

Automated Software Requirements Prioritization using Natural Language Processing



By

Israr Ahmad

(Registration No.: 00000317474)

Supervisor: Dr. Wasi Haider Butt

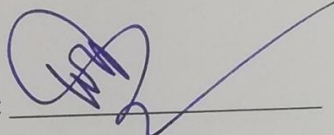
DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

September 2023

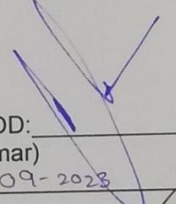
THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by NS **Israr Ahmad** Registration No. **00000317474**, of College of E&ME has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the thesis.

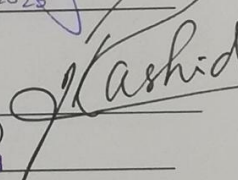
Signature :  _____

Name of Supervisor: **Dr Wasi Haider Butt**

Date: 20-09-2023

Signature of HOD:  _____
(Dr Usman Qamar)

Date: 20-09-2023

Signature of Dean:  _____
(Brig Dr Nasir Rashid)

Date: 20 SEP 2023

Dedicated to my exceptional parents and adored brothers whose tremendous support and cooperation led me to this accomplishment.

Acknowledgements

All praise and glory to Almighty Allah (the most glorified, the most high) who gave me the courage, patience, knowledge and ability to carry out this work and to persevere and complete it satisfactorily. Undoubtedly, HE eased my way and without HIS blessings I can achieve nothing.

I would like to express my sincere gratitude to my advisor Dr Wasi Haider Butt for boosting my morale and for his continual assistance, motivation, dedication and invaluable guidance in my quest for knowledge. I am blessed to have such a co-operative advisor and kind mentor for my research.

Along with my advisor, I would like to acknowledge my entire thesis committee: Dr. Arslan Shaukat and Dr Farooq e Azam for their cooperation and prudent suggestions.

My acknowledgement would be incomplete without thanking the biggest source of my strength, my family. I am profusely thankful to my beloved mother and father (late) who raised me when I was not capable of walking and continued to support me throughout every department of my life and my loving brothers who were with me through my thick and thin.

Finally, I would like to express my gratitude to all my friends and the individuals who have encouraged and supported me through this entire period.

Abstract

The software requirements specifications (SRS) may become a barrier to the successful completion of the project if they are written in a language that is difficult to understand. In certain situations, they cause failure to meet the actual requirements. The SRS dataset may contain redundant information or material that is disputed, either of which might result in higher expenditures and a loss of time, diminishing the overall efficiency of the project. The current developments in machine learning have led to a rise in the amount of work being put towards the development of automated solutions for the creation of a seamless software requirements specification (SRS). In this study, we employ the transformer models, including BERT and RoBERTa for classification. We focus on analyzing RoBERTa capacity for multi-class text classification tasks that involve predicting the type, priority, and severity of the requirements specified by the users. Moreover we compare its performance to that of other deep learning methods like LSTM and BiLSTM. We tested the performance of these models on the DOORS dataset. We have also compared the proposed model. We achieved higher accuracy i.e., '84.7%', sensitivity, precision, recall and F1 score by using RoBERTa and compared our results with existing approaches.

Keywords: NLP, Text classification, Software requirement specification (SRS), RoBERTa, Deep learning, Transformers

Table of Contents

ACKNOWLEDGEMENTS.....	III
ABSTRACT.....	IV
1. INTRODUCTION.....	1
1.1 OVERVIEW AND BACKGROUND	1
1.2 ROLE OF NLP IN SOFTWARE REQUIREMENT PRIORITIZATION	4
1.3 MOTIVATION	5
1.4 PROBLEM STATEMENT	6
1.5 AIMS AND OBJECTIVES	6
1.6 THESIS OUTLINE	7
2 LITERATURE REVIEW	8
2.1 OVERVIEW AND MAJOR OUTCOMES OF SLR	8
2.2 COMPARISON OF LITERATURE REVIEW	12
2.3 RESEARCH QUESTIONS	14
2.4 RESEARCH GAP	14
3 PROPOSED APPROACH	15
3.1 DATASET	15
3.2 CLASS DISTRIBUTION OF CATEGORICAL FEATURES IN THE DOORS SRS DATASET.....	16
3.3 DATA PRE-PROCESSING	17
3.4 EXPERIMENTAL SETUP SETTINGS	17
3.5 NLP MODELS FOR CLASSIFICATION.....	20
4.1 EVALUATION METHODOLOGY	21
4.2 EVALUATION PARAMETERS	21
4.3 PROPOSED APPROACH	22
4.4 OPTIMIZERS	23
4.5 ROBERTA	23
4.6 RESULTS OF NLP MODELS.....	24
4.7 COMPARISON BETWEEN PROPOSED APPROACH AND EXISTING APPROACHES.....	26
5 CONCLUSION & FUTURE WORK	28
5.1 CONCLUSION.....	28

5.2	FUTURE WORK.....	28
	REFERENCES.....	I

List of Figures

Figure 1 Block NLP process pipeline retrieve the information from text.....	2
Figure 2 Block Diagram of Proposed method.....	15
Figure 3. Class distribution for the each class level [2].....	16
Figure 4. Implementation and importing libraries using python language.....	22
Fig5. Fine Tuning the Model.....	22
Fig6. Fine-tuning and adjusting hyper parameters.....	22
Fig7: Training the Data and calculating the loss.....	23
Fig8. Tested the accuracy.....	23
Figure 9 Architecture Diagram of RoBERTa [20].....	23
Figure 10 Results comparison of proposed model.....	24
Figure 11 Results comparison of priority in DOORs Dataset.....	25
Figure 12 Results comparison of Severity in DOORs Dataset.....	26

List of Tables

Table 1 Comparison table of previous studies.....	12
Table 2 Classification Results of proposed models	24
Table 3 Results of Priority.....	25
Table 4 Results of Severity.....	26
Table 5 Comparison between Proposed approach and existing approach	27

1. INTRODUCTION

1.1 Overview and background

Software Requirements Specifications, or SRS for short, play an important part in the Software Development Life Cycle (SDLC) since they act as a tool to convey user requirements to software developers and other stakeholders. This is an important part of the SDLC. SRS, or Software Requirements Specifications, are written papers that outline the primary characteristics, limitations, and capabilities of a software product [1]. These papers have to be created in accordance with the standards that have been specified in order for all of the stakeholders, including users, analysts, and developers, to have the same understanding of what the specifications represent. At the conclusion of the project, SRS may also serve as indicators for assessing the quality and acceptance of the product or process. The degree to which the completed program me follows the SRS documentation is one of the factors that determines whether or not a software project was successful. Statements that are susceptible to being misunderstood, explanations that are vague, or conclusions that are imprecise might potentially lead to a catastrophic failure later on in the project [2]. In light of this, it is necessary to make consistent use of the contextual terminology that is relevant to that particular domain in order to get a clear and consistent comprehension of these requirements. In addition, the success of succeeding phases in the SDLC process is dependent on well specified requirements and the precise implementation of those needs; failing to do so may result in delays as well as additional expenditures [3]. Studies on the feature extraction from existing systems mostly use the source code as the object or input of the extraction process [4]. While other studies also conducted to use models including a class diagram and use case diagram as the objects for the extraction process [5] [6].

However, most software developer only measures their product quality on the released software product or the implementation result regardless of the original requirement

[7].Therefore, software feature extraction from specification document is more suitable based on the software engineering perspective rather than model or source code to acquire the more valid feature. This is because the specification document is the basis of the validation and verification of system functionality in the software development process [8].

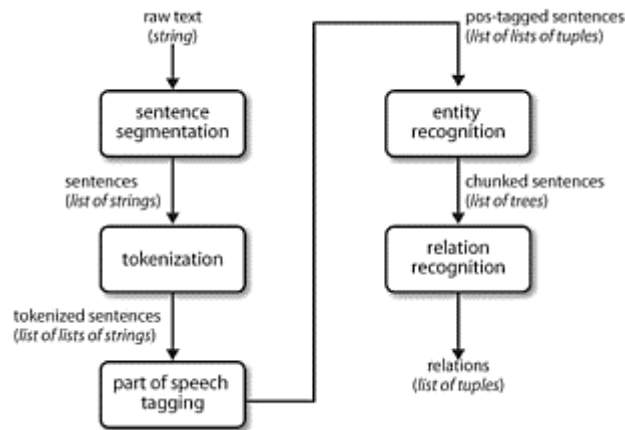


Figure 1 Block NLP process pipeline retrieve the information from text

Automated software requirements prioritization techniques have gained significant attention in the field of software engineering, aiming to enhance the efficiency and effectiveness of software development processes. One such approach that has emerged is the utilization of Natural Language Processing (NLP) to automate the prioritization of software requirements. By harnessing the power of NLP, this technique enables the automated analysis and understanding of natural language requirements documents. Traditionally, requirements prioritization has relied on manual effort, which can be time-consuming, subjective, and prone to human biases. However, with the advent of NLP, software engineers can now leverage advanced algorithms and linguistic models to extract relevant information and prioritize requirements in a more objective and efficient manner [11]. These models can be trained using historical data, expert knowledge, or a combination of both. The prioritization models learn from past prioritization decisions made by software engineers, considering factors such as customer needs, project constraints, and business objectives. The benefits of this automated approach are manifold.

- Firstly, it reduces the manual effort required for requirements prioritization, freeing up valuable time for software engineers to focus on other critical tasks.

- Secondly, it helps eliminate subjective biases by providing an objective and data-driven prioritization process. Additionally, by automating the prioritization process, it enhances consistency and reproducibility, ensuring that similar requirements are consistently ranked across different projects [12].

However, it is essential to acknowledge some of the challenges associated with this technique.

- NLP algorithms heavily rely on the quality of the requirements documents, which may contain ambiguities, inconsistencies, or incomplete information.
- Noise reduction techniques and domain-specific customization are often employed to mitigate these challenges.
- Additionally, ongoing monitoring and periodic updating of the prioritization models are necessary to adapt to evolving project requirements and changes in stakeholder preferences.
- The automated software requirements prioritization technique using NLP presents a promising approach to enhance the efficiency and accuracy of prioritizing software requirements.
- By leveraging NLP algorithms and linguistic models, software engineers can streamline the prioritization process, reduce manual effort, and make more informed decisions.

While challenges exist, ongoing advancements in NLP and machine learning continue to pave the way for more sophisticated and reliable automated prioritization techniques in the realm of software engineering [13]. We have used NLP models .The suggested model was developed through the examination of DOORS dataset which contains SRS. The proposed work would focus on improving the recognition accuracy by addressing the problem of software requirement prioritization.

1.2 Role of NLP in Software Requirement prioritization

The use of automated software needs prioritization approaches that make use of natural language processing (NLP) is very necessary in order to achieve both increased productivity and precision in the prioritization process. NLP is a collection of tools and methods that enable software engineers to automatically analyses, comprehend, and extract useful information from natural language requirement documents. NLP is provided via natural language processing (NLP). The following is a list of some of the particular functions that NLP plays in the process of automating software needs prioritization:

- **Processing of Text:** Natural Language Processing (NLP) methods are utilized in order for processing and parse the requirements documents. This requires performing activities like as tokenization, syntactic parsing, and part-of-speech tagging, all of which assist in separating the text into meaningful pieces and comprehending the grammatical structure of sentences. The processing of text establishes the groundwork for further analysis as well as the extraction of information. NLP algorithms are able to recognize and extract named entities from the requirements documents. This is referred to as "named entity recognition." Recognizing objects such as product names, organizations, places, dates, and other pertinent information is included in this. Named Entity Recognition is helpful in understanding the context as well as the dependencies that exist between the various criteria. This is something that is often essential for achieving appropriate prioritization.
- **Analysis of Sentiment:** Natural Language Processing (NLP) is able to ascertain the feeling or opinion communicated in the requirements documents by applying several approaches from the field of analysis of sentiment. This can be helpful in determining if certain criteria are related with good or negative attitudes, assisting in the prioritization of those requirements that are essential for the satisfaction of customers, or addressing possible dangers.
- **Extraction of Dependencies and linkages:** NLP algorithms may examine the requirements to determine the dependencies and linkages between the various needs. This assists in identifying needs that are connected to one another or have constraints

on the specifications of other requirements. The prioritization approach can guarantee that needs with dependencies on other high-priority objectives are likewise given adequate priority by taking into consideration these dependencies and ensuring that they are given appropriate priority.

- **Extraction of Features:** Natural Language Processing methods make it easier to extract significant features or keywords from a set of requirements papers. These qualities may consist of specialized terminology, keywords that are exclusive to a domain, or any number of other pertinent indications of relevance. The creation of a representation of each need that can be used by machine learning models for the purpose of prioritization is facilitated by feature extraction.
- **Models of Automated Prioritization that Are Driven by Machine Learning:** Natural Language Processing (NLP) is an extremely important component in the process of training machine learning models. NLP facilitates the building of models that are able to learn from previous judgements and assign priority levels to needs. These models are created by integrating the extracted characteristics with contextual information and historical prioritization data. For the purpose of making informed prioritization decisions, these models can take into consideration a variety of criteria, including the preferences of stakeholders, the limitations of the project, and the aims of the organization.

1.3 Motivation

The motivation of research is the moment, there are few limitations in published researches on feature extraction from natural language documents, i.e. unavailable tools for evaluation, restricted or limited input, irrelevant feature naming, non-reproducible result, and domain engineer intervention in the process [2]. While this research is aimed to produce a tool for automatically extracting software features directly from SRS documents without any human intervention in the process. The tool will be applied and tested using selected SRS from the Public Requirement Engineering (DOORs) dataset [3] to justify its correctness.

1.4 Problem Statement

Automated software requirements prioritization techniques have gained significant attention in the field of software engineering, aiming to enhance the efficiency and effectiveness of software development processes. One such approach that has emerged is the utilization of Natural Language Processing (NLP) to automate the prioritization of software requirements. By harnessing the power of NLP, this technique enables the automated analysis and understanding of natural language requirements documents. Traditionally, requirements prioritization has relied on manual effort, which can be time-consuming, subjective, and prone to human biases. However, with the advent of NLP, software engineers can now leverage advanced algorithms and linguistic models to extract relevant information and prioritize requirements in a more objective and efficient manner [11]. While challenges exist, ongoing advancements in NLP and machine learning continue to pave the way for more sophisticated and reliable automated prioritization techniques in the realm of software engineering [13]. We have used NLP models. The suggested model was developed through the examination of DOORs dataset which contains SRS. The proposed work would focus on improving the recognition accuracy by addressing the problem of software requirement prioritization.

1.5 Aims and Objectives

The major objectives of the research are as follows:

- To perform experiment using NLP tools/techniques used in requirements engineering for automated requirement prioritization.
- To obtain good results on NLP already used in requirement prioritization
- To explore algorithms that are used particularly for the requirement prioritization technique.
- To propose approach to achieved higher accuracy and compared our results with existing approaches to show the effectiveness of our approach.
- Analyze the results obtain via the new approach and compare it with previous results to any significance change.

1.6 Thesis Outline

This remaining work is structured as follows:

Chapter 2 stated a literature review in detail and the important relevant work performed by analysts and researchers in the previous few years, which covers the basics and background of the ambiguity detection and NLP approaches usage for the analyze of requirements. The systematic literature review is composed of three core sections. The ever first is the review-protocol which displays detail upon the procedure using which the literature-review has carried out. Second Section offers detail on research study carried out on this area in the form of research-questions and tables. And section three shows the research-gap that were encountered in the study.

Chapter 3 consist of the proposed approach in detail. It discusses the method in terms of an overview of the algorithm, main components of the approach and depiction of solution.

Chapter 4 includes implementation, validation, and discussion on results together with research-question and related figures. It also makes detail to the assessment of our work with the state of the art. Moreover, it precisely describes the limitations of this study.

Chapter 5 conclude the research thesis and reveals the future work of this research.

2 LITERATURE REVIEW

This chapter covers the systematic literature review for the area of this research. The Chapter comprises of an overview and major outcome of Systematic literature, contribution of literature review, review methodology, research questions, category definitions, review protocol of literature review, results and analysis, answers to the research questions for literature and conclusion of SLR.

2.1 Overview and Major Outcomes of SLR

The research that goes into natural language processing (NLP) entails applying concepts and strategies from a variety of fields (such as computer science, artificial intelligence, and computational linguistics) in order to develop automated methods of analyzing and representing human language [14]. The fundamental objective is to design algorithms that will give computers the ability to process, understand, and produce natural language in a manner that is comparable to that of humans. A wide array of applications, including as speech recognition, language translation, text categorization, and summarization, have been created throughout the years in this field. However, because natural language is so dependent on its surroundings, it is the most difficult thing for robots to duplicate when it comes to human engagement with natural language. When interpreting and creating language, people are able to take into account real-world situations and conditions; however, automated systems have not yet perfected this capacity to manage the comparable complexity of context [15]. Individuals possess the capacity to incorporate real-life circumstances and contexts while comprehending and generating language. The field of natural language processing (NLP) has witnessed significant advancements in algorithmic development. Initially, NLP algorithms primarily relied on basic statistical language models that focused solely on calculating the probability distribution of sentences, without considering semantic structures. However, over time, there has been a shift towards more sophisticated techniques such as parts-of-speech tagging and named entity recognition. These advanced methods aim to uncover the meaning of sentences by incorporating semantic components into the analysis [16]. This has been done in order to address the problems that have been brought to light by such problems. The use of deep learning as a strategy for different NLP tasks, such as text categorization, has become increasingly common since it achieves state-of-the-art outcomes

for a wide range of issues [17]. In Previous three studies on feature extraction from SRS document process requirement statement sentences that were already broken down into the list as they appear in the document. First, the statistical approach Term Frequency and Inverse Document Frequency (TF/IDF) is used in feature mining from the SRS document's functional requirement sentences. The research focused on several Mapping Rules (MRs) to identify the Semantic Model (SM) from each functional requirement sentence [18]. Second, Feature and Feature Relation Extraction (FFRE) tool for Eclipse plugin are also introduced to assist the feature model extraction process from the SRS document. This tool use NLP processing to identify actor, action, and object from each requirement sentence and heuristic processing afterward to determine which features are mandatory and which are optional [19]. Transfer learning has had a significant impact on the most current studies in the fields of software engineering and requirement engineering that are connected to natural language processing. The pre-trained language models (PLM), and more especially the encoder component of the Transformers, have shown to be the most successful models in terms of making use of transfer learning. In this sense, Bidirectional Encoder Representations from Transformers (BERT) has shown to be beneficial, particularly for issues involving the categorization of text and tokens, such as sentiment analysis and named entity identification. Recent years have seen the emergence of several BERT variations, and these variants are now being utilised in software analytics, including the classification jobs that involve SRS data [20]. The BERT model was fine-tuned by Hey et al. on particular tasks in which the model predicts whether a need is functional or non-functional. In a separate piece of research, Sainani et al. used BERT in a novel approach, in which the researchers first extracted requirements from huge software engineering contracts and then categorised those needs. Their strategy is predicated on the concept that business contracts may be able to assist in the identification of high-level requirements for the purpose of enhancing the overall performance of software engineering projects [21]. Kici et al. fine-tuned several PLMs for distinct SRS tasks. In order to determine the extent to which their findings are generalizable, the researchers examined three distinct datasets using a variety of BERT models and variations.

According to Regnell et al. (2001), requirements engineering places a strong emphasis on the prioritisation of needs. This is due to the fact that it is necessary to take into account the interests of a variety of stakeholders, as well as the fact that there are limited resources and time found that the process of prioritizing needs was fraught with numerous difficulties. This

was due to the huge number of factors involved in the process [22].

Research in natural language processing (NLP) involves integrating the knowledge and methods from several fields (such as computer science, artificial intelligence, and computational linguistics, for example) to develop automated analyses and ways to represent human language [23]. The fundamental objective is to design algorithms that will give computers the ability to process, comprehend, and produce natural language in a manner that is comparable to that of humans. A wide array of applications, including as speech recognition, language translation, text categorization, and text summarization, have been created throughout the course of the years in this field. However, because natural language is so dependent on its surroundings, it is the most difficult thing for robots to duplicate when it comes to human engagement with natural language. When interpreting and creating language, people are able to take into account real-world situations and conditions; nevertheless, automated systems have not yet completely mastered this capacity to manage the related context complexity [24]. People are able to take real-world situations and conditions into consideration when interpreting and producing language. To overcome the problem the field of natural language processing (NLP) has witnessed the advancement of algorithms. These algorithms have progressed from basic statistical language models, which solely calculate the probability distribution of sentences without taking into account semantic structures, to more sophisticated techniques such as parts-of-speech tagging and named entity recognition. The objective of these techniques is to uncover the meaning of sentences by considering the semantic elements present within them [25]. Deep learning has emerged as a widely used approach for a range of natural language processing (NLP) tasks, such as text categorization. It has demonstrated its effectiveness by obtaining state-of-the-art performance across multiple problem domains [26].

In Previous three studies on feature extraction from SRS document process requirement statement sentences that were already broken down into the list as they appear in the document. First, the statistical approach Term Frequency and Inverse Document Frequency (TF/IDF) is used in feature mining from the SRS document's functional requirement sentences. The research focused on several Mapping Rules (MRs) to identify the Semantic Model (SM) from each functional requirement sentence [27]. Second, Feature and Feature Relation Extraction (FFRE) tool for Eclipse plugin are also introduced to assist the feature model extraction process from the SRS document. This tool use NLP processing to identify actor, action, and object from each requirement sentence and heuristic processing afterward

to determine which features are mandatory and which are optional [28].

It is imperative that these documents adhere to predetermined criteria in order to establish a shared understanding of the specifications among all relevant parties, including users, analysts, and developers. Software Requirements Specifications (SRS) serve as valuable indicators for assessing the level of quality and acceptability of a product or process upon project completion. The evaluation of a software project's success is contingent upon the extent to which the final result aligns with the specifications outlined in the Software Requirements Specification (SRS) documentation. The presence of statements that can be interpreted in multiple ways, explanations that lack clarity, or inferences that are not clear might lead to significant failures at a later stage of the project [29]. Therefore, achieving a comprehensive and coherent comprehension of these standards necessitates the consistent utilization of contextual terminology within the given area. Moreover, the efficacy of following phases in the Systems Development Life Cycle (SDLC) is contingent upon the establishment of clearly specified requirements and their precise execution. Neglecting to adhere to this practice may result in delays and additional expenses [30]. Studies on the feature extraction from existing systems mostly use the source code as the object or input of the extraction process while other studies also conducted to use models including a class diagram and use case diagram as the objects for the extraction process [31].

However, most software developer only measures their product quality on the released software product or the implementation result regardless of the original requirement [32]. Therefore, software feature extraction from specification document is more suitable based on the software engineering perspective rather than model or source code to acquire the more valid feature. This is because the specification document is the basis of the validation and verification of system functionality in the software development process [33]. Presently, the majority of research efforts related to the extraction of Software Product Line (SPL) features from Software Requirement Specification (SRS) documents have focused on processing pre-existing lists of needs, rather than considering the SRS document as a cohesive entity [34]. Therefore, this approach still necessitates the involvement of an expert to manually extract required sentences from the SRS document, a task that can be both laborious and susceptible to errors. This study involves the direct processing of SRS documents that employ requirement boilerplate in order to build requirement statements. These constraints establish precise patterns that can be analyzed using the Natural Language Processing (NLP) technique [35].

2.2 Comparison of Literature review

Table 1 Comparison table of previous studies

Ref paper	Year	Dataset	NLP Classifier	Evaluation parameters	Results Accuracy
[1]	2020	PURE	FM, VSM, LSA,	Accuracy, precision, recall and f1 score	80%,82%, 83%
[2]	2021	NFR Promise And DOORS	DistilBERT, Bi LSTM	Accuracy, precision, recall and f1 score	80%, 77%
[3]	2021	DOORS	RoBERTa, BiLSTM, BERT	Accuracy, precision, recall and f1 score	80%,77%, 76%
[4]	2018	PURE	SVM, KNN,LR,	Accuracy, precision, recall and f1 score	77%
[5]	2017	SRS	CNN	Accuracy, precision, recall and f1 score	79%
[6]	2020	PURE	FM, VSM, LSA,	Accuracy, precision, recall and f1 score	80%,82%, 83%
[7]	2021	NFR Promise And DOORS	DistilBERT, Bi LSTM	Accuracy, precision, recall and f1 score	80%, 77%
[8]	2021	DOORS	RoBERTa, BiLSTM, BERT	Accuracy, precision, recall and f1 score	80%,77%, 76%
[9]	2018	PURE	SVM, KNN,LR,	Accuracy,	77%

Ref paper	Year	Dataset	NLP Classifier	Evaluation parameters	Results Accuracy
				precision, recall and f1 score	
[10]	2017	SRS	CNN	Accuracy, precision, recall and f1 score	79%
[11]	2020	PURE	FM, VSM, LSA,	Accuracy, precision, recall and f1 score	80%,82%, 83%
[12]	2021	NFR Promise And DOORS	DistilBERT, Bi LSTM	Accuracy, precision, recall and f1 score	80%, 77%
[13]	2021	DOORS	RoBERTa, BiLSTM, BERT	Accuracy, precision, recall and f1 score	80%,77%, 76%
[14]	2018	PURE	SVM, KNN,LR,	Accuracy, precision, recall.	77%
[15]	2017	SRS	CNN	Accuracy, precision, recall and f1 score	79%
[16]	2020	PURE	FM, VSM, LSA,	Accuracy, precision, recall and f1 score	80%,82%, 83%
[17]	2021	NFR Promise And DOORS	DistilBERT, Bi LSTM	Accuracy, precision, recall and f1 score	80%, 77%
[18]	2021	DOORS	RoBERTa, BiLSTM, BERT	Accuracy, precision, recall and f1 score	80%,77%, 76%
[19]	2018	PURE	SVM, KNN,LR,	Accuracy, precision, recall and f1 score	77%
[20]	2017	SRS	CNN	Accuracy, precision, recall and f1 score	79%

2.3 Research Questions

Research questions have been summarized as below:

RQ1: What NLP tools/techniques used in requirements engineering?

RQ2: What automated requirement prioritization technique available in literature?

RQ3: Does NLP already used in requirement prioritization and what is recorded accuracy?

RQ4: Does supervised learning or data classification used in requirement prioritization?

2.4 Research gap

In this section area for the improvement in the existing research literature is discussed. A detailed analysis of the selected articles was carried out in which tools, techniques, frameworks, and other NLP approaches were used. After a comprehensive screening procedure filtered the research that stipulates an endorsement for the detection of the ambiguities caused by terms in requirements across different domain. At the moment, there are few limitations in published researches on feature extraction from natural language documents, i.e. unavailable tools for evaluation, restricted or limited input, irrelevant feature naming, non-reproducible result, and domain engineer intervention in the process [2]. While this research is aimed to produce a tool for automatically extracting software features directly from SRS documents without any human intervention in the process. The tool will be applied and tested using selected SRS from the Public Requirement Engineering (DOORs) dataset [3] to justify its correctness

3 PROPOSED APPROACH

This Chapter presents approach for the most suitable alternative of the Word2Vec algorithm for the detection of ambiguities caused by the terms used in natural language requirements that are domain dependent. The approach has a pipeline of data collection, pre-processing of data, building language model, apply alternative algorithm(s), and resultant terms score of dissimilarity.

In fig 2. Shows the steps proposed method in block diagram. We have used four deep learning models for the classification the BERT, RoBERTa, BiLSTM and LSTM for DOORS dataset. In our methodology first we make sentence segmentation then generate the tokens then use so part of speech the ass to entity recognition and then the relating recognition and final step fond the references of word in software requirements specification . We considered three characteristics for classification of SRS documents, namely, Type, Priority and Severity.

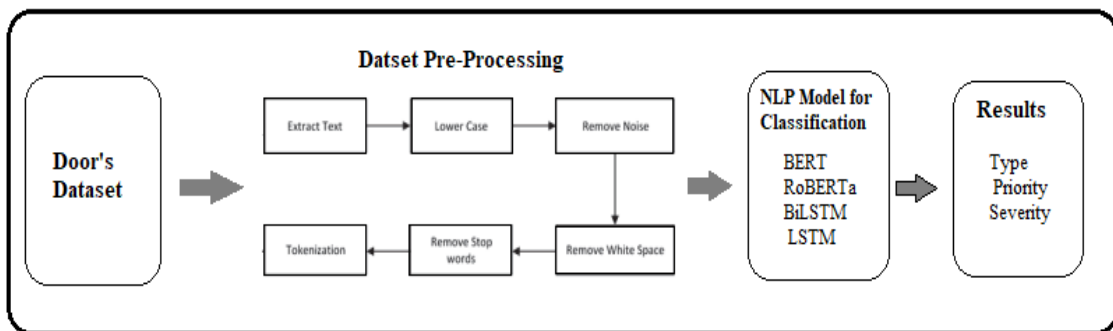


Figure 2 Block Diagram of Proposed method

3.1 Dataset

In this work, our SRS data was obtained with the Dynamic Object Oriented Requirements System (DOORS) Next Generation, a requirement management technology created and sold by IBM [2], which is extensively used by engineers all around the world. The SRS dataset that we are working with has 83,837 occurrences and 213 characteristics. However, in order to identify the categorical elements of the needs, such as their kind,

severity, and priority, we just looked at the summary of the criteria. When classifying SRS papers, we took into consideration three characteristics: kind, priority, and severity.

3.2 Class distribution of categorical features in the DOORS SRS dataset

The breakdown of the classes into their respective category labels is presented in figure there are four classes in the Priority, and those classes are unassigned, high, medium, and low. There are also six classes in the Severity, and those classes are normal, major, minor, blocker, critical, and undetermined. The Type category had a total of 20 distinct classes when it was first created. After performing certain procedures including merging and preprocessing, we were able to acquire seven classes for the Type category.

In addition, we got rid of the values that were represented by the notation 'nan,' which stood for the absence of values for the Priority and Severity categories. It has come to our attention that the classes belonging to the Type category are relatively distributed evenly, however the classes belonging to the other categories exhibit a serious imbalance in the data.

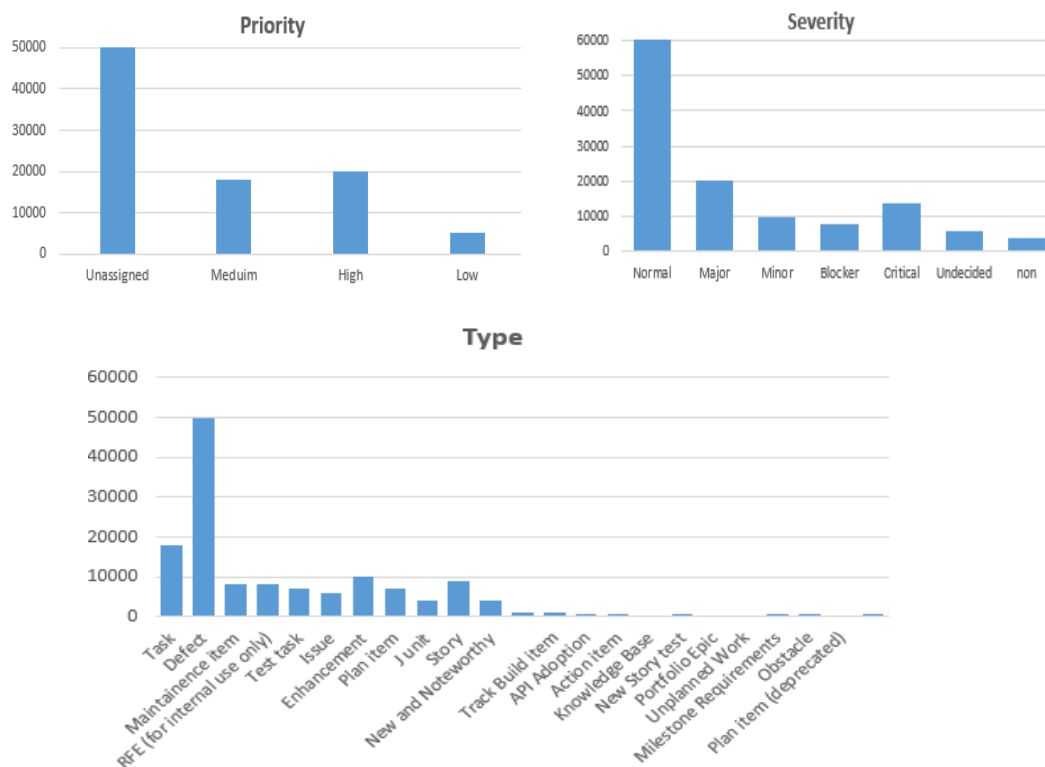


Figure 3. Class distribution for the each class level [2]

3.3 Data Pre-Processing

In the preprocessing steps we applied tokenization by splitting sentences into individual's words. We extract the text from SRS and gives output in the form of paragraph. In second we convert the all the characteristics to lowercase as an uppercase letter for feature extraction then removal of noise and remove the white spaces. In third removed the stop words then we applied tokenization by splitting sentences into individuals words. We next moved to classification process on classes we considered the three characteristics: kind, priority, and severity.

3.4 Experimental Setup Settings

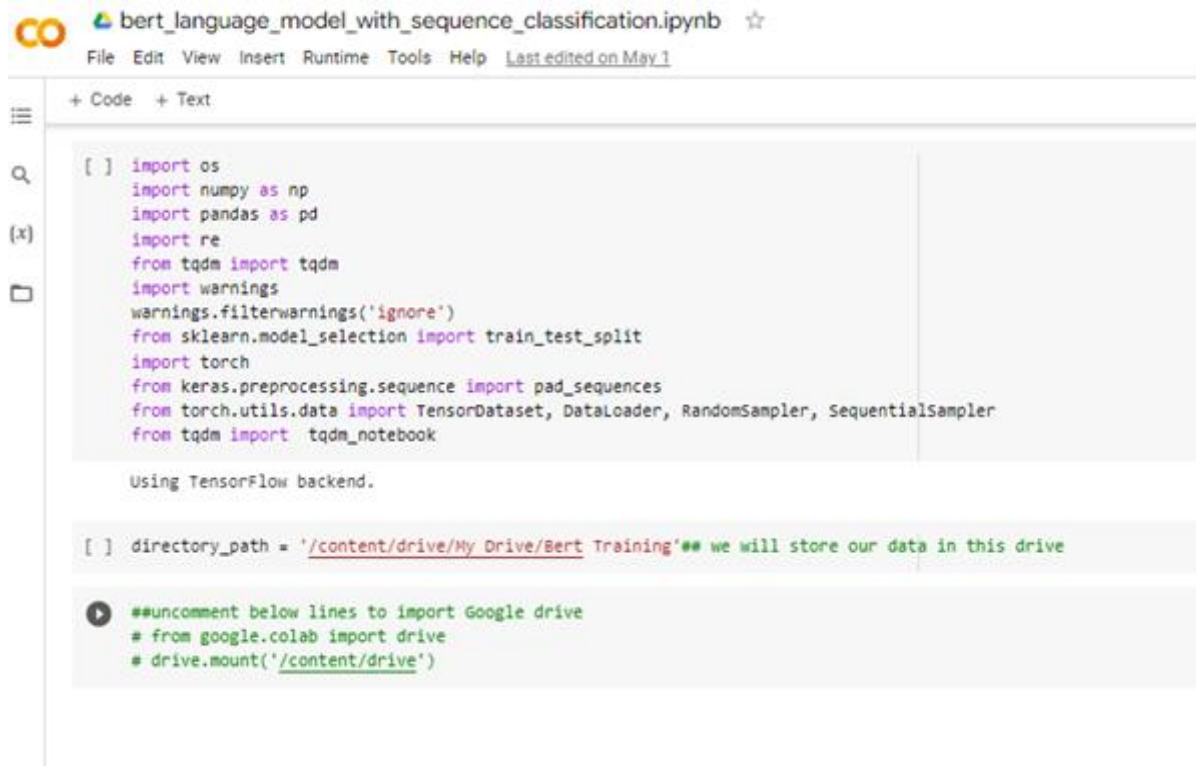
In a standard supervised training pipeline, unlabeled data would be removed, and the fine-tuning step would be carried out on a dataset that has been labelled. More than 80,000 requirement texts are included in the DOORS dataset; however, the bulk of these texts do not have any labels attached to them. As a result of this, we use the MLM algorithm as the unsupervised learning objective for the adaptive fine-tuning process. This algorithm uses all of the requirement texts from the complete dataset. As part of the MLM implementation process, the tokens in the input are masked at random with a chance of 0.15. As a result of this procedure, a brand-new adapted pre-trained model with the newly calibrated parameters, θ_S , is produced.

In the remaining steps of the training operation, we will adhere to the standard practice of using hyper-parameter selection. The dataset is comprised of three different sets: the training set, the validation set, and the test set. To maintain consistency with the other pre-training hyper-parameter settings, such as BERT checkpoints, the learning rate is maintained at around $2e-5$. The AdamW optimizer is used throughout the training process, which lasts for a total of 30 iterations [43]. At the very end, just the very best model that was discovered throughout the training gets loaded. After that, the phase of fine-tuning is applied as the last step.

In the phase that comes following the adaptive phase and is known as the fine-tuning phase, we likewise adhere to the standard practice. To be more specific, we have downstream activities that target the tree categorization categories of Priority, Severity, and Type. For each task, we used up to three epochs to fine-tune a previously trained model, namely, an adapted previously trained model that was produced in step two. We have noticed that maintaining a relatively low value for the learning rate at this step is

quite critical in order to prevent the modified language model from being corrupted.

Implementations and Coding



```
[ ] import os
import numpy as np
import pandas as pd
import re
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
import torch
from keras.preprocessing.sequence import pad_sequences
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
from tqdm import tqdm_notebook

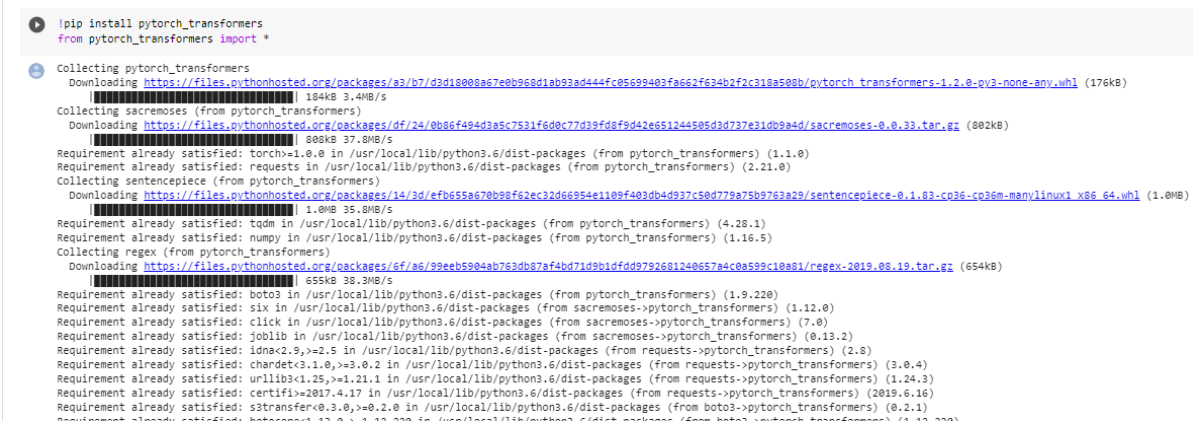
Using TensorFlow backend.

[ ] directory_path = '/content/drive/My Drive/Bert Training'## we will store our data in this drive

##uncomment below lines to import Google drive
# from google.colab import drive
# drive.mount('/content/drive')
```

Fig4. Implementation and importing libraries using python language

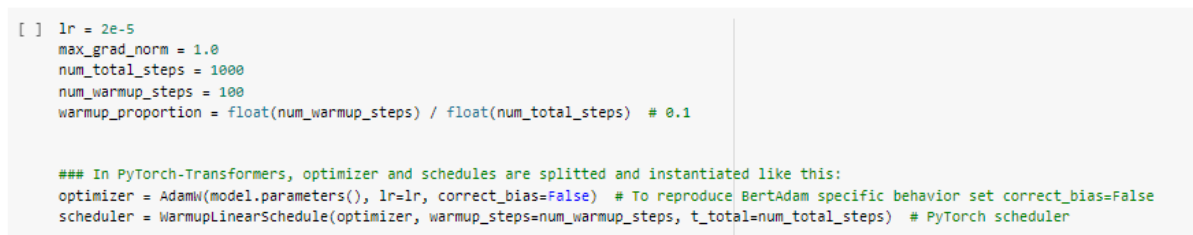
Now, Finetuning Language model on the data



```
!pip install pytorch_transformers
from pytorch_transformers import *

Collecting pytorch_transformers
  Downloading https://files.pythonhosted.org/packages/as/b7/d3d1800a67e0b96ad1ab93ad444fc05699403facc2fe34b2fc319a508b/pytorch_transformers-1.2.0-py3-none-any.whl (176KB)
    104KB 3.4MB/s
Collecting sacremoses (from pytorch_transformers)
  Downloading https://files.pythonhosted.org/packages/df/24/0b86f494d3a5c7531fed0c77d39fd8fd42e651244505d3d737e31db9a4d/sacremoses-0.0.33.tar.gz (802KB)
    808KB 37.8MB/s
Requirement already satisfied: torch>=1.0.0 in /usr/local/lib/python3.6/dist-packages (from pytorch_transformers) (1.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from pytorch_transformers) (2.21.0)
Collecting sentencepiece (from pytorch_transformers)
  Downloading https://files.pythonhosted.org/packages/14/3d/efb55a670b98f62ec32d66954e109f403db4d9337c50d779a75b9763a29/sentencepiece-0.1.83-cp36-cp38m-manylinux1_x86_64.whl (1.0MB)
    1.0MB 35.8MB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from pytorch_transformers) (4.28.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from pytorch_transformers) (1.16.5)
Collecting regex (from pytorch_transformers)
  Downloading https://files.pythonhosted.org/packages/6f/a6/99eeb5904ab763db87af4bd71d9b1dfd9792681240657a4c0a599c10a81/regex-2019.08.19.tar.gz (654KB)
    655KB 38.3MB/s
Requirement already satisfied: boto3 in /usr/local/lib/python3.6/dist-packages (from pytorch_transformers) (1.9.220)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from sacremoses->pytorch_transformers) (1.12.0)
Requirement already satisfied: click in /usr/local/lib/python3.6/dist-packages (from sacremoses->pytorch_transformers) (7.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages (from sacremoses->pytorch_transformers) (0.13.2)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->pytorch_transformers) (2.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests->pytorch_transformers) (3.0.4)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests->pytorch_transformers) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests->pytorch_transformers) (2019.6.16)
Requirement already satisfied: s3transfer<0.3.0,>=0.2.0 in /usr/local/lib/python3.6/dist-packages (from boto3->pytorch_transformers) (0.2.1)
Requirement already satisfied: botocore<1.12.0,>=1.12.0 in /usr/local/lib/python3.6/dist-packages (from boto3->pytorch_transformers) (1.12.0)
```

Fig5. Fine Tuning the Model



```
[ ] lr = 2e-5
max_grad_norm = 1.0
num_total_steps = 1000
num_warmup_steps = 100
warmup_proportion = float(num_warmup_steps) / float(num_total_steps) # 0.1

### In PyTorch-Transformers, optimizer and schedules are splitted and instantiated like this:
optimizer = AdamW(model.parameters(), lr=lr, correct_bias=False) # To reproduce BertAdam specific behavior set correct_bias=False
scheduler = WarmupLinearSchedule(optimizer, warmup_steps=num_warmup_steps, t_total=num_total_steps) # PyTorch scheduler
```

Fig6. Fine-tuning and adjusting hyper parameters

```

tr_loss = 0
nb_tr_examples, nb_tr_steps = 0, 0

# Train the data for one epoch
for i, batch in enumerate(train_dataloader):
    # Add batch to GPU
    batch = tuple(t.to(device) for t in batch)
    # Unpack the inputs from our dataloader
    b_input_ids, b_input_mask, b_labels = batch
    # Forward pass
    outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)
    loss = outputs[0]
    train_loss_set.append(loss.item())
    # Backward pass
    loss.backward()
    # Update parameters and take a step using the computed gradient
    optimizer.step()
    scheduler.step()
    optimizer.zero_grad()
    if (i) % 50 == 0:
        print ('Epoch [{} / {}], Step [{} / {}], Loss: {:.4f}'
              .format(epoch+1, epochs, i+1, total_step, loss.item()))

torch.save(model.state_dict(), directory_path+'model_with_language_model.ckpt')

```

```

Epoch [1/2], Step [1/1250], Loss: 0.8097
Epoch [1/2], Step [51/1250], Loss: 0.8031
Epoch [1/2], Step [101/1250], Loss: 0.6182
Epoch [1/2], Step [151/1250], Loss: 0.3168
Epoch [1/2], Step [201/1250], Loss: 0.1888
Epoch [1/2], Step [251/1250], Loss: 0.1677
Epoch [1/2], Step [301/1250], Loss: 0.2950
Epoch [1/2], Step [351/1250], Loss: 0.1150
Epoch [1/2], Step [401/1250], Loss: 0.0999

```

Fig7: Training the Data and calculating the loss

bert_language_model_with_sequence_classification.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

+ Code + Text

```

[ ] # Test the model
with torch.no_grad():
    correct = 0
    total = 0
    for i, batch in enumerate(validation_dataloader):
        batch = tuple(t.to(device) for t in batch)
        # Unpack the inputs from our dataloader
        b_input_ids, b_input_mask, b_labels = batch
        # Forward pass
        outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_i
        # print (outputs)
        prediction = torch.argmax(outputs[0],dim=1)
        total += b_labels.size(0)
        correct+=(prediction==b_labels).sum().item()

```

```

print('Test Accuracy of the model on val data is: {} %'.format(100 * corre
Test Accuracy of the model on val data is: 84.9 %

```

Fig8. Tested the accuracy

3.5 NLP models for Classification

We have used four deep learning models BERT, RoBERTa, LSTM and BiLSTM.

- **BERT:** The most well-known encoder that makes use of Masked Language Model (MLM) and Next Sentence Prediction (NSP) aims is called BERT. The BERT-base-uncased model has a hidden size of 768, which equates to 12 heads, and a head embedding size of 64. This model also does not have a casing. The model is comprised of a total of twelve layers. BERT-largeuncased, the big equivalent model, has a hidden size of 1024 and has 24 layers and 16 attention heads. BERT checkpoints were designed with a learning rate of $1e-4$ in mind originally. In order to avoid interrupting the learning that takes place during pre-training when it is being fine-tuned, a slower learning rate (such as $1e-5$ or $2e-5$) is chosen.
- **RoBERTa:** A variant of BERT that is more stable and efficient is known as RoBERTa. The BERT large version is utilized as the basis for its underlying structure. The model is trained for a longer period of time with larger batches, and the following sentence 6 has been removed. These are some of the adjustments made to vanilla BERT prediction target, training on bigger sequences, and dynamically modifying the masking pattern are all aspects of modelling that need to be improved.
- **LSTM:** It has been discovered that LSTM networks [26] are capable of learning the long-term relationships and patterns that are present in natural language text. LSTMs have a significant advantage over other RNNs due to the fact that their use of forget and update gates enable them to manage and maintain consistent information flow. This successfully protects them from the problems of disappearing and exploding information, which are two of the most common RNN-related problems.
- **BiLSTM:** The input data are processed twice by the bidirectional variation of the LSTM model [27], once from the beginning to the end and once from the end to the beginning. The BiLSTM technique investigates the more profound semantics of the word structure. Training in both the forward and backward direction offers an extra comprehension of the contextual dependency that is present in the tokens.

4 IMPLEMENTATION, RESULTS & DISCUSSION

In this chapter, implementation of the approach as stated in the chapter 3 is discussed, and results of the approach are analyzed in detail. The implementation consist of data collection of different fields and results preparation and examination. Basic information of the approach implementation, use of algorithms and programs is also discussed in this section.

In this section we have discussed obtained results and experiments. These four CNN architecture (BERT, RoBERTa, LSTM, BiLSTM) are used with hyper parameters to process the classification. We retrieved the results and compare the results. Our experiments are designed to investigate the effectiveness of deep learning models and transfer learning in SRS document classification.

4.1 Evaluation methodology

In order to do the analysis, we partitioned the dataset into three distinct sections: training, testing, and validation. We consider 80% of the data to be training data in order to update the weights in the fine-tuning phase. 10% of the data is used for validation in order to assess the out-of-sample performance of the model while it is being trained, and 10% of the data is used for final testing in order to measure the out-of-sample performance of the model after it has been trained. We utilize stratified sampling to choose 0.8, 0.1, and 0.1 sections of the SRS document from each class for training, validation, and testing respectively. This helps us avoid over-fitting the data.

4.2 Evaluation Parameters

The performance of the model was assessed based on a number of measures, such as accuracy, precision, recall, and f1 score, all of which are derived through the use of the following formulas:

- **Accuracy:** Accuracy is a statistic that gauges the overall performance of the model under the assumption that all classes have the same weight. It is determined by taking the ratio of the true positives to the total number of samples and subtracting one from the other. In the formula TP means true positive and TN means true negative FP means false positive and FN means

false negative.

The formula for accuracy is as follows:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

- **Precision:** Precision is a statistic that evaluates the ratio of correctly recognized positive samples to the total number of samples that are identified as positive, regardless of whether or not the identification was correct. It does it by determining how accurate the model's predictions are. In the formula TP means true positive and TN means true negative FP means false positive and FN means false negative The formula for calculating precision is as follows:

$$Precision = TP / (TP + FP)$$

- **Recall:** This metric calculates the ratio of the number of positive samples to the total number of positive samples and determines whether or not the positive samples were accurately labelled as positive. In the formula TP means true positive and TN means true negative FP means false positive and FN means false negative. It analyses how effectively the model can locate positive samples by looking at the following factors:

$$Recall = TP / (TP + FN)$$

- **F1 Score:** This statistic takes into account both accuracy and recall in determining its value. It determines how effectively the accuracy and recall measures interact with one another:

$$F1\ Score = 2 * (Precision * Recall) / (Precision + Recall)$$

4.3 Proposed approach

In our proposed approach RoBERTa achieved higher accuracy for the classification on the DOORs dataset. We have used the pre-trained RoBERTa model with transformers for the feature extraction. We have performed feature preprocessing on dataset like data cleaning data removing null values and the arrangements of data extracting the required data the pass it to the NLP model for training then test the data on test set. We split the data into train set, test set and validation set. We tested the accuracy on validation set.

4.4 Optimizers

Optimizers are designed with the purpose of improving the effectiveness of learning algorithms by enhancing the accuracy with which they perform prediction tasks. Given this, it should come as no surprise that a solid pick of It is impossible for the prediction model to be successful without an optimizer. An optimization method known as Adam [14] is responsible for calculating the adaptive learning rates of individual parameters by making use of just first order gradients. It combines the strengths of many optimization methods, such as RMSprop and AdaGrad. It functions well with sparse gradients, a characteristic that was obtained from AdaGrad, and it functions well with mobile and online environments, a feature that was obtained from RMSprop. An alternative implementation of the Adam optimization method is known as Adamax [15].

4.5 RoBERTa

To maintain consistency with the other pre-training hyper parameter settings, such as RoBERTa checkpoints, the learning rate is maintained at around $1e-4$. The Adam W optimizer is used throughout the training process, the number of hidden layers is 16, the hidden size is 1024, the maximum sequence length is 100, the epoch is 3, the dropout rate is 0.1, the learning rate is $5e-04$, and the batch size is 8.

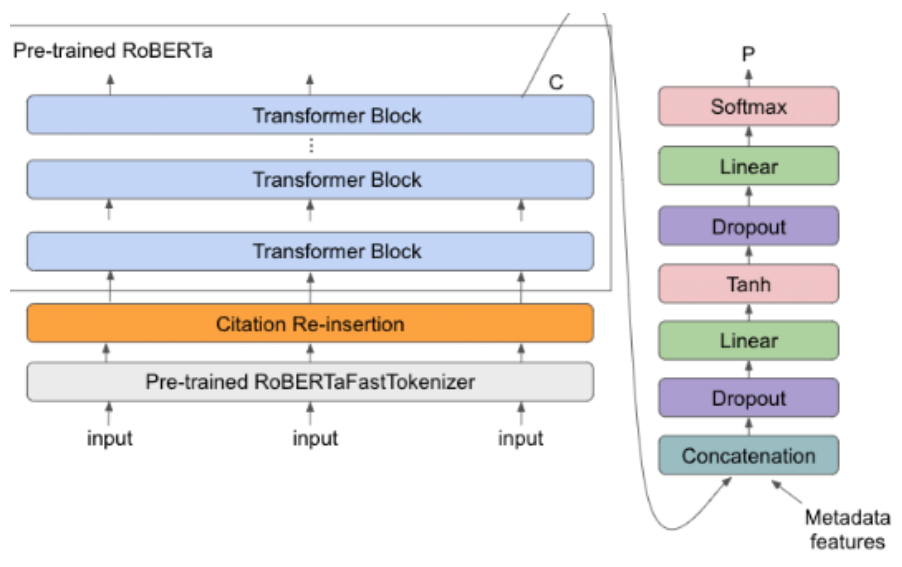


Figure 10. Architecture Diagram of RoBERTa [20]

4.6 Results of NLP Models

The below table shows the results and the accuracy of the model of BERT, DisilBERT, LSTM and BiLSTM results. The RoBERTa obtained highest accuracy then the other CNN models. The evaluation performed on the basis of accuracy precision recall and F1 score.

Table 2 Classification Results of proposed models

Proposed NLP Models	Accuracy	Precision	Recall	F1-score
BERT	78.9%	0.79	0.80	0.77
RoBERTa	84.9%	0.81	0.87	0.84
LSTM	76%	0.74	0.75	0.76
BiLSTM	80.2%	0.75	0.81	0.80

RoBERTa, BERT and BiLSTM model is evaluated using optimizers, such as adam, as well as the number of epochs and the number of LSTM .This is done in order to determine the optimal choice of hyper parameters, which eventually results in a classification model that is more accurate.

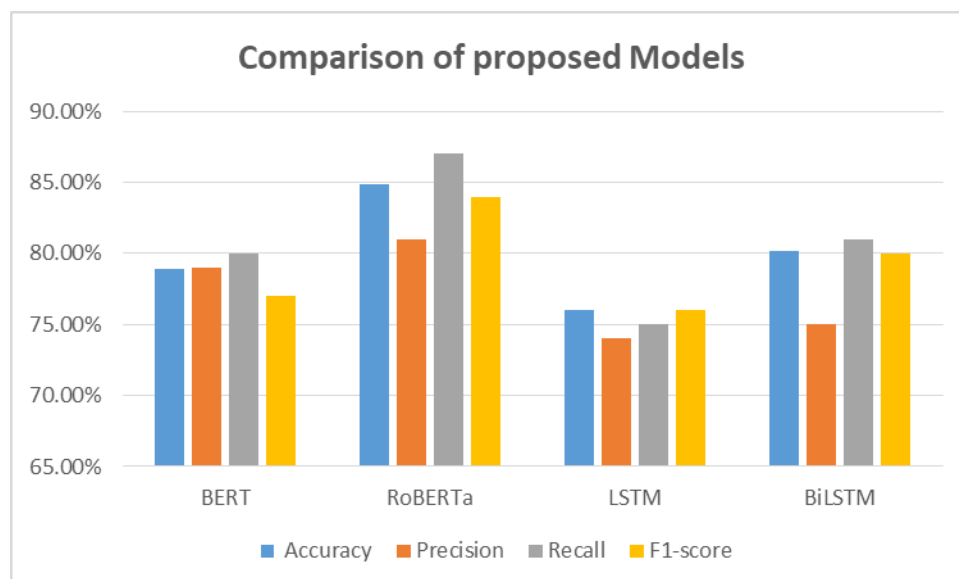


Figure 11. Results comparison of proposed model.

Table 2 and Table 3 showed the comparative results of the models were reported for the categorization of DOORS data according to Priority, and Severity categories, respectively.

Table 3 Results of Priority

Proposed NLP Models	Accuracy	Precision	Recall	F1-score
BERT	75.9%	0.76	0.78	0.74
RoBERTa	78.9%	0.79	0.80	0.77
LSTM	69.2%	0.71	0.65	0.69
BiLSTM	72.2%	0.74	0.75	0.72

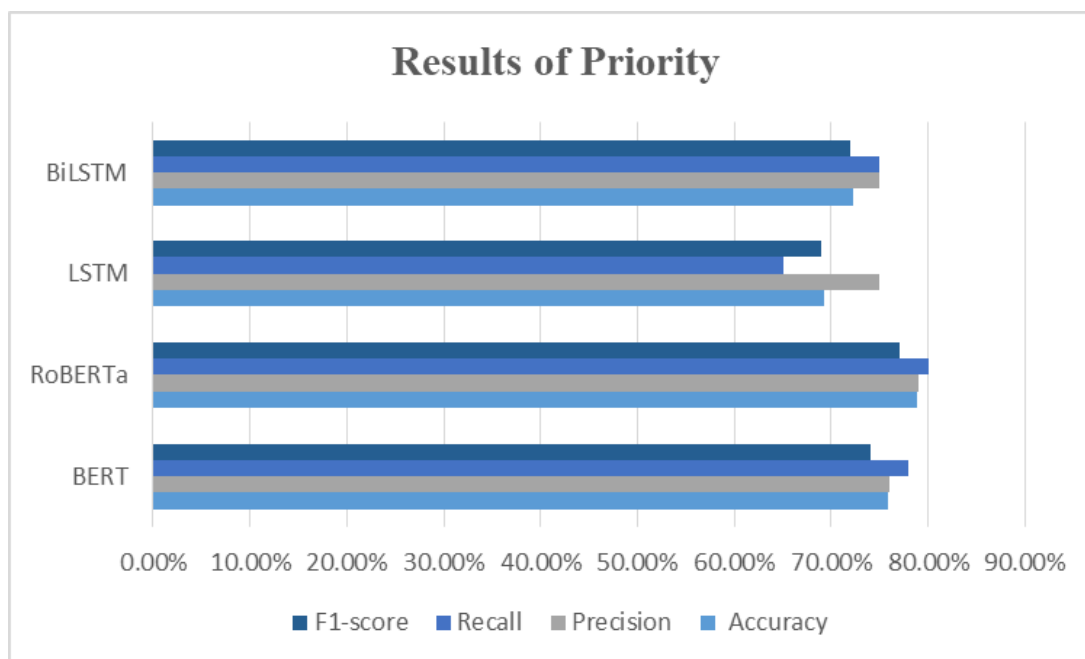


Figure 12. Results comparison of priority in DOORs Dataset

Table 4 Results of Severity

Proposed NLP Models	Accuracy	Precision	Recall	F1-score
BERT	76%	0.76	0.80	0.76
RoBERTa	80%	0.78	0.85	0.80
LSTM	74%	0.72	0.75	0.76
BiLSTM	78%	0.78	0.78	0.78

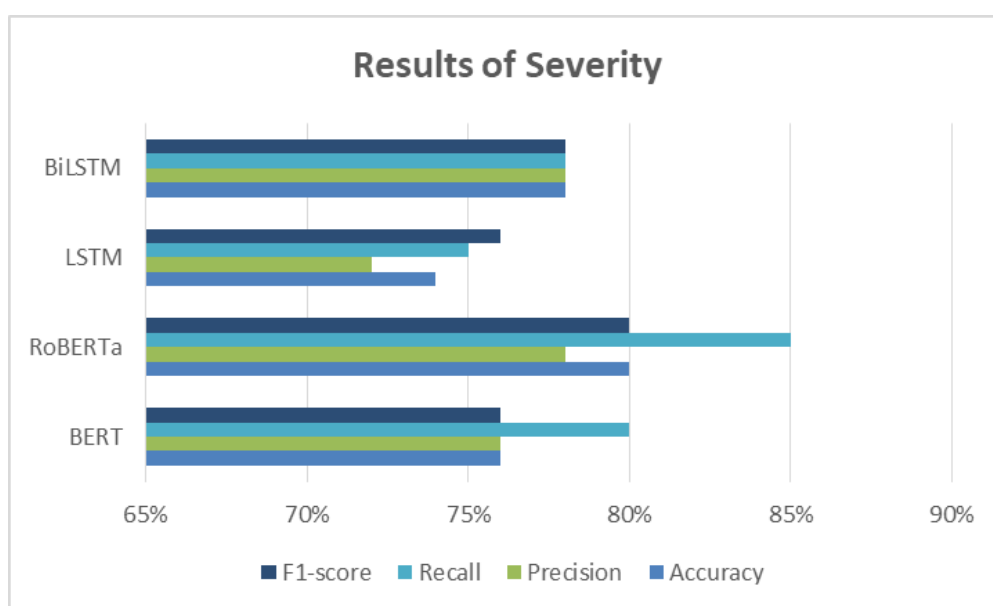


Figure 13. Results comparison of Severity in DOORs Dataset

4.7 Comparison between Proposed approach and existing approaches

The table 5 shows the comparisons between the proposed approach and existing approaches. Our proposed approach BERT achieved 84.7% accuracy with 0.84 F1 score.

At the moment, there are few limitations in published researches on feature extraction

from natural language documents, i.e. unavailable tools for evaluation, restricted or limited input, irrelevant feature naming, non-reproducible result, and domain engineer intervention in the process [2]. While this research is aimed to produce a tool for automatically extracting software features directly from SRS documents without any human intervention in the process. The tool will be applied and tested using selected SRS from the Public Requirement Engineering (DOORs) dataset [3] to justify its correctness.

Table 5 Comparison between Proposed approach and existing approach

NLP Models	Existing approaches Accuracy	Proposed approach Accuracy
RoBERTa [2]	80%	84%
BiLSTM [2]	77%	80%
BERT [2]	76%	78%

5 CONCLUSION & FUTURE WORK

5.1 Conclusion

In this study we examined the effectiveness of the RoBERTa transformer as a state-of-the-art transfer learning model for multi-class text classification over SRS documents. The proposed approach RoBERTa achieved higher accuracy than the other models. We successfully classified the priority requirements using natural language processing technique. For the classification we prioritize the requirements by labeling the data and then classify on the basis of high priority. We compared our proposed approach with existing approach to show the effectiveness of our approach. We successfully achieved higher accuracy than previous studies.

5.2 Future Work

For the future work we will the proposed approach can be extended to implement by increasing the number of algorithms and implement these models by adding layers and considered more dataset.

REFERENCES

- [1] Haris, M.S., Kurniawan, T.A. and Ramdani, F., 2020. Automated features extraction from software requirements specification (SRS) documents as the basis of software product line (SPL) engineering. *Journal of Information Technology and Computer Science*, 5(3), pp.279-292.
- [2] Kici, D., Malik, G., Cevik, M., Parikh, D. and Basar, A., 2021, June. A BERT-based transfer learning approach to text classification on software requirements specifications. In *Canadian Conference on AI*.
- [3] Savas Yildirim¹, Mucahit Cevik^{1*}, Devang Parikh² and Ayse Basar¹ ¹ Toronto Metropolitan University, 44 Gerrard St E, Toronto, M5B 1G3, Ontario, Canada. ² IBM, Cary, 27709, North Carolina, USA. Adaptive Fine-tuning for Multiclass Classification over Software Requirement Data
- [4] Ali, M. Asif, M. Shahbaz, A. Khalid, M. Rehman, and A. Guergachi. “Text categorization approach for secure design pattern selection using software requirement specification”. In: *IEEE Access* 6 (2018), pp. 73928–73939.
- [5] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea. “Towards supporting software engineering using deep learning: A case of software requirements classification”. In: *CONISOFT 2017*. IEEE. 2017, pp. 116–120.
- [6] Hussain, O. Ormandjieva, and L. Kosseim. “Automatic quality assessment of SRS text by means of a decision-tree-based text classifier”. In: *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE. 2007, pp. 209–218.
- [7] U Shah and D Jinwala. “Resolving ambiguities in natural language software requirements: a comprehensive survey”. In: *ACM SIGSOFT Software Engineering Notes* 40.5 (2015), pp. 1–7.
- [8] S. Ruder, M. E. Peters, S. Swayamdipta, and T. Wolf. “Transfer learning in natural language processing”. In: *Proceedings of the 2019 Conference of the North American Chapter of the ACL: Tutorials*. 2019, pp. 15–18.
- [9] E. Hull, K. Jackson, and J. Dick. “DOORS: a tool to manage requirements”. In: *Requirements engineering*. Springer, 2002, pp. 187–204.
- [10] J. Dick, E. Hull, and K. Jackson. “Requirements Engineering in the Problem Domain”. In: *Requirements Engineering*. Springer, 2017, pp. 113–134.

- [11] T. Young, D. Hazarika, S. Poria, and E. Cambria. “Recent trends in deep learning based natural language processing”. In: *IEEE Computational Intelligence Magazine* 13.3 (2018), pp. 55–75.
- [12] B. S. Haney. “Patents for NLP Software: An Empirical Review”. In: Available at SSRN 3594515 (2020). [13] N. Ranjan, K. Mundada, K. Phaltane, and S. Ahmad. “A Survey on Techniques in NLP”. In: *International Journal of Computer Applications* 134.8 (2016), pp. 6–9.
- [13] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao. “Deep learning based text classification: A comprehensive review”. In: *arXiv preprint arXiv:2004.03705* (2020).
- [14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).
- [15] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *arXiv preprint arXiv:1910.01108* (2019).
- [16] A. Bacchelli, T. Dal Sasso, M. D’Ambrosio, and M. Lanza. “Content classification of development emails”. In: *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 375–385.
- [17] B. Khan, T. Syed, Z. Khan, and M. Rafi. “Textual analysis of End User License Agreement for red-flagging potentially malicious software”. In: *ICECCE 2020*. IEEE, 2020, pp. 1–5. [19] E. Dias Canedo and B. Cordeiro Mendes. “Software Requirements Classification Using Machine Learning Algorithms”. In: *Entropy* 22.9 (2020), p. 1057.
- [18] D. Ott. “Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements”. In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2013, pp. 50–64
- [19] J. Wieting, M. Bansal, K. Gimpel, K. Livescu, Towards universal paraphrastic sentence embeddings, *arXiv preprint arXiv:1511.08198* (2015).
- [20] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Céspedes, S. Yuan, C. Tar, et al., Universal sentence encoder, *arXiv preprint arXiv:1803.11175* (2018).

- [21] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: International conference on machine learning, PMLR, 2014, pp. 1188–1196.
- [22] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, arXiv preprint arXiv:1908.10084 (2019).
- [23] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, S. Fidler, Skip-thought vectors, in: Advances in neural information processing systems, 2015, pp. 3294–3302.
- [24] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep contextualized word representations, arXiv preprint arXiv:1802.05365 (2018).
- [25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in neural information processing systems, 2017, pp. 5998–6008.
- [26] A. Mustafa, W. M. W. Kadir, and N. Ibrahim, “Automated Natural Language Requirements Analysis using General Architecture for Text Engineering (GATE) Framework,” *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 3–4, pp. 97–101, 2017, [Online]. Available: <https://jtec.utem.edu.my/jtec/article/view/2925>
- [27] V. Jain, R. Malhotra, S. Jain, and N. Tanwar, “Cross-Domain Ambiguity Detection using Linear Transformation of Word Embedding Spaces.” arXiv, Mar. 29, 2020.
- [28] Accessed: Apr. 11, 2023. [Online]. Available: <http://arxiv.org/abs/1910.12956>
- [29] S. Çevikol and F. B. Aydemir, “Detecting Inconsistencies of Natural Language Requirements in Satellite Ground Segment Domain,” REFSQ Workshops, 2019, [Online].
- [30] Available: https://ceur-ws.org/Vol-2376/NLP4RE19_paper15.pdf
- [31] A. Chattopadhyay, N. Niu, Z. Peng, and J. Zhang, “Semantic Frames for Classifying Temporal Requirements: An Exploratory Study,” REFSQ Workshops, pp. 1–9, 2021, [Online]. Available: <https://homepages.uc.edu/~niunn/papers/NLP4RE21.pdf>
- [32]
- [33] M. Arrabito, A. Fantechi, S. Gnesi, and L. Semini, “A comparison of NLP Tools for RE to extract Variation Points,” REFSQ Workshops, 2020, [Online]. Available: <https://ceur-ws.org/Vol-2584/NLP4RE-paper1.pdf>

- [34] T. Baldwin, Y. Li, B. Alexe, and I. R. Stanoi, “Automatic Term Ambiguity Detection,” Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 804–809, Aug. 2013.
- [35] D. M. Blei, Andrew Y. Ng, and Michael I. Jordan, “Latent Dirichlet Allocation,”
- [36] Journal of Machine Learning Research 3 (2003) 993-1022, pp. 993–1022, Jan. 2003.
- [37] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, “Automated Extraction and Clustering of Requirements Glossary Terms,” IEEE Trans. Softw. Eng., vol. 43, no. 10, pp. 918–945, Oct. 2017, doi: 10.1109/TSE.2016.2635134.
- [38] Z. S. Harris, “Distributional Structure,” WORD, vol. 10, no. 2–3, pp. 146–162, Aug. 1954, doi: 10.1080/00437956.1954.11659520.
- [39] R. Collobert and J. Weston, “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning”.
- [40] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality”.
- [41] P. D. Turney and P. Pantel, “From Frequency to Meaning: Vector Space Models of Semantics,” J. Artif. Intell. Res., vol. 37, pp. 141–188, Feb. 2010, doi: 10.1613/jair.2934.
- [42] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space.” arXiv, Sep. 06, 2013. Accessed: May 05, 2023. [Online]. Available: <http://arxiv.org/abs/1301.3781>
- [43] K. Bhatia, S. Mishra, and A. Sharma, “Clustering Glossary Terms Extracted from Large-Sized Software Requirements using FastText,” in Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference, Jabalpur India: ACM, Feb. 2020, pp. 1–11. doi: 10.1145/3385032.3385039.
- [44] F. Dalpiaz and N. Niu, “Requirements Engineering in the Days of Artificial Intelligence,” IEEE Softw., vol. 37, no. 4, pp. 7–10, Jul. 2020, doi: 10.1109/MS.2020.2986047.
- [45] B. Wang, A. Wang, F. Chen, Y. Wang, and C.-C. J. Kuo, “Evaluating word embedding models: methods and experimental results,” APSIPA Trans. Signal Inf. Process., vol. 8, no. 1, 2019, doi: 10.1017/ATSIP.2019.12.
- [46] P. Vora, M. Khara, and K. Kelkar, “Classification of Tweets based on Emotions

- using Word Embedding and Random Forest Classifiers,” *Int. J. Comput. Appl.*, vol. 178, no. 3, pp. 1–7, Nov. 2017, doi: 10.5120/ijca2017915773.
- [47] S. Mishra and A. Sharma, “A Generalized Semantic Filter for Glossary Term Extraction from Large-Sized Software Requirements,” in *14th Innovations in Software Engineering Conference (formerly known as India Software Engineering Conference)*, Bhubaneswar, Odisha India: ACM, Feb. 2021, pp. 1–9. doi: 10.1145/3452383.3452387.
- [48] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching Word Vectors with Subword Information.” *arXiv*, Jun. 19, 2017. Accessed: May 02, 2023. [Online].
- [49] Available: <http://arxiv.org/abs/1607.04606>
- [50] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of Tricks for Efficient Text Classification.” *arXiv*, Aug. 09, 2016. Accessed: May 05, 2023. [Online]. Available: <http://arxiv.org/abs/1607.01759>
- [51] “Word representations · fastText.” <https://fasttext.cc/index.html> (accessed May 17, 2023).
- [52] C. D. Manning, “Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?,” in *Computational Linguistics and Intelligent Text Processing*, A. F. Gelbukh, Ed., in *Lecture Notes in Computer Science*, vol. 6608. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 171–189. doi: 10.1007/978-3-642-19400-9_14.
- [53] J. Pfeiffer, A. Rücklé, C. Poth, A. Kamath, I. Vulić, S. Ruder, K. Cho, I. Gurevych, Adapterhub: A framework for adapting transformers, *arXiv preprint arXiv:2007.07779* (2020).
- [54] S.-A. Rebuffi, H. Bilen, A. Vedaldi, Learning multiple visual domains with residual adapters, *arXiv preprint arXiv:1705.08045* (2017).
- [55] V. Sanh, L. Debut, J. Chaumond, T. Wolf, Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, *arXiv preprint arXiv:1910.01108* (2019).
- [56] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, *arXiv preprint arXiv:1907.11692* (2019).
- [57] A. Torralba, A. A. Efros, Unbiased look at dataset bias, in: *CVPR 2011, IEEE*, 2011, pp. 1521–1528.

- [58] J. Quiñero-Candela, M. Sugiyama, N. D. Lawrence, A. Schwaighofer, Dataset shift in machine learning, Mit Press, 2009.
- [59] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, arXiv preprint arXiv:1711.05101 (2017).
- [60] E. Alpaydm, Combined 5×2 cv f test for comparing supervised classification learning algorithms, Neural computation 11 (1999) 1885–1892.
- [61] J. Pfeiffer, A. Rücklé, C. Poth, A. Kamath, I. Vulić, S. Ruder, K. Cho, I. Gurevych, Adapterhub: A framework for adapting transformers, arXiv preprint arXiv:2007.07779 (2020).
- [62] S.-A. Rebuffi, H. Bilen, A. Vedaldi, Learning multiple visual domains with residual adapters, arXiv preprint arXiv:1705.08045 (2017).
- [63] V. Sanh, L. Debut, J. Chaumond, T. Wolf, Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, arXiv preprint arXiv:1910.01108 (2019).
- [64] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, arXiv preprint arXiv:1907.11692 (2019).
- [65] A. Torralba, A. A. Efros, Unbiased look at dataset bias, in: CVPR 2011, IEEE, 2011, pp. 1521–1528.
- [66] J. Quiñero-Candela, M. Sugiyama, N. D. Lawrence, A. Schwaighofer, Dataset shift in machine learning, Mit Press, 2009.
- [67] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, arXiv preprint arXiv:1711.05101 (2017).
- [68] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, arXiv preprint arXiv:1908.10084 (2019).
- [69] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, S. Fidler, Skip-thought vectors, in: Advances in neural information processing systems, 2015, pp. 32