SE, A, MN, L, PE, FT, O, LF	-	BERT	-	0.79	0.82
[117]			PE, US, SE, O	-	BiGRU			
[117]			SC, US, SE, A, MN, L, PE, FT, O, LF	-	BiGRU			
Arabic model			PE, US, SE, O	-	BiGRU	0.89	0.90	0.89
Arabic model			SC, US, SE, A, MN, L, PE, FT, O, LF	-	BiGRU	0.74	0.75	0.73
3) Functional (F), legal (L), maintainability (MN), availability (A), operational (O), look and feel (LF), fault tolerance (FT), performance (PE), usability (US), scalability (SC), and security (SE)								
4) BNB: Binarized Naive Bayes, NB: Naive Bayes, SVM: Support Vector Machine, RF: random forest, LR: Logistic Regression, CNN: convolutional neural network, ANN: Artificial Neural Network, BERT: Bidirectional Encoder Representations from Transformers, RNN: recurrent neural network, and BiGRU: Bidirectional Gated Recurrent Unit.								

Table 6.10: Comparison of PROMISE multi-class classification

Preprocessing level	Arabic length	English length	Arabic unique words	English unique words
Without Punctuation	89	124	40917	15635
Without Stopwords	75	76	40742	15605
With Lemmatization	75	76	14480	14156

Table 6.11: Sentence length and vocabulary size in ArNews and Ar2ENNews datasets

6.4.2 Impact of Translation on Classifying Arabic Software Requirements

The ultimate objective of this subsection is investigating the capability of classifying Arabic and English requirements. This investigation contains four dimensions that need to examine in Arabic and English requirements text: translation, preprocessing, dataset size, and domain type as shown in Figure 6.8.

Previously, the first two subsections represents that the comparable results can be achieved by translating the English SRS documents and used in the model training

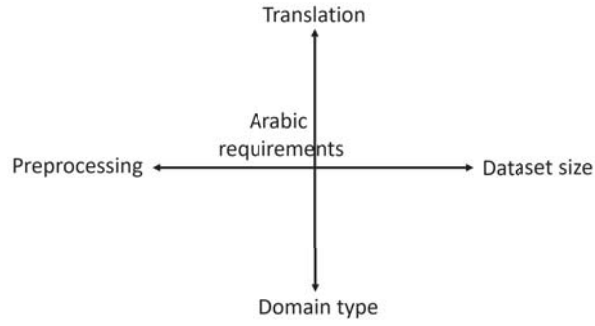


Figure 6.8: The four dimensions of conducting experiments on ArNews and Ar2EnNews

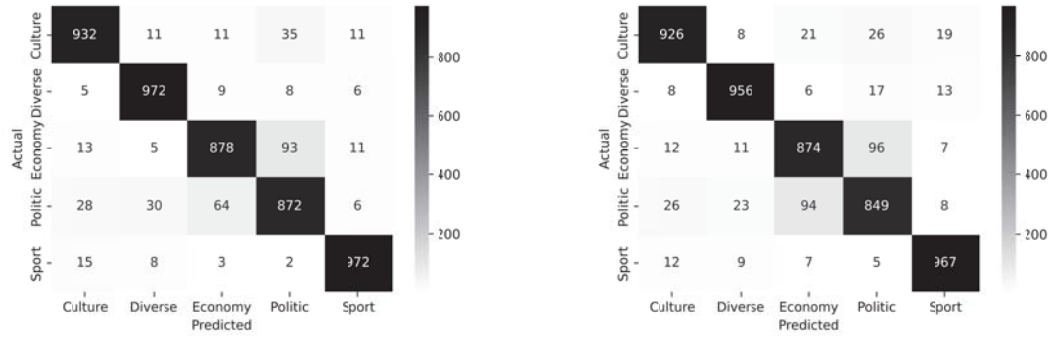
and validation. To investigate the validity of translating Arabic documents to English language and compare the results, we used the well-known dataset written in Arabic language. Since there is a lack of labelled Arabic SRS documents, we used a subset of the Arabic news dataset [111]. The best result using CNN techniques with the news dataset had been reported by the researchers in [111] is 92.94% when they used the whole dataset.

The proposed models show similar results in classifying both Arabic and English news datasets. The results show in Table 6.12 the preprocessing has limited impact on the datasets. The large number of misclassified samples between the economy and politics can be noticed in both Arabic and English confusion matrices in Figure 6.9.

Model Name	Preprocessing	Accuracy	Precision	Recall	F-score
Arabic	Without Punctuation	91.10	91.28	91.10	91.07
	Without Stopwords	91.12	91.27	91.12	91.11
	With Lemmatization	92.52	92.55	92.52	92.50
English	Without Punctuation	91.38	91.55	91.38	91.37
	Without Stopwords	91.10	91.17	91.10	91.08
	With Lemmatization	91.44	91.56	91.44	91.42

Table 6.12: Five news classification results in ArNews and Ar2EnNews datasets

The ultimate objective of this subsection is investigating the capability of classify-



((a)) Arabic model

((b)) English model

Figure 6.9: Arabic and English models in news classification

ing Arabic and English requirements. This investigation contains four dimensions that need to examine in Arabic and English requirements text: translation, cross-testing, and cross-learning as shown in Figure 6.10.

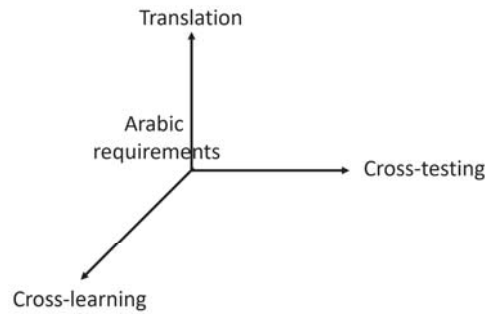


Figure 6.10: The three dimensions of conducting experiments on ArSRS and Ar2EnSRS

In all the previous three sections, the evaluation method is the cross validation. To answer the question about the capability of the proposed models trained on translated Arabic or English datasets, the new annotated dataset ArSRS has been used to evaluate the model that trained only on the PROMISE, trained on PROMISE and partial split of ArSRS, or trained on PROMISE and validated on partial split of ArSRS. All the previous scenarios have been used to develop an appropriate model to be tested

on unseen samples. The best result has been achieved in Arabic experiments is 76% while the best one in English experiments is 80% as shown in Table 6.13.

Language	Data split method	Accuracy	Precision	Recall	F-score
Arabic	Trained and validated on En2ArPROMISE and tested on ArSRS	67.22	80.64	64.76	71.83
	Trained and validated on (En2ArPROMISE + half of ArSRS) and tested on ArSRS	68.00	76.76	75.24	76.00
	Trained on En2ArPROMISE and validated on half of ArSRS and tested on half of ArSRS	68.66	82.14	68.31	74.59
English	Trained and validated on PROMISE and tested on Ar2EnSRS	65.62	80.36	67.87	73.59
	Trained and validated on (PROMISE + half of ArSRS) and tested on half of Ar2EnSRS	69.33	79.56	73.26	76.28
	Trained on PROMISE and validated on half of ArSRS and tested on half of Ar2EnSRS	72.66	78.84	81.18	80.00

Table 6.13: Word and character Ten classes model results

The functional requirement that is written in Arabic as follow: "يجب أن يدعم نظام إدارة السجلات توفر وسائل لتحديد السجلات" is classified to be non-functional requirements when translated to English because of the word *توفر* which is translated to English by "The records management system should support the **availability** of means to identify the records" that leads the model to classify the sentence as non-functional requirement of class availability.

On way to improve the results of the classification is minimizing the percentage

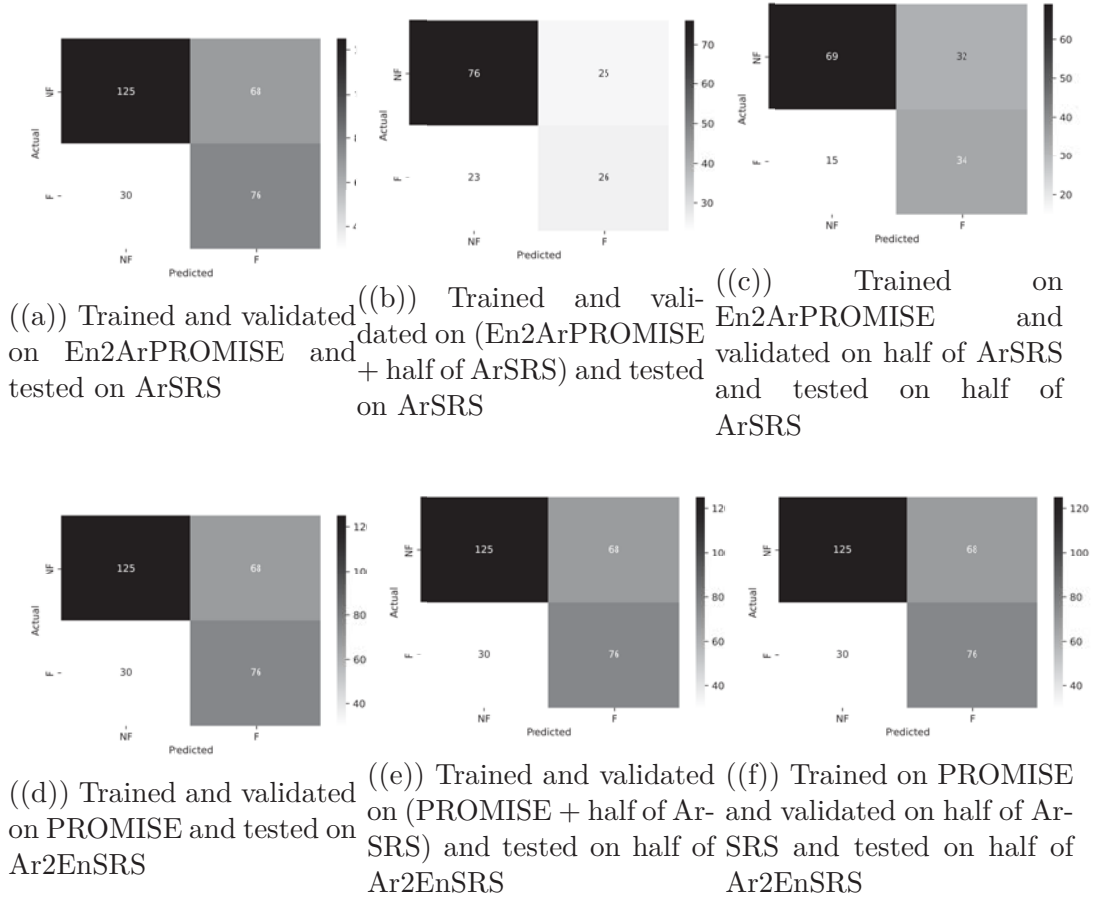


Figure 6.11: Confusion matrices of classifying ArSRS and Ar2EnSRS datasets

the unknown words by selecting a fraction of the dataset to be used in training or fine-tuning the proposed models. For example, the word **قصر** in the new dataset as in "قصر تعديل محتوى العناصر الوصفية للسجلات على المستخدمين المعتمدين ومديري النظام" have been used frequently with the non-functional requirements of type security and is not part of the Arabic PROMISE dataset.

6.4.3 Summary and Recommendations

Based on the previous experiments and results, the recommendation can be outline as follow:

- The best way to tackle the challenge of large vocabulary is using the lemmatization which minimizes the size to less than the half in some experiments and improves that F-score by 5%. The other preprocessing such as terminology removal or stop words removal minimize the length of the sentences without significantly improve the classification results or reduce the size of the vocabulary compared with lemmatization.
- The impact of preprocessing is varying based on two factors. The first one is the number of classes to be classified. In binary classification, the preprocessing has no impact on the classification results. The second factor is the size of the samples in the class that needs to be classified. Based on the experiments, The class that has enough number of samples requires less input preprocessing. One of the reasons behind that is the abundant features that can extracted from the inputs.
- The translation from English to Arabic increases the number of unique words which might affect the classification performance. On the contrary, the translation from Arabic to English text minimizes the number of unique features to be used in the classification. The minimal feature can be use as an indicator of the simplicity of the classification process.
- There is a lack of formal or semi-formal vocabulary in writing Arabic requirements which complicates the classification. The high variations of styles in writing the software requirements make the translation and classification harder. One way to mitigate this issue is using semi-formal style as in English which

facilitates the requirements classification by minimizing the variability in words and styles. Another way is rephrasing the requirements to have unified features as in Figure.

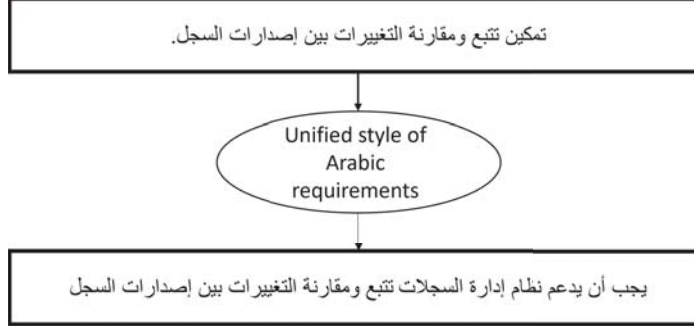


Figure 6.12: Semi-formal Arabic requirements style converter

- The translation process showed a subtle performance and promising results among the three different datasets. The translation a way of mitigating the costly labeling and utilizing the availability of different reliable tools. In addition, we are able achieve a higher results in English with limited processing steps compared with Arabic.

6.5 Threats to Validity

Because our study is sensitive to a variety of challenges to its validity, we employ a variety of mitigation techniques. Threats and mitigation are discussed in depth in this section.

Construct Validity: The quality of the evaluation depends on the quality of the manually annotated samples. The first two datasets have been validated in the literature while the the third dataset has been validated to minimize any negative impact.

The annotators were given several examples and definitions about the requirements classes. Both annotators are native Arabic speakers and they conducted some trials before the annotation to avoid any misunderstanding.

External Validity: Students' PROMISE projects are not indicative of all projects, particularly those from the business. The PROMISE dataset was chosen since it is the most often used in the literature and can be compared to published works in the literature. To mitigate this issue, we have selected other datasets such as news dataset and annotated Arabic software requirements.

Conclusion Validity: The conclusion has been drawn from experiments that can be reproduced by detailing all the factors in the conducted experiments with the accessible link. Some challenges have been tackled in the datasets such as the imbalance issue as in news dataset while the other datasets were used under-sampling and over-sampling techniques without any improvements.

6.6 Conclusions and Future Works

In this chapter, we experimented the capability of utilizing the models in rich-resources languages in classifying Arabic software requirements documents. The approach used automatic machine translation that has limited impact on the classification results. The comparable results between the English and Arabic achieved after conducting preprocessing steps in Arabic language such as lemmatization. The models have been evaluated on different datasets using different evaluation methods. The Arabic word-based models outperform the Arabic character-based models in all the experiments.

Using the translation mechanism, the classification of translated Arabic requirements to English can mitigate several challenges such as costly annotation and difficult processing in Arabic language. The lack of formal or semi-formal language in writing the Arabic software requirements complicate the classification and increase the vocabulary size of the dataset.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In this chapter, the thesis conclusions are discussed, an outlook on potential future work is presented, and the outcomes from this thesis are listed.

7.1 Conclusions

This dissertation presented an end-to-end approach to classify software requirements using raw word and character sequences employing BiGRUs. The proposed systems achieved state-of-the-art results on publicly available datasets. The models are evaluated on binary and multiclass classifications on the original PROMISE dataset and multilabel multiclass classification on the PROMISE ML dataset. In addition, the EHR dataset is used to evaluate the proposed model on multilabel binary and multilabel multiclass classification tasks. Two approaches were investigated in terms of input text tokenization: word sequences as tokens and character sequences as tokens.

The word sequence generally outperforms the character models in binary and multiclass classification. The performance of the character models is affected when the number of classes increases, while the performance of the word models maintains its classification ability. Additionally, the models based on BiGRU outperform most of the other deep learning or traditional machine learning techniques.

Data scarcity and imbalanced datasets are two main challenges in software requirements classification. Two approaches based on class and length were proposed to assign dynamic weights to every sample. The proposed method achieved better results than standard data augmentation. Using the general augmentation techniques without considering the domain characteristics did not achieve excellent results. We proposed domain-vocabulary and transformer-based augmentation techniques to address the domain-specific nature of the problem and prevent the distortion of the main features in the augmented sample. The proposed technique achieved a 3% improvement compared with standard data augmentation.

Finally, we experimented with utilizing the models in rich-resources languages to classify Arabic software requirements documents. The approach used automatic machine translation that has limited impact on the classification results. Comparable results between English and Arabic were achieved after preprocessing steps in the Arabic language, such as lemmatization. The models have been evaluated on different datasets using different evaluation splits. The Arabic word-based models outperform the Arabic character-based models in all the experiments. Using the translation mechanism, the classification of translated Arabic requirements to English can mitigate several challenges, such as costly annotation and difficult processing in Arabic.

7.2 Limitations

The lack of reporting and evaluation standards in research studies complicates the analysis and performance comparison. To mitigate this, we conducted several experiments containing various categories to compare with their work. Additionally, we used four metrics that provide insight into classifier performance. To facilitate research reproducibility, we fix the seed of the random number generators to be equal to 42, as in [27], in all the conducted experiments. The PROMISE projects written by students are not representative of all projects, especially projects from the industry. We used the PROMISE dataset because it is the most frequently used dataset in the literature and, thus, enables comparison with other approaches. The imbalanced dataset might have some impact on our findings, especially in the NFR subcategories, which suffer from variation in the number of samples.

The quality of the evaluation depends on the quality of the manually annotated samples. The annotators were given several examples and definitions of the requirements classes. Both annotators are native Arabic speakers and they conducted some trials before the annotation to avoid any misunderstanding. Manually building the dictionary can be one of the threats to validity. To minimize the impact, we include the minimal number of features with the minimal number of synonyms to preserve its generality while maintaining the domain-specific features and keywords. Students' PROMISE projects are not indicative of all projects, particularly those from the business. The PROMISE dataset was chosen since it is the most often used in the literature and can be compared to published works. We selected other datasets, such as

the Arabic news dataset and EHR, to mitigate this issue.

7.3 Future Directions

In future work, we plan to extend the PROMISE dataset by adding more NFR categories, such as safety, security, and confidentiality. Another direction is to use the proposed models to conduct several experiments on other industrial-scale datasets. Manual processing is generally used to extract requirements from a document. Automating the extraction and categorization of requirements from the original SRS can minimize the load on the user side. In future work, we plan to build cross-lingual word embedding for Arabic and English languages to improve the classification results in different languages. The lack of formal or semi-formal language in writing Arabic software requirements complicates the classification and increases the vocabulary size of the dataset.

7.4 Outcomes

This section presents the publications that have been emanated from this thesis.

- **Published Paper:** An end-to-end deep learning system for requirements classification using recurrent neural networks. Information and Software Technology
- **Under-review Paper:** Software Requirements Classification using Deep Learning Techniques: Trends and Opportunities. Journal paper.

- **Under-review Paper:** Software Requirement Classification: Translate or not to Translate. Journal paper.
- **In-progress Paper:** Augmentation-based Techniques in Software Requirements Classification. Journal paper

REFERENCES

- [1] D. of Statistics, “Modern approaches in natural language processing,” *https://compstat-lmu.github.io/seminar_nlp_ss20/*, 2020.
- [2] J. Wei and K. Zou, “Eda: Easy data augmentation techniques for boosting performance on text classification tasks,” *arXiv preprint arXiv:1901.11196*, 2019.
- [3] V. B. R. V. Sagar and S. Abirami, “Conceptual modeling of natural language functional requirements,” *Journal of Systems and Software*, vol. 88, pp. 25--41, 2014.
- [4] A. Rashwan, O. Ormandjieva, and R. Witte, “Ontology-based classification of non-functional requirements in software specifications: a new corpus and svm-based classifier,” in *2013 IEEE 37th Annual Computer Software and Applications Conference*. IEEE, 2013, pp. 381--386.
- [5] J. Cleland-Huang, R. Settini, X. Zou, and P. Solc, “Automated classification of non-functional requirements,” *Requirements engineering*, vol. 12, no. 2, pp. 103--120, 2007.

- [6] M. Glinz, “On non-functional requirements,” in *15th IEEE international requirements engineering conference (RE 2007)*. IEEE, 2007, pp. 21--26.
- [7] F.-L. Li, J. Horkoff, J. Mylopoulos, R. 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)*. IEEE, 2014, pp. 293--302.
- [8] M. Binkhonain and L. Zhao, “A review of machine learning algorithms for identification and classification of non-functional requirements,” *Expert Systems with Applications: X*, vol. 1, p. 100001, 2019.
- [9] I. Hussain, L. Kosseim, and O. Ormandjieva, “Using linguistic knowledge to classify non-functional requirements in srs documents,” in *International Conference on Application of Natural Language to Information Systems*. Springer, 2008, pp. 287--298.
- [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)*. Ieee, 2017, pp. 490--495.
- [11] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436--444, 2015.
- [12] L. Deng and D. Yu, “Deep learning: methods and applications,” *Foundations and trends in signal processing*, vol. 7, no. 3--4, pp. 197--387, 2014.

- [13] M. A. Rahman, M. A. Haque, M. N. A. Tawhid, and M. S. Siddik, "Classifying non-functional requirements using rnn variants for quality software development," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019, pp. 25--30.
- [14] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746--1751. [Online]. Available: <https://www.aclweb.org/anthology/D14-1181>
- [15] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, "Towards supporting software engineering using deep learning: A case of software requirements classification," in *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*. IEEE, 2017, pp. 116--120.
- [16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.
- [17] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
- [18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

- [19] B. Krawczyk, “Learning from imbalanced data: open challenges and future directions,” *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.
- [20] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, “Qanet: Combining local convolution with global self-attention for reading comprehension,” *arXiv preprint arXiv:1804.09541*, 2018.
- [21] Z. Xie, S. I. Wang, J. Li, D. Lévy, A. Nie, D. Jurafsky, and A. Y. Ng, “Data noising as smoothing in neural network language models,” *arXiv preprint arXiv:1703.02573*, 2017.
- [22] S. Kobayashi, “Contextual augmentation: Data augmentation by words with paradigmatic relations,” *arXiv preprint arXiv:1805.06201*, 2018.
- [23] A. Farghaly and K. Shaalan, “Arabic natural language processing: Challenges and solutions,” *ACM Transactions on Asian Language Information Processing (TALIP)*, vol. 8, no. 4, p. 14, 2009.
- [24] H. Al-Muhtaseb and C. Mellish, “Some differences between arabic and english: A step towards an arabic upper model,” in *6th International Conference on Multilingual Computing*, 1998.
- [25] T. Kudo and J. Richardson, “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing,” *arXiv preprint arXiv:1808.06226*, 2018.

- [26] T. Hiraoka, H. Shindo, and Y. Matsumoto, “Stochastic tokenization with a language model for neural text classification,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 1620--1629.
- [27] 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)*. IEEE, 2019, pp. 142--152.
- [28] P. R. Anish, A. Sainani, A. Ahmed, and S. Ghaisas, “Implementation-centric classification of business rules from documents,” in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2019, pp. 227--233.
- [29] A. Sainani, P. R. Anish, V. Joshi, and S. Ghaisas, “Extracting and classifying requirements from software engineering contracts,” in *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 2020, pp. 147--157.
- [30] T. Hey, J. Keim, A. Koziolk, and W. F. Tichy, “Norbert: Transfer learning for requirements classification,” in *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 2020, pp. 169--179.
- [31] R. Chatterjee, A. Ahmed, and P. R. Anish, “Identification and classification of architecturally significant functional requirements,” in *2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. IEEE, 2020, pp. 9--17.

- [32] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2016, pp. 39--45.
- [33] ISO/IEC/IEEE, "Systems and software engineering life cycle processes requirements engineering," 2018.
- [34] M. O. Hegazi, Y. Al-Dossari, A. Al-Yahy, A. Al-Sumari, and A. Hilal, "Preprocessing arabic text on social media," *Heliyon*, vol. 7, no. 2, p. e06191, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405844021002966>
- [35] I. ISO, "Ieee 12207-2008-iso/iec/ieee international standard-systems and software engineering--software life cycle processes," 2018.
- [36] S. Fakhoury, V. Arnaoudova, C. Noiseux, F. Khomh, and G. Antoniol, "Keep it simple: Is deep learning good for linguistic smell detection?" in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 602--611.
- [37] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 3--14.
- [38] A. Ahmad, C. Feng, M. Khan, A. Khan, A. Ullah, S. Nazir, and A. Tahir, "A systematic literature review on using machine learning algorithms for soft-

- ware requirements identification on stack overflow,” *Security and Communication Networks*, vol. 2020, 2020.
- [39] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, “Natural language processing (nlp) for requirements engineering: A systematic mapping study,” *arXiv preprint arXiv:2004.01099*, 2020.
- [40] A. F. Del Carpio and L. B. Angarita, “Trends in software engineering processes using deep learning: A systematic literature review,” in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2020, pp. 445–454.
- [41] B. Kitchenham, “Procedures for performing systematic reviews,” *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.
- [42] S. Keele *et al.*, “Guidelines for performing systematic literature reviews in software engineering,” Citeseer, Tech. Rep., 2007.
- [43] C. Wohlin, “Guidelines for snowballing in systematic literature studies and a replication in software engineering,” in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1–10.
- [44] A. Dekhtyar and V. Fong, “Re data challenge: Requirements identification with word2vec and tensorflow,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 2017, pp. 484–489.

- [45] J. Slankas and L. Williams, “Automated extraction of non-functional requirements in available documentation,” in *2013 1st International workshop on natural language analysis in software engineering (NaturaLiSE)*. IEEE, 2013, pp. 9--16.
- [46] M. Riaz, J. King, J. Slankas, and L. Williams, “Hidden in plain sight: Automatically identifying security requirements from natural language artifacts,” in *2014 IEEE 22nd international requirements engineering conference (RE)*. IEEE, 2014, pp. 183--192.
- [47] A. Mahmoud and G. Williams, “Detecting, classifying, and tracing non-functional software requirements,” *Requirements Engineering*, vol. 21, no. 3, pp. 357--381, 2016.
- [48] A. Ferrari, G. O. Spagnolo, and S. Gnesi, “Pure: A dataset of public requirements documents,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 2017, pp. 502--505.
- [49] J. Fischbach, B. Hauptmann, L. Konwitschny, D. Spies, and A. Vogelsang, “Towards causality extraction from requirements,” in *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 2020, pp. 388--393.
- [50] A. Casamayor, D. Godoy, and M. Campo, “Identification of non-functional requirements in textual specifications: A semi-supervised learning approach,” *Information and Software Technology*, vol. 52, no. 4, pp. 436--445, 2010.

- [51] P. Singh, D. Singh, and A. Sharma, “Rule-based system for automated classification of non-functional requirements from requirement specifications,” in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2016, pp. 620--626.
- [52] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, “Automated classification of non-functional requirements,” *Requirements engineering*, vol. 12, no. 2, pp. 103--120, 2007.
- [53] 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)*. Ieee, 2017, pp. 490--495.
- [54] 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)*. IEEE, 2017, pp. 496--501.
- [55] S. Liu, R. Zeng, and X. He, “An empirical study on classification of non-functional requirements.” in *SEKE*, 2011, pp. 444--449.
- [56] T. H. Nguyen, J. Grundy, and M. Almorsy, “Rule-based extraction of goal-use case models from text,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 591--601.

- [57] R. Jindal, R. Malhotra, and A. Jain, “Automated classification of security requirements,” in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2016, pp. 2027--2033.
- [58] R. Malhotra, A. Chug, A. Hayrapetian, and R. Raje, “Analyzing and evaluating security features in software requirements,” in *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*. IEEE, 2016, pp. 26--30.
- [59] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens, “Supporting requirements engineers in recognising security issues,” in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2011, pp. 4--18.
- [60] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, “Pre-trained models for natural language processing: A survey,” *Science China Technological Sciences*, pp. 1--26, 2020.
- [61] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532--1543.
- [62] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of tricks for efficient text classification,” *arXiv preprint arXiv:1607.01759*, 2016.

- [63] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, “Deep contextualized word representations,” *arXiv preprint arXiv:1802.05365*, 2018.
- [64] K. He, R. Girshick, and P. Dollár, “Rethinking imagenet pre-training,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4918--4927.
- [65] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533--536, 1986.
- [66] P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu, “Text classification improved by integrating bidirectional lstm with two-dimensional max pooling,” *arXiv preprint arXiv:1611.06639*, 2016.
- [67] J. Schmidhuber, “Long short-term memory: Tutorial on lstm recurrent networks,” 2003.
- [68] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” *arXiv preprint arXiv:1409.1259*, 2014.
- [69] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673--2681, 1997.
- [70] G. Liu and J. Guo, “Bidirectional lstm with attention mechanism and convolutional layer for text classification,” *Neurocomputing*, vol. 337, pp. 325--338, 2019.

- [71] Chapter 4 recurrent neural networks and their applications in nlp. [Online]. Available: https://compstat-lmu.github.io/seminar_nlp-ss20/recurrent\neural-networks-and-their-applications-in-nlp.html
- [72] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization.” *Journal of machine learning research*, vol. 13, no. 2, 2012.
- [73] X. Liu and A. Wangperawong, “Transfer learning robustness in multi-class categorization by fine-tuning pre-trained contextualized language models,” *arXiv preprint arXiv:1909.03564*, 2019.
- [74] M. U. Salur and I. Aydin, “A novel hybrid deep learning model for sentiment classification,” *IEEE Access*, vol. 8, pp. 58 080--58 093, 2020.
- [75] Y. Zhang and B. Wallace, “A sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification,” *arXiv preprint arXiv:1510.03820*, 2015.
- [76] J. Howard and S. Ruder, “Universal language model fine-tuning for text classification,” *arXiv preprint arXiv:1801.06146*, 2018.
- [77] G. Pruthi, F. Liu, S. Kale, and M. Sundararajan, “Estimating training data influence by tracing gradient descent,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.
- [78] A. Rios and R. Kavuluru, “Few-shot and zero-shot multi-label learning for structured label spaces,” in *Proceedings of the Conference on Empirical Methods in*

Natural Language Processing. Conference on Empirical Methods in Natural Language Processing, vol. 2018. NIH Public Access, 2018, p. 3132.

- [79] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, “The detection and classification of non-functional requirements with application to early aspects,” in *14th IEEE International Requirements Engineering Conference (RE’06)*. IEEE, 2006, pp. 39--48.
- [80] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie, “Automated extraction of security policies from natural-language software documents,” in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 2012, pp. 1--11.
- [81] 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)*. IEEE, 2017, pp. 496--501.
- [82] 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. IEEE, 2019, pp. 610--615.
- [83] M. Abdul-Mageed, M. Diab, and M. Korayem, “Subjectivity and sentiment analysis of modern standard arabic,” in *Proceedings of the 49th Annual Meeting of*

- the Association for Computational Linguistics: Human Language Technologies*, 2011, pp. 587--591.
- [84] S. R. El-Beltagy, M. El Kalamawy, and A. B. Soliman, "Niletmrg at semeval-2017 task 4: Arabic sentiment analysis," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017, pp. 790--795.
- [85] A. M. Alayba, V. Palade, M. England, and R. Iqbal, "A combined cnn and lstm model for arabic sentiment analysis," in *International cross-domain conference for machine learning and knowledge extraction*. Springer, 2018, pp. 179--191.
- [86] M. Abdul-Mageed, "Modeling arabic subjectivity and sentiment in lexical space," *Information Processing & Management*, vol. 56, no. 2, pp. 291--307, 2019.
- [87] I. A. Farha and W. Magdy, "A comparative study of effective approaches for arabic sentiment analysis," *Information Processing & Management*, vol. 58, no. 2, p. 102438, 2021.
- [88] W. Antoun, F. Baly, and H. Hajj, "Arabert: Transformer-based model for arabic language understanding," *arXiv preprint arXiv:2003.00104*, 2020.
- [89] R. Mihalcea, C. Banea, and J. Wiebe, "Learning multilingual subjective language via cross-lingual projections," in *Proceedings of the 45th annual meeting of the association of computational linguistics*, 2007, pp. 976--983.

- [90] A. Balahur and M. Turchi, “Comparative experiments using supervised learning and machine translation for multilingual sentiment analysis,” *Computer Speech & Language*, vol. 28, no. 1, pp. 56--75, 2014.
- [91] S. M. Mohammad, M. Salameh, and S. Kiritchenko, “How translation alters sentiment,” *Journal of Artificial Intelligence Research*, vol. 55, pp. 95--130, 2016.
- [92] S. Amasaki and P. Leelaprute, “The effects of vectorization methods on non-functional requirements classification,” in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 175--182.
- [93] P. Singh, D. Singh, and A. Sharma, “Rule-based system for automated classification of non-functional requirements from requirement specifications,” in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2016, pp. 620--626.
- [94] N. T. Alhindawi, “Information retrieval-based solution for software requirements classification and mapping,” in *2018 5th International Conference on Mathematics and Computers in Sciences and Industry (MCSI)*. IEEE, 2018, pp. 147--154.
- [95] M. Younas, K. Wakil, M. Arif, and A. Mustafa, “An automated approach for identification of non-functional requirements using word2vec model,” *Int. J. Adv. Comput. Sci. Appl*, vol. 10, 2019.
- [96] F.-L. Li, J. Horkoff, J. Mylopoulos, R. 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)*.
IEEE, 2014, pp. 293--302.
- [97] A. Rashwan, O. Ormandjieva, and R. Witte, "Ontology-based classification of non-functional requirements in software specifications: a new corpus and svm-based classifier," in *2013 IEEE 37th Annual Computer Software and Applications Conference*. IEEE, 2013, pp. 381--386.
- [98] D. A. Ramadhani, S. Rochimah, and U. L. Yuhana, "Classification of non-functional requirements using semantic-fsknn based iso/iec 9126," *Telkomnika*, vol. 13, no. 4, p. 1456, 2015.
- [99] M. Arsenovic, M. Karanovic, S. Sladojevic, A. Anderla, and D. Stefanovic, "Solving current limitations of deep learning based approaches for plant disease detection," *Symmetry*, vol. 11, no. 7, p. 939, 2019.
- [100] J. K. Quijas, *Analysing the effects of data augmentation and free parameters for text classification with recurrent convolutional neural networks*. The University of Texas at El Paso, 2017.
- [101] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *Advances in neural information processing systems*, vol. 28, 2015.
- [102] N. T. Vu, H. Adel, P. Gupta, and H. Schütze, "Combining recurrent and convolutional neural networks for relation classification," *arXiv preprint arXiv:1605.07333*, 2016.

- [103] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735--1780, 1997.
- [104] B. Liu, Y. Zhou, and W. Sun, “Character-level text classification via convolutional neural network and gated recurrent unit,” *International Journal of Machine Learning and Cybernetics*, vol. 11, no. 8, pp. 1939--1949, 2020.
- [105] A. Chakrabarty, O. A. Pandit, and U. Garain, “Context sensitive lemmatization using two successive bidirectional gated recurrent networks,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 1481--1491.
- [106] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *arXiv preprint arXiv:1310.4546*, 2013.
- [107] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [108] G. Tsoumakas, I. Katakis, and I. Vlahavas, “Mining multi-label data,” in *Data mining and knowledge discovery handbook*. Springer, 2009, pp. 667--685.
- [109] V. L. Fong, “Software requirements classification using word embeddings and convolutional neural networks,” 2018.
- [110] C. Fellbaum, “A semantic network of english: the mother of all wordnets,” in *EuroWordNet: A multilingual database with lexical semantic networks*. Springer, 1998, pp. 137--148.

- [111] S. Boukil, M. Biniz, F. El Adnani, L. Cherrat, and A. E. El Moutaouakkil, "Arabic text classification using deep learning technics," *International Journal of Grid and Distributed Computing*, vol. 11, no. 9, pp. 103--114, 2018.
- [112] K. Krippendorff, *Content analysis: An introduction to its methodology*. Sage publications, 2018.
- [113] —, "Reliability in content analysis: Some common misconceptions and recommendations," *Human communication research*, vol. 30, no. 3, pp. 411--433, 2004.
- [114] T. Zerrouki, "pyarabic, an arabic language library for python," 2010.
- [115] H. Mubarak, "Build fast and accurate lemmatization for arabic," *arXiv preprint arXiv:1710.06700*, 2017.
- [116] J. P. Winkler, J. Grönberg, and A. Vogelsang, "Optimizing for recall in automatic requirements classification: An empirical study," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 2019, pp. 40--50.
- [117] O. AlDhafer, I. Ahmad, and S. Mahmood, "An end-to-end deep learning system for requirements classification using recurrent neural networks," *Information and Software Technology*, p. 106877, 2022.

VITAE

- **Name:** Osamah Mohammed Qaed Aldhafer
- **Nationality:** Yemeni
- **Date of Birth:** 10.12.1988
- **Email:** *aldafeer@gmail.com*
- **Permanent Address:** Saudi Arabia - Medina
- **EDUCATION**
- **RESEARCH INTEREST**
- **COMPUTER AND NETWORKING SKILLS**
 - **Networking:** Extensive knowledge of data communications/networking, routing protocols, CNN/Deep Learning (DL), LANs, WANs, Bidirectional Encoder Representations from Transformers (BERT), and Recurrent Neural Networks (RNN).

2016 - 2022	Ph.D. , Information & Computer Science, (May, 2022) Faculty of Computer Science and Engineering King Fahd University of Petroleum and Minerals (KFUPM), KSA Thesis Title : "Software Requirements Classification using Deep Learning: Dealing with Limited Training Data"
2013 - 2016	M.Sc. , Information & Computer Science, (May, 2016) Faculty of Computer Science and Engineering King Fahd University of Petroleum and Minerals (KFUPM), KSA Thesis Title: "Models for Conducting Progress and Milestone Reviews: Implementation of Cmmi Level 2 Specific Practices for Small and Medium Sized Software Development Organizations"
2007 - 2012	B.Sc. , Computer, (May, 2007), KKU KSA
2006	Secondary School , Scientific Major Abdulaziz Alrabea, Medina, Saudi Arabia

- **Simulation and emulation tools:** Machine Learning (ML), and COOJA.
- **Data Analytics and Visualization Tools:** GNU PLOT, Matplotlib, and SPSS.
- **Programming Languages:** Matlab, C/C++, Bash Programming, Python, and Java.
- **IDEs:** Eclipse, NetBeans, JDeveloper, and Visual Studio.NET, and TeXstudio.
- **Operating Systems:** Linux, MS-DOS, and Microsoft Windows, and Virtualization of Windows using VirtualBox, and VMWare
- Excellent command of Latex environment.

