Optimisation Method for Training Deep Neural Networks in Classification of Non- functional Requirements

Maliha Sabir

School of Engineering,

Division of Computer Science and Informatics

A Thesis submitted in partial fulfilment of the

requirements of London South Bank University for the degree of

Doctor of Philosophy

September 2022

Dedication

This is dedicated to my lovely husband, who has always believed in me, even when I didn't believe in myself.

(Maliha Sabir)

Acknowledgements

First and foremost, I thank Allah Almighty for providing me with the strength, health, and capabilities required to complete this thesis.

I'd want to show my gratitude to the following people, without whom I would not have been able to complete this study.

My esteemed supervisor, Prof. Ebad Banissi, for his tremendous support, meetings and chats with him was crucial in motivating me to look outside the box to produce a comprehensive and objective critique.

In addition, I'd want to thank Dr Mike Child and Dr Christos Chrysoulas for their guidance, review, and input on my study. I am grateful to all the panel members and the research board for enabling me to complete my studies at London South Bank University.

My heartfelt thanks to my parents (Sheikh Sabir Hussain and Saeeda Sabir) for everything they've done for me and my husband Mehtab Qureshi for his love, patience, and unwavering support. Finally, I'd like to thank my children Hashim and Parizah for never failing to inspire me.

Declaration

I confirm that the work provided in this thesis is the result of my original effort. I believe it does not contain any previously published or authored work unless an appropriate acknowledgement is stated in the text. I also declare that no substantial portion of my dissertation has been submitted, or is currently being submitted, for any degree, diploma, or other qualification at London South Bank University or any other University, except as stated in the Preface and specified in the text.

However, certain items have been presented at international conferences. Finally, the work adheres to the mandated word restriction set forth by the relevant degree committee.

(Signed) ..... Maliha Sabir

(Date).....20/09/2022

Abstract

Non-functional requirements (NFRs) are regarded critical to a software system's success. The majority of NFR detection and classification solutions have relied on supervised machine learning models. It is hindered by the lack of labelled data for training and necessitate a significant amount of time spent on feature engineering.

In this work we explore emerging deep learning techniques to reduce the burden of feature engineering. The goal of this study is to develop an autonomous system that can classify NFRs into multiple classes based on a labelled corpus. In the first section of the thesis, we standardise the NFRs ontology and annotations to produce a corpus based on five attributes: usability, reliability, efficiency, maintainability, and portability. In the second section, the design and implementation of four neural networks, including the artificial neural network, convolutional neural network, long short-term memory, and gated recurrent unit are examined to classify NFRs.

These models, necessitate a large corpus. To overcome this limitation, we proposed a new paradigm for data augmentation. This method uses a sort and concatenates strategy to combine two phrases from the same class, resulting in a two-fold increase in data size while keeping the domain vocabulary intact. We compared our method to a baseline (no augmentation) and an existing approach Easy data augmentation (EDA) with pre-trained word embeddings. All training has been performed under two modifications to the data; augmentation on the entire data before train/validation split vs augmentation on train set only. Our findings show that as compared to EDA and baseline, NFRs classification model improved greatly, and CNN outperformed when trained using our suggested technique in the first setting. However, we saw a slight boost in the second experimental setup with just train set augmentation. As a result, we can determine that augmentation of the validation is required in order to achieve acceptable results with our proposed approach. We hope that our ideas will inspire new data augmentation techniques, whether they are generic or task specific. Furthermore, it would also be useful to implement this strategy in other languages.

Table of Contents

List of Tables

List of figures

List of Equations

List of Symbols

D	Dataset
A_m	Measure to estimate the quality of the agreement among annotators
P_o	Probability of observed agreement
P_e	Probability of expected agreement
R	Set of real numbers
γ	Classification Function
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
{C1, C2, C3,….Cn}	Set of labels
N	Set of Natural numbers
log	Logarithm with base e
S	An arbitrary set
x	Input space
yˆ	Output space
∅	Learnable parameter
Q	Augmented dataset

$t^{\sim}$	Predicted Class label
R^M	Extracted Representation
$\{x^1x^2x^3, \ldots\ldots, x^N\}$	Set of requirements
$\emptyset_e and\ \beta_e$	Weights
$f(h_i)$	Activation Function
r	Reset gate
z	Update gate
σ_z	Classification function
j_c	Predicted class

List of Abbreviations and Acronyms

FRs	Functional requirement/s
NFRs	Non-functional requirement/s
RB	Rule-Based
ML	Machine Learning
NN	Neural Network
DNN	Deep Neural Network
NLP	Natural language processing
SL	Supervised Learning
USL	Un-Supervised Learning
ANN	Artificial Neural Network
CNN	Convolution Neural Network
GRU	Gated Recurrent Unit
LSTM	Long-Short Term Memory
CBOW	Continuous Bag of Word
DA	Data Augmentation
GSC	Gold Standard Corpus
SRS	Software Requirement Specification
EDA	Easy Data Augmentation
CDA	Custom Data Augmentation

List of Publications

The following papers are based on the research presented in this thesis. They have all been peer review and published.

- Sabir, M., Banissi, E., Child, M. (2022). Custom Data Augmentation Technique (A Deeper Insight). In: Rocha, A., Adeli, H., Dzemyda, G., Moreira, F. (eds) Information Systems and Technologies.
- Sabir, M., Banissi, E., Child, M. (2021). A Deep Learning-Based Framework for the Classification of Non-functional Requirements. In: Rocha, Á., Adeli, H., Dzemyda, G., Moreira, F., Ramalho Correia, A.M. (eds) Trends and Applications in Information Systems and Technologies.
- Sabir, M., Chrysoulas, C., Banissi, E. (2020). Multi-label Classifier to Deal with Misclassification in Non-functional Requirements. In: Rocha, Á., Adeli, H., Reis, L., Costanzo, S., Orovic, I., Moreira, F. (eds) Trends and Innovations in Information Systems and Technologies.

Chapter 1 Introduction

1.1 Overview

In software engineering, requirements are used to describe the framework for the software system along with any limits on the development process. The method of elicitation and formulation of those specifications is called requirement engineering. These specifications include a framework for consensus on what to do with the software system, and a basis for evaluation, verification, and enhancement. Requirement engineering process provides an estimation of implementation costs and schedules. Requirement analysts encounter two types of requirements: functional requirements (FRs), which describe the functions, tasks, or behaviours that a system must support and non-functional requirements (NFRs), which represent a system's particular qualities (Rahman, 2013).

Stakeholders use natural language to express their requirements. However, NFRs are abstractly evoked (Wilson, 1999). FRs are the main focus for the developers, while NFRs are left unaddressed (Barmi, 2011). In practice, various design and architectural decisions depend on NFRs (Matoussi, 2008; Felfernig, 2012), resulting in increased project failure rates (Rao, 2011; Rahman, 2013). According to Lawrence (2001), NFRs are rated as one of the ten most significant risks in requirement engineering. Many software systems have faced cost and schedule overruns due to poor handling of NFRs (Rao & Gopichand, 2011). Examples of poor NFR management include the London ambulance system (Finkelstein, 1996), the Starbucks electronic system (Miners, 2015), and electronic health records in England (Bertman, 2010), all of which failed due to issues with performance, dependability, and usability, among other things. The identification of NFRs is extremely important in the realm of requirement engineering. In practice, the discovery and analysis of NFRs is mostly a manual process that is time-consuming and subject to the will and interest of the client or developers. The field of requirement mining, on the other hand, has developed as an active area of research with a variety of automated and semi-automated techniques being offered for eliciting, analysing, and tracing FRs and NFRs.

Automatic identification and classification methods help to save time and effort for requirement engineers by reducing the amount of manual work required to process NFRs. This study proposes an automated classification of NFRs based on multiple classes using deep learning techniques. Timely detection of NFRs and complete identification of requirements will benefit stakeholders in a practical way, such as facilitating the prioritisation of requirements, better management of resources (which will help to achieve increased customer satisfaction), and more valuable and higher quality products, and ultimately, more substantial organisational competitiveness.

The remainder of this chapter is organised as follows. Section 1.2 describes the motivation for this study based on limitations of the existing solutions. Section 1.3 presents contribution of this study. The research methodology is described in section 1.5, and finally, the thesis structure is outlined in section 1.5.

1.2 Motivation

Today, the ability of machines to interpret and extract information from natural language is one of the central areas of research in artificial intelligence. Natural language processing (NLP) is a term used to describe a study that aids computers in solving problems, including text classification, machine translation, natural language generation, reading comprehension, and sentiment analysis.

Automatic requirement classification is becoming increasingly popular because it can save time invested in the manual labelling of requirements, thus increasing transparency in requirements engineering process. Software requirement classification differs from other forms of text classification, such as spam detection, language identification, and sentiment analysis, where the labels assigned to the text do not represent semantic information (Griffiths *et al.*, 2007). The labelling of software requirements, on the other hand, should be based on semantic characteristics since the label captures some semantic or topical information (Joachims, 2002).

Previously, this task has been addressed mainly using machine learning techniques. Traditional supervised machine learning algorithms necessitate a significant amount of feature engineering effort, which can be time-consuming. During the last several years, deep neural network (DNN) approaches have advanced to the point where

models based on neural networks may give state-of-the-art classification predictions using features that have been directly learned from data (LeCun et al., 2015; Sze et al., 2017). There is an increasing amount of work in this discipline with visual data.

The field of NLP is presently being overtaken by deep learning (DL) based solutions with the help of neural networks (NNs). However, these solutions have significant drawbacks. First, they generally require a significant amount of labelled data. Model overfitting and poor performance are also major concerns since the vast dynamic range of features may not be helpful in classification (Krizhevsky et al., 2017). The goal of this research is to investigate the concept of data augmentation in the pre-processing of DNNs to tackle the problem of data sparsity for the classification of NFRs and word embeddings in order to enrich the model with semantic knowledge. According to the notion of data augmentation, changing data to train a neural network can help the network perform better (Cagli et al., 2017). Data augmentation has proven successful in a variety of deep learning tasks ranging from image classification (Krizhevsky et al., 2017) to speech recognition (Graves, 2014; Amodei et al., 2016; Shorten and Khoshgoftaar, 2019). However, approaches that can be utilised for images and sounds are not acceptable for text due to the risk of losing the meaning of a sentence. This work primarily focuses on the augmentation of textual data and proposes a new data augmentation approach that helps to bridge this gap. Furthermore, in the case of data sparsity, transfer learning techniques have emerged as a suitable option (Perera and Patel, 2019). This study makes use of pre-trained word embeddings to train the DNNs.

The classification of software requirement is undoubtedly an area of interest not only in academia but also in industry. The findings of this study can be used as a reference or guideline for developers and researchers interested in this domain.

1.2.1 Shortcomings in Existing Solutions

In the literature, two standard approaches (i) a rule-based (RB) approach and (ii) machine learning (ML) techniques were discovered as viable options for requirement classification. RB techniques use a set of hand-crafted linguistic rules to group text into different classes. However, these techniques require in-depth subject knowledge and a significant amount of effort on the part of specialists to develop rules. It is more difficult to maintain and upgrade these systems. As the number of rules increases,

adding a new rule may render the current ones ineffective. ML-based techniques can address these challenges by using supervised ML algorithms to automatically derive rules from pre-labelled data. ML involves the collection of requirements, classifications, and the validation of these classifications. In relevant studies, SVM and Nave Bayes were extensively utilised in ML-based classification, and they have been demonstrated to produce good outcomes, outperforming RB approaches (Binkhonain and Zhao, 2019). However, these methodologies are impractical and have several shortcomings, including (i) the absence of NFR representation, (ii) the restriction of a domain corpus, (iii) the constraints of feature training, (iv) the inability to handle multiclass/multi-label classification, and (v) overfitting and generalisation.

1.3 Key Contributions

This thesis aims to design an optimal framework for the classification of non-functional requirements that includes a unique method of data augmentation, word2vec embeddings, and a deep learning model based on a newly created NFR corpus.

Figure 1- 1 shows the conceptual framework for this study. More specifically, the study's contributions are divided into two phases motivated by the size of training data necessary for deep learning-based systems and the limitation of a domain corpus.

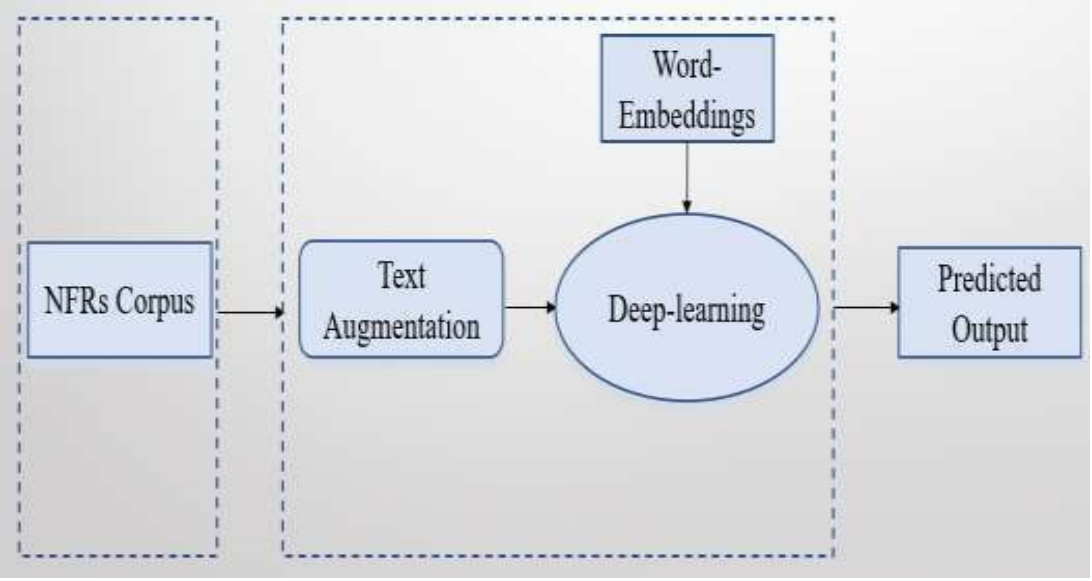

Figure 1- 1: Conceptual Framework for Classification of NFRs

1.3.1 Domain Corpus for Non-functional Requirements (NFRs)

This study proposes two approaches for creating a corpus of NFRs. 1) it seeks to construct a corpus dedicated to NFRs; to the best of our knowledge, this will be the first corpus in this field to include a representative sample.

As an extension to this, 2) it proposes a framework for developing a multi-label corpus for NFRs. Software requirements have not been explored as multi-label classifications, and one obvious reason for this is the unavailability of such domain corpus. Therefore, the proposed framework may be used as a resource for future research.

1.3.2 A Multi-Class Classification System for Non-functional Requirements

This research also presents a DNN-based methodology for dealing with multiclass classification problems. Usually, these models are trained with a huge commercial dataset; the idea is to benefit from the strength of these models with a small dataset. It would be interesting to discover which state of the art models are capable of learning with a smaller dataset. This can form a core reference point for future research in this field.

As evident from the literature, DNNs require a large corpus, which is practically difficult to obtain within the scope of this study. Therefore, this work explores another strategy to deal with data sparsity in the form of pre-trained word embedding and data augmentation to optimise the NFR classification system.

This research presents a unique data augmentation technique that enhances data size while retaining domain knowledge. It would be fascinating to investigate how pre-trained word embedding and a data augmentation technique affect the learning of the neural networks. The results of this study could benefit future researchers in drawing knowledge based on the impact of the adopted methodology.

1.4 Methodology

It has been determined that the research methodology will investigate and evaluate a series of techniques directed at the classification of non-functional requirements to achieve the research goal and provide a solution to the associated research issues, as described in the previous sections. The start point for the study is simply to apply

straightforward, well-established, standard DNNs for the classification task at hand. The purpose is to provide a benchmark against which other proposed techniques can be compared and assessed.

For the experimentation, the study will assess the performance of four state-of-the-art DNNs, artificial neural network (ANN), convolution neural network (CNN), long-short term memory (LSTM), and gated recurrent unit (GRU).

When it comes to the second strategy, it is also conjectured that using some data augmentation that synthetically generates more data and pre-trained embeddings that transfer knowledge learned from a previous large corpus to a domain of interest will help to train the classifier.

The study further creates a set of research objectives that will be addressed through experimentation and analysis to build and develop an efficient multiclass NFR classification system, as shown in Figure 1- 2.

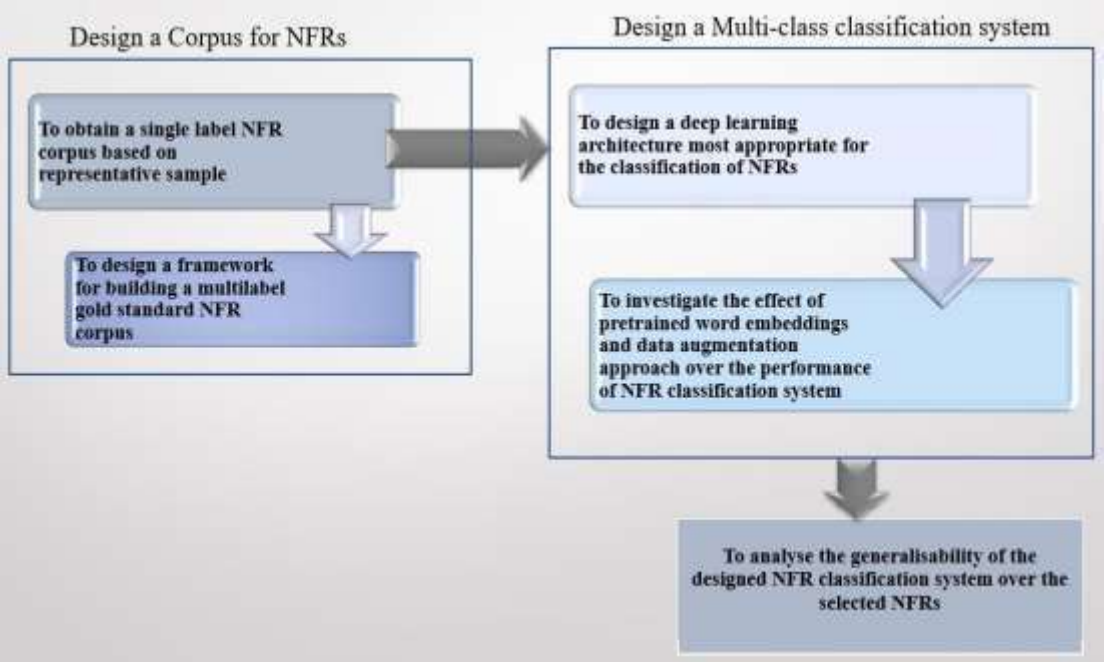

Figure 1- 2: A Flowchart for Framework Design

- To obtain a single label NFR corpus based on a representative sample.
- To design a framework for building a multilabel gold standard NFR corpus.
- To design a deep learning model most appropriate for the classification of NFRs.
- To investigate the effect of pre-trained word embeddings and data augmentation approach over the performance of the baseline NFR classification system.
- To analyse the generalisability of the designed NFR classification system over the selected NFRs.

The overall performance of the proposed framework will be evaluated using standard metrics used for multiclass classification problems, including precision, recall, and F1-score. The main findings will be analysed based on the research aim and objectives, comparison between neural networks, word embeddings, and data augmentation approach.

1.5 Thesis Outline

This thesis is organised as follows: **Chapter 2** provides some background on the classification of FRs/NFRs and current state-of-the-art techniques used in text classification. It provides an insight on the existing corpus in the NFR domain, the formal NLP rule-based approach, the machine learning approach, feature selection techniques, and finally, a discussion about the limitations of existing studies.

Chapter 3 addresses the first two research objectives and provides the theoretical background for defining NFRs. It discusses the representative sample for NFRs and, based on that sample, it creates an NFR corpus that is further utilised in the experiments in this study. It also presents a framework for obtaining a gold standard in multi-label NFR corpus annotation. Finally, the results for annotation are presented at the end of this chapter. **Chapter 4** provides the background for deep neural networks, specifically ANN, CNN, GRU and LSTM, for text-based classification, and the application of word embedding, and transfer learning are analysed for the given classification task. It discusses various hyperparameter tuning and overfitting issues in deep neural networks.

Chapter 5 consists of experiments based on the research questions of the proposed classification approach. It focuses on the experimental setting, corpus, word embedding and representation model. **Chapter 6** provides an insight into the data augmentation in the deep learning domain, extending to the practical implementation of the proposed data augmentation approach. **Chapter 7** presents the results and analysis based on the experiments performed in Chapters 5 and 6. Furthermore, **Chapter 8** discusses shortcomings of the current study and proposes future directions for potential improvements.

Chapter 2 Fundamentals of Automatic Classification for Non-Functional Requirements

Automatic text classification is a well-studied topic in the field of natural language processing. The development of NLP techniques has embarked in to the classification of functional and non-functional requirements to manage user specifications into pre-determined classes.

This chapter provides an in-depth study of the relevant literature. Section 2.2 provides step wise guide for the reader through the classification process in terms of dataset and feature selection strategies. It further explores the use of a rule-based method and machine learning approaches to classify requirements.

2.1 Related Work

Text classification was introduced as a research topic by Maron (1961) and has become an important issue in information science due to increased attention over the last two decades. Traditionally, text classification models have been trained with labelled data in a supervised setting. Text classification is the activity of assigning a label to a piece of text (document/sentence) from a predefined set of semantic tags (Zha & Li,2017). Different applications include indexing documents to regulate vocabulary, filtering irrelevant content, categorising web pages, email management, genre detection, and many others. Text classification is now a prevalent practice because most content is digitally generated and processed. Company and personal correspondence, scientific and entertaining posts, conferences, and patient data are just a few examples of electronic text collections. These enormous text data require automatic storage and retrieval methods provided by text classification.

The classifier's goal in text classification is to learn a classification function (or model). For example, consider a dataset of *'D'* documents and a set of *'C'* classes/labels, which makes a set of records of training datasets represented as *(D; C)*. This categorisation task might be single-label or multi-label. The single label can be further classified as

binary or mutually exclusive classes and multi-class classification. For example, *X* belongs to the dataset "*D*", and the class "*C*" represents a set of classes *C1* and *C2*. In binary, *X* can be classed as either a *C1* or a *C2*. In the multiclass scenario, however, there are the additional labels *C1, C2, C3, ... Cn*, and *X* can be classified in any but one of the supplied labels.

On the other hand, in multi-label categorisation, an instance may be associated with numerous labels at the same time. In the preceding example, *X* can belong to C_1, C_2, or more classes at the same time.

This section presents work related to the detection and classification of FRs and NFRs. Seventeen (17) studies have been reviewed for this analysis; ML techniques have been found in 13 selected studies, whereas four have adopted the RB approach. These algorithms fall into three types, comprising eight supervised learning (SL), three unsupervised learning (USL), and two semi-supervised learning (SSL) approaches, as presented in Table A- 1.

A noticeable contribution to identify NFRs from structured and unstructured documents was first presented by Cleland-Huang et al. (2006), who developed a classifier that employs a list of keywords to classify NFRs from requirement documents obtained from Siemens Logistics and Automotive Organization. In another study by Cleland-Huang et al. (2007), an improved iterative approach was presented to classify security, performance, and usability. After conducting a series of tests, the proposed NFR classifier was proved to be unable to identify all of the NFRs, yet it emerged as a starting point for various studies in this domain.

Hussain et al. (2008) suggested a supervised automated process of detecting NFR sentences by using a text classifier equipped with a part-of-speech (POS) tagger for both binary and multi-class NFR classification. The classifier makes use of syntactic features and keyword characteristics for training. The results reported in this paper outperformed the recent work in the field and achieved a higher accuracy of 98.56% using 10-folds-cross-validation over the same data used in the literature.

Another study by Zhang et al. (2011) employed n-grams, individual words, and multi-word expressions (MWEs) as index terms at various levels of linguistic semantics

features to identify FRs and NFRs. The classifier is thus an SVM with a linear kernel. Soon after that, Slankas and Williams (2013) built an NFR locator utilising KNN, SVM, and Naive Bayes methods. When using a word vector representation of the NFRs, an SVM was twice as effective compared to the Naïve Bayes classifier. Furthermore, a KNN classifier with a unique distance metric outperformed the optimal Naïve Bayes classifier. Misclassifications were made as a result of the tool, and generalisability concerns were also present. Slankas discovered that some NFR qualities are linked to particular traits.

A supervised learning-based approach for mining and classifying FRs and NFRs in agile software development was proposed by Sunner and Bajaj (2016). A genetic algorithm was utilised with a neural network to classify FRs and NFRs from multiple documents. The approach was compared with a model based on SVM, and the results indicated that the cluster neuro-genetic approaches outperform SVM and RBF kernels, according to the findings. However, there isn't any real-world application for this research. Thus, it can't be evaluated for usefulness.

Kurtanovic and Maalej (2017) developed and tested a multiple supervised machine learning technique based on meta-data, lexical, and syntactic characteristics, including usability, security, operational, and performance criteria. Many feature extraction and feature selection strategies were employed to improve the accuracy of classification algorithms by Abad et al. (2017). Using the tera-PROMISE repository, a study found that pre-processing an existing classification technique enhanced its performance.

Mahmoud (2017) proposed a semi-supervised approach for extracting software requirements based on an SVD model used with cosine similarity to classify FRs and NFRs. When working with the Tera PROMISE dataset, he used different representation models, including the trigram, bigram representations, and the LSA with cosine distance. He also tried out the TF-IDF model. Promising results were observed when cosine similarity was implemented with the trigram. Casamayor et al. (2010) proposed a semi-supervised technique based on user feedback, iteratively collecting data using an expectation maximisation strategy to develop a classifier for NFRs. For the initial training of the binary classifier, the technique utilises a multinomial naive Bayes classifier with expectation maximisation (EM) and claims to achieve

better results than the supervised approach discussed earlier.A semi-supervised semantic similarity-based technique based on the word2vec model and prominent keywords was developed by Younas et al. (2019) to automatically detect NFRs. The PROMISE-NFRs dataset was used for this investigation. NFRs identification based on semantics was a suitable starting point; however, the results are modest and require additional investigation, as the manual identification of indicator terms is not only labour intensive but also makes the detection of NFRs dependent on the keywords, so there are chances that it will not be able to produce the same effect with other datasets. Similarly, the results are dependent on the word2vec model, which is based on Wikipedia vocabulary.

With the use of a clustering algorithm, Mahmoud and Williams (2016) suggested an unsupervised technique that uses the semantic meaning of the texts in question to detect NFRs. A systematic analysis of a series of word similarity methods and clustering techniques was used to generate semantically cohesive clusters of FR words. These clusters were then classified into various categories of NFRs based on their semantic similarity to basic NFR labels. The results show that this approach worked well for FRs on data from SafeDrink, SmartTrip, and BlueWallet.

Recently Baker et al. (2019) conducted the first study to adopt a state-of-the-art ANN and CNN for the classification of NFRs. The PROMISE dataset was used in the experiment. Their dataset contained five classes. Their results show that CNN produced better results than ANN. However, inexperience and a lack of pragmatism are evident in the tool's design.

RCNL is a multilayer ontology established by Vlas and Robinson (2012) that utilises a keyword-based approach that uses generic English grammar at the lower levels for the discovery of requirements and uses two parsing schemes to implement the design. They compared the result of their study with that of Cleland (2006).

Wen ontology language was used to classify the requirement into ISO/IEC 9126 ontology classes by Rashwan et al. (2013) using an SVM. The ISO 9126 quality model was used by Singh et al. (2016) to identify and classify NFRs and their subcategories. Based on thematic role and NFR incidence, an RB classifier is proposed to classify NFRs in this work. Two corpora were used to examine the findings. The results

indicate there are more highly scored sentences in the PROMISE corpus than in the Concordia Corpus. Sharma et al. (2014) proposed a technique to detect NFRs based on syntactic and semantic elements by parsing the requirements. Here, the presence of certain combinations of relationships among the specified feature as pattern-based rules.

Using the requirement sentence-based classification algorithms of FSKNN, Badave et al. (2015) proposed an automated system for identifying non-functional requirements (NFRs) that incorporates semantic factors such as term development by hypernym and measurement of semantic relatedness between the term and each category of quality aspects based on ISO / IEC 9126.

The studies discussed in this section have revealed a general process for classifying NFRs. There are three primary stages to this method: 1) the text preparation phase, which entails pre-processing the requirements and extracting meaningful features from it; 2) the learning phase, which incorporates the RB or ML approach; and 3) the evaluation phase, which implies evaluating the approach of an ML algorithm to classify NFRs. All these phases of text classification are discussed in the following section in this chapter.

2.2 Corpus for Software Requirement

The term corpus (plural corpora1) was introduced by Sinclair (1998) and refers to an electronic collection of authentic texts or speeches produced and preserved in a machine-readable format by language speakers. Later, there was considerable discussion regarding the fundamental objective as well as the design criteria for a corpus. McEnery et al. (2006)[1] argue that it is a set of text-based standards, and it should be machine-readable, authentic, and representative of a specific language or variety. The term “automatic” refers to the classification of software requirements based on two common approaches: rule-based (RB) approaches and machine learning (ML) based techniques. The ML-based systems produce better results as compared to RB approaches.

[1] McEnery (2003) pointed out that corpuses is perfectly acceptable as a plural form of corpus.

However, previous studies lack theoretical evidence for including certain NFR categories for classification, and one open challenge is limitation of the NFR corpus. Only two datasets exist, namely, the PROMISE dataset (Cleland-Huang et al., 2007) and the Concordia corpus (Rashwan et al., 2013), which contain attributes of FRs and NFRs. The PROMISE corpus comprises 15 SRS documents created by MSc students at DePaul University. It has 255 FRs and 370 NFRs, which expand to the following categories: functionality, availability, fault tolerance, legal, look and feel maintainability, operational, performance, portability, scalability, security, and usability. In this dataset, the NFRs do not have equal training examples foe each attribute.

Table 2- 1: Class-wise Representation of the PROMISE Dataset

Requirements in the PROMISE Dataset	No. of Samples
Availability	21
Fault tolerance	10
Legal	13
Look and feel	38
Maintainability	17
Operational	62
Performance	54
Portability	1
Scalability	21
Security	66
Usability	67
Functional	255
Total Sample Size	625

Table 2- 1 shows the class-wise sample distribution in the PROMISE dataset; disproportionate samples in a dataset can create bias in the training phase. The NFR

characteristics that were chosen to be a part of the PROMISE dataset are not the critical NFR attributes that would be required to describe the whole domain of the dataset. Furthermore, this corpus has a misclassification when it comes to labelling requirements (Mahmoud, 2017). Each of the requirement sentences in this corpus is composed of a single type of requirement sentence. After some time, this corpus was upgraded and renamed as the improved PROMISE corpus to reflect the new information. On the other hand, in the Concordia corpus, NFRs are classified based on a requirements ontology, modelled using the Web Ontology Language (OWL). Within the intra-model dependence perspective, there are several distinct types of NFRs divided into sub-categories according to the ISO 9126 standard. Functional requirements (FR), external and internal quality (accessibility, accuracy, configurability, dependability, efficiency, functionality, maintainability, portability, reliability, security and usability/utility), constraints, and non-functional requirements are the four significant categories classified by the Concordia RE corpus, as shown in Table 2- 2.

Table 2- 2: Class wise Representation of Concordia Corpus

Doc.	NR	FR	CO	US	SE	EF	FU	RE	Total
1	59	17	26	7	1	1	0	1	112
2	114	32	20	7	10	1	1	2	187
3	180	54	14	6	8	4	1	1	268
4	191	19	21	0	2	3	13	5	254
5	213	23	13	8	2	5	1	0	265
6	1365	642	16	0	34	0	0	0	2057
Total	2122	787	110	28	57	14	16	9	3140

It is intended to be used as an annotation exercise for the entire SRS text that has been prepared, where the SRS document was used as input and then annotated on a sentence-by-sentence basis based on the categorisation described above. The Concordia corpus was created for a specific platform (the GATE annotation platform), which makes it undesirable to academics working on different platforms.

2.3 Text Preparation for Requirement Classification

The accuracy of the classification system is directly related to how clean and accurate data is used for training (Breck et al., 2019). Text pre-processing and feature selection are two procedures that are considered essential to obtain a refined dataset. This step takes textual requirements as input and applies different NLP techniques to pre-process the data. It is the first stage in dealing with datasets. According to Agarwal and Yu (2009), these steps involve techniques that help refine and clean the data to support the algorithm to speed up identifying keywords without disturbing the original syntax of the sentence. These can range from basic syntactic routine pre-processing to more complex semantic feature extraction approaches dependent on the requirement. Table 2- 3 shows the pre-processing techniques used in related studies and defines the most-used feature and techniques to cleanse the data. It includes syntactic features (i.e., stop word removal, stemming, lemmatisation). These ensure that documents are processed, non-alphabetic characters and mark-up tags are discarded, stop words are removed, and morphological stemming is performed. Most of the studies used more than one pre-processing operation on their data, as shown in the last column of Table 2- 3. Another technique used frequently in the related studies as part of speech (POS) tagging, categorised as advanced text categorisation pre-processing. This technique performs parsing (a syntactic procedure in which the POS information and dependencies of sentences are collected) to annotate the text into different segments like subject, verb, and object of the sentence.

Table 2- 3: Preprocessing Technique for Requirement Classification

Feature	Technique	Source
Tokenisation	Words, keywords, phrases, symbols	(Casamayor *et al.*, 2010; Kurtanovic and Maalej, 2017; Sunner and Bajaj, 2016)
Stop word removal	Removing words that do not convey any meaning	(Casamayor *et al.*, 2010; Kurtanovic and Maalej, 2017; Sunner and Bajaj, 2016)
Stemming	Convert into a root form	(Casamayor *et al.*, 2010; Hussain *et al.*, 2008; Kurtanovic and Maalej, 2017; Rashwan *et al.*, 2013; Sunner and Bajaj, 2016)
Lemmatisation	Removing inflectional endings	(Kurtanovic and Maalej, 2017; Sharma *et al.*, 2014; Slankas and Williams, 2013)
Temporal tagging	Selecting time, weight, and dates as features	(Abad *et al.*, 2017; Casamayor *et al.*, 2010; Kurtanovic and Maalej, 2017; Sunner and Bajaj, 2016)
POS tagging	Labelling as a subject, verb, or object to different parts of the sentence	(Abad *et al.*, 2017; Hussain *et al.*, 2008; Kurtanovic and Maalej, 2017; Vlas and Robinson, 2012)

Semantic features are another type of pre-processing that selects token or phrase-level features from the text to annotate (Haury et al., 2011). They remove unnecessary components from a text to improve the algorithm's execution speed as well as the classification accuracy and precision (Challita et al., 2016). Table 2- 4 provides the description of feature annotation and its adoption in different studies in the related literature. Like semantic role labelling, chunking, named-entity recognition, and N-gram appeared as the most commonly used techniques. These techniques are based on annotating the text on a single word or multiple words that are considered a representative feature of that text. In an N-gram model, each of the n consecutive tokens is regarded as a separate dimension. The N-gram was used to select the most meaningful word (unigram) or group of words (bigram or multi-word expression) that uniquely characterise the sentence. When each token is viewed as a different dimension, hyperspace is called a unigram. N-grams can be used alone or with dimensionality reduction methods (described in section 2.5). Compared to other semantic feature annotation strategies (Mahmoud, 2017; Zhang et al., 2011), N-gram was superior at detecting semantic dependence between words. At the same time, some unique features like a singular unit, phrasal unit (Kurtanovic and Maalej, 2017), and entity tagging were also seen in practice (Abad et al., 2017).

Table 2- 4: Feature Annotation Techniques for Requirement Classification. Table 2- 4 describes various feature annotation techniques and their functionality. It is significant to note that some of these techniques work at different levels of text, requiring the use of other techniques as a prerequisite for their effectiveness.

Table 2- 5: Feature Annotation Techniques for Requirement Classification

Scheme	Description	Pre-requisite	Annotation level	Source
Chunking	Delimiter based approach	POS tagging	Phrase level	(Vlas and Robinson, 2012
Semantic role labelling	Thematic role labelling	POS tagging	Syntactic labels	(Singh et al., 2016)
Ngram	Labelling one, two or three words in the sentence	POS tagging	Unigram, Bigram, or trigram	(Zhang et al., 2011; (Mahmoud, 2017)
Named entity recognition	Identification of keywords	POS tags, chunk tags, prefix, and suffix	Token level	(Hussain et al., 2008; (Abad et al., 2017)

2.3.1 Feature Selection Techniques for Requirement Classification

The basic idea of this process is to adopt a simple and efficient approach to reach a smaller but discriminative feature set. In the feature selection technique, a subset of original features is selected (Walowe Mwadulo, 2016). These selected components are then used to train and test the classifiers. For instance, in the case of POS tagging, only the tagged part of the requirement is used, and the remaining text is discarded. Similarly, in the N-gram, the part of the text that covers the N-gram is used for training.

This process then converts those selected features into vectors using statistical techniques. The association between each input variable and the target variable is evaluated using the statistic. The choice of statistical measures for both the input and output variables is dependent on the data type, and the input variables with the strongest correlation with the target variable are selected. This process reduces the training time and overfitting by preserving data characteristics for interpretability (Tomar and Agarwal, 2014). These are used in combination with the ML algorithm to improve the accuracy of classification (Varghese, 2012).

The feature selection can be either unsupervised or supervised. Unsupervised feature selection techniques ignore the target variable by removing redundant variables using correlation, whereas supervised methods are transformed based on three techniques. Wrapper, filter, and embedded techniques are used to extract features from the text in supervised methods. According to a comparative study by Wu (2016), the wrapper method selects features based on classifier performance. A predictive model is used to add one feature at a time in each round.

On the contrary, the filters extract features from the data without learning them. Instead, selecting the subset of features based on a user-specified threshold, assuming that features with a higher variance will have more useful information (Wu, 2016).

Still, it does not take account of the relationship between feature variables. In related studies, Zhang et al. (2011) used information gained to train the ML algorithm. Filter methods are generally faster than wrappers (Haury, 2011). Whereas, wrappers require cross-validation for each feature, making them computationally expensive (Khalid and Nasreen, 2014). By contrast, the filter approach is computationally efficient, as they work independently from the classifier.

Finally, certain machine learning algorithms automatically select features. These could be referred to as intrinsic or embedded feature selection methods. This includes techniques like Lasso's penalised regression model and decision trees, including random forest. Embedded methods learn which features contribute best to the accuracy of the model while the model is being created (Mirończuk and Protasiewicz, 2018) and reduces the degree of overfitting or variance of a model by adding more bias (Khalid and Nasreen, 2014).

In previous studies, Hussain et al. (2008) and Kurtanovi (2017) used a decision tree as a classification algorithm to create an embedded ability to find the best feature. Embedded approaches are faster than wrappers in computation, but they produce classifier-specific choices that may not work with any other classifier (Wu, 2016).

2.3.2 Dimensionality Reduction Techniques for Requirement Classification

After data cleansing and feature selection, the next step involves converting the text into a machine-readable format. Like feature selection, dimensionality reduction techniques reduces the number of random variables that are under consideration based on features extracted from the original feature set (Zena M. Hira and Gillies, 2015). The difference is that feature selection chooses which features to maintain or eliminate from the dataset, whereas dimensionality reduction generates new input features by projecting the data. As a result, dimensionality reduction is an alternative instead of a subset of feature selection. It performs transformation of the features so that the original features are not reversible, as some uncorrelated and superfluous information is lost (Varghese, 2012).

The related literature is analysed using either feature selection techniques or the dimensionality reduction techniques described below. Instead of the original values, the new collection of characteristics results in different values. This method can be further divided into linear methods and non-linear methods.

The extraction of features is performed so that the transformation of the original characteristics is not reversible, as some uncorrelated and unnecessary information is lost (Varghese, 2012). The most commonly used dimensionality reduction methods are those that apply linear transformations like those described in Table 2- 6. These include principal component analysis (PCA) and linear discriminant analysis (LDA), as well as the non-linear GDA and kernel PCA.

2.2.3.1 Principal component analysis (PCA)

According to Jolliffe and Cadima (2016), PCA is an unsupervised machine learning approach for finding the single best subspace of a given dimension using orthogonal transformation. PCA linearly maps the data to maximise its variance in the low-dimensional representation (Sorzano et al., 2010) by decreasing the number p of associated variables by a large factor to a smaller number k of orthogonal variable ($k<p$). The results of PCA depend on the scaling of the variables. The applicability of PCA is limited by certain assumptions made in its derivations.

2.2.3.2 Linear discriminant analysis (LDA)

LDA is typically used for multi-class classification. It is assumed that words with comparable meanings will appear in similar sections of text (Sun et al., 2017). LDA seeks a linear combination of input features that optimise class separability, whereas PCA attempts to discover a set of orthogonal components of maximum variance in a dataset. However, the data should be regularly distributed to avoid LDA constraints. The dataset should also contain known class labels; however, PCA does not require class labels.

2.2.3.3 Singular value decomposition (SVD)

It performs a transformation utilising truncated singular value decomposition (SVD). This works well with sparse data, where many rows have zero values. In contrast, PCA works well with dense data. Another significant distinction between truncated SVD and PCA is that SVD factorisation is performed on the data matrix, whereas PCA is performed on the covariance matrix.

2.2.3.4 Kernel PCA

Kernel PCA is the non-linear variant of standard PCA. Kernel PCA is beneficial for non-linear datasets where traditional PCA is ineffective.

In kernel PCA, input is initially passed through a kernel function, which temporarily projects the input into a higher-dimensional feature space and separates classes linearly. The data is then projected into a lower-dimensional space using the standard PCA algorithm. Kernel PCA converts non-linear data into a lower-dimensional space that may be used with linear classifiers in this fashion.

2.2.3.5 Generalised discriminant analysis (GDA)

GDA deals with nonlinear discriminant analysis using the kernel function operator. The GDA method converts input vectors into a high-dimensional feature space. Support vector machines are similar to the underlying idea (Baudat and Anouar, 2000).

2.2.3.6 Latent semantic analysis (LSA)/ Latent Semantic Analysis (LSI)

It is a method of examining relationships between a group of documents and the terms they include by producing a set of concepts connected to the documents and terms in NLP, specifically distributional semantics.

LSA assumes that words with similar meanings will appear in similar texts. A massive chunk of text is converted into a matrix where the rows show all the unique terms in the document and the columns represent each document. As a mathematical approach, SVD reduces the number of rows while maintaining the similarity structure within the columns. The cosine of the angle generated by any two columns is then used to compare documents. Closer values to 1 indicate very similar documents, whereas values closer to 0 indicate very different documents. Mahmoud (2017) used LSA with cosine similarity.

Table 2- 6: Dimensionality Reduction Techniques for Requirement Classification

Dimensionality Reduction	Feature Selection	Analysis	Source
PCA	Linear, Filter, Unsupervised learning	Orthogonal transformation	(Mahmoud, 2017)
SVD model	Nonlinear, Unsupervised/Filter	Probability model for taxonomy learning	(Hussain et al., 2008; Kurtanovic and Maalej, 2017)
LSA/LSI	Linear Unsupervised based on SVD	Latent semantic analysis is a technique for file representation and the cosine similarity measure	(Casamayor *et al.*, 2010; Mahmoud, 2017; Singh et al., 2016)
GDA	Non-linear	Feature projection into the low dimension	(Singh et al., 2016)
LDA	Linear, Unsupervised	Identify topics that the documents contain based on specific probabilities	(Kurtanovic and Maalej, 2017)

2.4 Training Machine Learning Algorithms for Requirement Classification

The training objective is to adapt the model to the training set while generalising new data. The dataset is separated into two groups for analysing the results of the supervised learning (SL) algorithm: the training set and the validation set.

The SL algorithms are trained with a training set and tested on a validation set that it has never seen before. A learning algorithm is prepared to learn from the training data and predict class labels for unseen data. The learning algorithm's performance is analysed based on its ability to generalise to the validation set.

2.4.1 Supervised Learning Approach

Supervised learning (SL), often known as inductive learning (Chen et al., 2014), is based on a methodology that involves training with labelled data to learn unique parameters (features, patterns, or functions). These algorithms find relationships between input and labels to optimise the classification accuracy of unseen data.

In general, SL classifiers build a mapping function that can be either parametric or non-parametric. A learning approach that summarises data using a set of fixed-size parameters is called parametric learning. On the other hand, non-parametric machine learning is an algorithm that does not make strong assumptions regarding the mapping function. Non-parametric methods seek to best fit the training data in constructing the mapping function (Kumar and Sahoo, 2012) while maintaining some ability to generalise to unseen data. It is unnecessary to select the appropriate characteristics; non-parametric techniques are the best option in situations where feature selection is difficult. They differ from the parametric model in that they make assumptions that are not dependent on the training data.

Naïve Bayes can be parametric or non-parametric depending on how the probability's densities are estimated and represented. Decision trees (DTs) and K-Nearest neighbour (KNN) take the non-parametric approach as such; they can fit a large number of functional forms, whereas support vector machines (SVMs) and simple neural networks (NNs) are parametric depending on what we are learning regarding the values of the network parameters or the high dimensional hyperplane.

2.4.1.1 Naïve Bayes (NB)

According to Lewis (1998), the Bayes theorem is the heart of the NB algorithm. With the support of statistical functions, it underpins a simple but powerful approach with the assumption of independence between features given the class label. It calculates

the likelihood of an input that is important to a particular pre-defined class (Feng et al., 2013; Berrar, 2019; Barber, 2011, pg-203). The output of NB is based on a Bayesian probabilistic model that assigns a posterior class probability to an instance: $P(Y=y_j|X=x_i)$. An instance is assigned to the class with the highest posterior probability.The simple NB classifier uses these probabilities to assign an instance to a class. This equation calculates the probability of Y based on the input features X.

$$P(Y|X) = P(X|Y) * P(Y)|P(X) \qquad \textbf{Eq2- I}$$

The purpose of NB is to pick the most likely class Y. Argmax is an operation that finds the argument that gives the maximum value from a target function.

NB can take on one of three types in practice, multinomial, Bernoulli, or Gaussian. Multinomial NB assumes that each $P(x_n|y)$ follows a multinomial distribution and is mainly used in classification documents to look at the frequency of words. Bernoulli is similar to multinomial, except it works with Boolean problems. Gaussian NB is based on the assumption that continuous values are samples from Gaussian distribution. NB holds strong assumptions about feature independence, which sometimes causes overfitting and an increased computation time (Murphy, 2012, pg-84). This algorithm has been used in three of the related studies in this domain (Cleland-Huang et al., 2007; Casamayor et al., 2010; Slankas and Williams., 2013).

2.4.1.2 Expectation maximisation

Expectation maximisation (EM) is an unsupervised clustering algorithm. It is an iterative approach to calculating maximum likelihood estimation in problems with missing data using probabilistic functions (Kamal et al., 2006), especially when some data items remained unobserved in the first place. EM works iteratively in two steps, the expectation step (E) and the maximisation step (M). EM is used as an alternative to gradient descent. In a related study, Casamayor (2010) used requirements that received feedback from the analysts as labelled requirements in combination with EM and NB to classify NFRs.

2.4.1.3 Decision tree (DT)

A decision tree is one of the classification techniques. Three types of nodes make up an established tree: the “root node”, the “internal node”, and the leaf to depict each possible result of a decision making in each possible outcome. In DTs, nodes represent features (attributes), branches represent a decision (rule), and leaf nodes represent outcomes (discrete and continuous).

A logic-based algorithm is used to model datasets in hierarchical structures using an if/else argument (Baharudin et al., 2010). The algorithm separates a set of data into smaller subsets for training while also building an associated decision tree incrementally. The root node, also known as the top node, is the best predictor for a tree’s decision-making when there are no incoming or outgoing edges. On the other hand, internal nodes have at least one incoming and outgoing edge. A leaf node with no outgoing edges indicates a categorization or final judgment.

Every node in the tree is made up of either decision nodes that contain words or “leaves” that correspond to the split point that yields the greatest information gain (IG) for a particular set of data (Gini or entropy in this example) (Dhurandhar and Dobra, 2008). The branches carry the weight of each term in the document.

The deeper the tree, the more complicated a DT may be in its decision rules, and the more fit the model is to the real world. In addition, the complexity of a tree will tend to affect the correctness of a choice made by the tree when it is used to make decisions. DTs are much more convenient for making classifications involving decision-making because they can compute both categorical and numerical data, are easily accessible and interpretable, require less calculation, and are capable of illustrating whether the relationship between dependent and independent variables is computationally low-cost. From the related literature two studies were found based on and decision tree (Hussain et al., 2008; Lu and Liang, 2017).

2.4.1.3 Support vector machine (SVM)

The SVM algorithm is based on statistical learning theory and the structural risk minimisation principle from the Vapnik–Chervonenkis (VC) dimensions. These statistical functions are referred to as the kernel. Different SVM algorithms use

different kernel functions that project the data into a higher dimensional domanial space, where it is more likely to be linearly separable.

In their most basic form, SVMs learn linear functions by constructing a hyperplane with the most considerable distance to any class's nearest training-data point (Vapnik and Lerner, 1963). The goal of structural risk minimisation is to establish a hypothesis 'h' by defining a notion of similarity, even in very high-dimensional domains, with little computing cost.

SVMs have the potential to generalise from the hypothesis space in the presence of a range of features because the complexity of the hypothesis is controlled using the margin rather than the number of features. As SVMs are trained by selecting support vectors that are further apart, they are independent of the dimensionality of the feature space. It suggests a heuristic for selecting good parameter settings for the learner. SVMs can, however, be used to learn polynomial classifiers, basic radial function (RBF) networks, and three-layer sigmoid neural nets by simply plugging in an appropriate kernel function. Text classification can be done with a linear p lines kernel in one dimension. Compared to naïve Bayes, SVMs are robust when working with high-dimensional data.

SVMs are considered shallow architectures, as they typically apply only one or two (non-linear SVM) transformations on top of their inputs; this limits their ability to capture hidden relationships. In related studies, SVM is identified as the most useful technique for classifying requirements (Zhang et al., 2011; Rashwan et al., 2013; Slankas and Williams, 2013; Sunner and Bajaj, 2016;).

2.4.1.5 Nearest neighbours (K-NN)

The K-NN statistical technique is utilized to calculate the correlation between the test data and the new instance to determine the nearest data point. As the name suggests, it uses K nearest neighbours (data points) to estimate the class or continuous value for the new data point (Baharudin et al., 2010). The data points with the shortest distance in feature space from a new data point are used to make decisions where K is the number of such data points considered during the implementation of the algorithm. As a result, while utilizing the KNN method, the distance metric and the K value are two key factors.

The most-used distance measure is Euclidean distance. Others include hamming distance, Manhattan distance, and Minkowski distance, depending on the problem.

When predicting a new data point's class/constant value, it uses all the data points in the training set to locate the new data point in the K nearest neighbours and their class labels in feature space. Finally, the class that is selected is the one that obtains the majority of data points from K's nearest neighbour. In the related literature, three studies used KNN for classification, but Casamayor et al. (2010) received the highest accuracy among the three, with 70% accuracy in classifying NFRs (Casamayor et al., 2010; Slankas and Williams, 2013; Badave et al., 2015).

2.4.1.6 Neural networks (NN)

Neural Networks typically form a layered architecture where each layer links with the previous layer to capture relationships in the inputs of the last layer (Mcculloch and Pitts, 1990). These layers carry some weights that multiply their information, and the connected layers transform the calculated inputs to produce a correspondent output. The neural network has not yet been introduced to effectively classify NFRs. Neural networks were used in this domain for the first time by Sunner and Bajaj (2016), but the study is naïve and lacks practicality.

2.4.2 Unsupervised Learning Approach

An ML algorithm draws inferences from the data by clustering data into different clusters without labelled responses (Usama et al., 2017). These algorithms use input requirement documents to drive structure by looking at the inputs' relationship. K means and hierarchical clustering are used within the domain of this study.

2.4.2.1 Hierarchical clustering

Hierarchical clustering is an iterative algorithm that forms a large cluster by merging similar elements in a dataset into a cluster that ends in a set of clusters. Each of the clusters is distinct from the others. However, the objects within each cluster are broadly similar to each other. Its training takes each observation as a separate cluster and produces a dendrogram based on a distance function that shows the hierarchical relationship between them. It then combines similar objects to form a cluster iteratively

until all the clusters are merged. Mahmoud and Williams (2016) used a hierarchical clustering algorithm using lists of keywords, including pre-defined NFR categories, to determine which cluster belonged to which NFR category. On the other hand, Abad et al. (2017) performed training without pre-defined data and got the worst performance compared to results obtained from supervised learning studies.

2.4.2.2 K-Means

K-means is an iterative technique that divides documents into clusters (K) based on their random classification. The K-means function is used to iteratively assign the nearest cluster of K-clusters to each data point in space using the K-means function. Sunner and Bajaj (2016) conducted an experiment using K means feature weighting and found that the results were stronger than those obtained with SVM.

2.4.3 Semi-Supervised Learning Approach

The semi-supervised learning approach contains approaches for supervised and unsupervised learning in one framework. This sort of learning uses a small amount of labelled data and many unlabelled data during the training phase. Previously labelled samples are mixed with unlabelled examples to assign labels. This unlabelled data compensates for the impact of a lack of labelled data on classifier accuracy. These solutions iteratively use the SL algorithm for both labelled and unlabelled data. The labelled data was used to train the classifier and then applied to more unlabelled criteria to determine classification accuracy. Two of the three primary studies used this algorithm, whereas the others used another technique.

2.4.4 NLP Rule-Based Approach

The classic approach in natural language processing (NLP) for text classification is rule-based (Afrin and Litman, 2018). RB classifiers classify data using a collection if /then rule (Kang et al., 2013). The rule is an expression made of the conjunction of characteristics. The rule consequent is based on a positive or negative classification and is directly learned from the data (Xu et al., 2017). It is similar to DT, where collections of rules model the dataset. However, RB classifiers allow overlaps in

decision space, while DT is strictly hierarchical. The RB approaches are easy to interpret and give the owners complete control of adding new rules or removing existing rules. These approaches require deep domain knowledge and extensive effort in order for experts to curate rules. However, these systems are harder to maintain and improve once the size of rules grows. Adding a new rule could lead to the ineffectiveness of existing ones. It is seen that there are limited rules to define the relationship between words, specifically in the context of handling security, usability, and maintainability (Sharma et al., 2014), which can result in the wrong classification. Moreover, the rule-based approach has limited generalisability (Vlas and Robinson, 2012).

Table 2- 7: Feature Learning Techniques for Machine Learning Classifiers

Learning Techniques	Algorithm	Analysis	Source
Information gain	Decision tree	Measures the reduction in entropy	(Zhang *et al.*, 2011)
Distance function	clustering filter, KNN	Sequential minimal optimisation (SMO)	(Rashwan *et al.*, 2013; Sharma *et al.*, 2014)
Expectation maximisation	Wrapper/unsupervised Naïve Bayes	Expectation–maximization (EM) algorithm is an iterative method to find the maximum likelihood. Estimation in problems with missing data.	(Kurtanovic and Maalej, 2017)
Unweighted Pair Group Method with Arithmetic mean (UPGMA)	Hierarchical clustering method, K-means	Hierarchical clustering algorithms Phylogenetic reconstruction dealt	(Singh et al., 2016)
Smoothed /Unsmoothed probability measure (SPM)/UPM	Decision tree	Probability measure to rank features	(Hussain *et al.*,2008)
TF/IDF	Rule-based	Latent semantic analysis is a technique for file representation and the cosine similarity measure. TF-IDF and bag-of-words are methods to represent a document as a vector.	(Casamayor *et al.*, 2010; Singh 2016; Mahmoud, 2017)
Kernel function, SMO	SVM	The kernel is a statistical function that takes data and converts it into the desired format.	(Slankas and Williams, 2013)

2.5 Requirement Classification and Performance Evaluation

Once text pre-processing is performed and a classification algorithm has been selected, the next phase is to train the algorithm for the given dataset. The performance of the ML algorithm is evaluated using a validation set, which is generally distinct from the training set. Most studies in the related literature employed K cross-validation procedures to validate the classifier performance (Hussain et al., 2008; Baker et al., 2019). K-cross validation is a typical method that divides data into distinct K-folds at random. The data size of each K-fold is the same; one is utilised for testing, while the others are used for training. The learning process is repeated K times to generate a single result, after which the average of the K results is calculated. Most of the studies used 60% of the dataset for training and 40% for testing. On the other hand, unsupervised learning research used the same data for testing and training.

A classification model predicts the probability of belonging to a specific class for each data item in the validation set. In the case of binary classification, a threshold is usually applied to decide which class has to be predicted for each item. While in the multi-class case, there are various possibilities; among them, the highest probability value is selected mostly based on a mathematical function, for instance, softmax.

Performance measurements are required to assess the algorithm's ability, such as precision, recall, accuracy, and F-score. These metrics are based on the confusion matrix, since it encloses all the relevant information about the algorithm and classification rule performance. The confusion matrix is a cross table that records the number of occurrences between two raters, the true/actual classification and the predicted classification. In the binary case, it only considers the positive class (meaning the true negative elements have no importance), while in the multi-class case, it considers all the classes one by one and, as a consequence, all the entries of the confusion matrix. The details of these techniques are given below.

Precision

Precision refers to the number of relevant records retrieved from the total number of irrelevant and relevant records. In the context of text classification, true positive represents the text classified as positive by the model that actually belonged to that class, while false positives are the elements that have been classified as positive by the model but that are actually negative and do not relate to the given label.

$$Precision = \frac{TP}{TP + FP} \qquad \textit{Eq2- II}$$

Recall

Recall is a measure of classification that is referred to as an accurate positive rate. In recall, a false negative is a piece of text that has been classed as negative by the model that is actually positive. In other words, the model fails to find relatedness with a label as true. Recall demonstrates the ability of the model to find the positive units in the dataset. This is essential, as classification aims to recognise all requirements.

$$Recall = \frac{TP}{TP + FN} \qquad \textit{Eq2- III}$$

Accuracy

Accuracy is a primary criterion for evaluating multi-class classification performance. It is calculated straight from the confusion matrix and represents the correct overall predictions by the model on the entire dataset. Accuracy is calculated as the fraction of true positive components divided by positively predicted units. True positives are the elements that the model has labelled as belonging to the positive class; simultaneously, a true negative is an outcome where the model correctly predicts the negative class. True positives and true negatives are the elements correctly classified by the model and lie on the main diagonal of the confusion matrix. The denominator also considers all the elements outside of the main diagonal that have been incorrectly classified by the model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad \textit{Eq2- IV}$$

F1 score

F1 score evaluation measures combine precision and recall by providing an average of the two values. The F1 score is widely used for measuring performance for multi-class classification. F1 score lies between 0-1, where 0 is the worst value and 1 is the best value.

$$F1 - score = 2\left(\frac{P * R}{P + R}\right) \qquad \textit{Eq2- V}$$

Table 2- 8: Analysis of the Performance Measures for the Classification of Requirements

Precision	Recall	F1-Score	Study
12.40%	76.70%	----	(Cleland-Huang *et al.*, 2007)
97%	1%	----	(Hussain *et al.*, 2008)
97.80%	100%	----	(Rashwan *et al.*, 2013)
72.80%	54.40%	62%	(Slankas and Williams, 2013)
73.9%	54.7%	----	(Ramadhani *et al.*, 2015)
98%	96%	----	(Singh et al., 2016)
86.86%	86.97%	----	(Sunner and Bajaj, 2016)
70%	70%	0	(Kurtanovic and Maalej, 2017)
98.00%	98.00%	98%	(Abad *et al.*, 2017)
92.85%	81.25%	86.66%	(Mahmoud, 2017)
50%	41%	42%	(Younas *et al.*, 2019)
82%-94	76%-97%	82%-92%	(Baker *et al.*, 2019)

Table 2- 8 describes participating studies' performances in terms of precision, recall, and F1-score. Following a thorough examination of the literature, it was found that POS tagging and Ngram were considered to be best among all other features. In a comparative study, one-word features resulted in higher recall for classifying NFRs. Kurtanovic and Maalej (2017) compared their work with Cleland-Huang et al. (2007), using a keyword-based approach to achieve a precision of 0.974. At the same time, Hussain et al. (2008) outperformed the result of Cleland-Huang et al. (2007), achieving an accuracy of 98.56% using the same data. Slankas and Williams (2013) proposed an NFR locator that effectively extracted relevant NFRs. SMO turned out to be highly effective when compared with Naïve Bayes. However, the SVM was seen as the best classifier among all studies.

2.6 Summary

This chapter presented an overview of the significant components of the requirements classification system that were pertinent to this thesis. This included a brief explanation of requirement classification, emphasising single label and multi-label classifications. Following that, we discussed the various models used to identify and categorise functional and non-functional requirements. Moreover, we fully explained the text categorisation technique, followed by a more in-depth examination of the various forms of machine learning and rule-based training. The chapter came to a close with an analysis of the various classification systems in use.

Chapter 3 Corpus Design for Non-Functional Requirements

Most state-of-the-art solutions for non-functional requirement classification are based on supervised machine learning models trained on manually annotated samples. However, these strategies have drawbacks like 1) a lack of theoretical representation for NFRs and 2) the unavailability of an open-source domain corpus. This chapter tries to close this gap by addressing these issues, as one of the contributions of this thesis.

The chapter starts with defining the ontology for NFRs in the context of requirement engineering and highlights some of the challenges faced when handling these requirements. Section 3.2 presents a single label NFR corpus. It is further extended in section 3.3 to design a crowd-based framework for corpus annotation to create a multilabel gold standard corpus. The formulation of the experiment is shown in section 3.4.

3.1 An Ontology for Non-Functional Requirements

Ontology is a term that refers to a common understanding of any domain of interest. It is commonly envisioned as classes (concepts), relations, functions, axioms, and instances. In requirement engineering (RE), NFRs are considered conditions over functional requirements. Some descriptions to express NFRs have been chosen for this study.

- “The functions and services that the system offers come with certain limitations. These include time, standards, and development” (Sommerville, 2007).
- “A description of a software system's characteristic or feature that a should demonstrate or a limitation that it should adhere to, except an observable behaviour” (Wiegers and Beatty, 2013).
- “NFRs are the requirements that pertain to a quality concern that the functional prerequisites do not cover” (Pohl and Rupp, 2015).

Quality requirements, quality attributes, non-behavioural requirements, “ilities”, or soft objectives are all terms used to describe NFRs (Chung, 2000).

It also suffers from both terminological and theoretical conflict, making them contradictory and synergistic. For example, Roman (1985) defined NFRs as constraints on the interface, performance, operation, life cycle, finances, and politics. According to Chung et al. (2000), NFRs are defined by more than 150 terms. Douglas (2010) defined NFRs in terms of their performance, compatibility, and secrecy, as well as their efficiency, flexibility, reliability, integrity, maintainability, portability, and usability.

As seen, there is an ambiguity in defining NFRs; similarly, there is no standardised method for determining whether or not NFRs have been met. Instead, they are often met to varying degrees of success.

Philosophers and researchers have been highlighting NFRs as an essential entity for consideration. However, there is a long-persistent debate among them regarding the NFRs considered essential for the success of a software project.

Each NFR attribute impacts software systems at some stage, leading to many architectural decisions (Felfernig et al., 2012) and requiring expertise, tools, and resources (Zhang et al. 2013). This diversity of NFRs indeed leads to a divergent classification of NFRs, which is practically not possible. Therefore, this study attempts to find a sample for the representation of NFRs. In the following section, the NFR hierarchy will first be identified. Then, we will investigate critical NFRs based on the software quality model’s perspective.

3.1.1 Mapping User Requirements into Non-functional Requirements

Stakeholder requirements are written in natural language, detailing the services and limitations they expect their system to provide (Sommerville, 2007). User requirements are unclear and can be perplexing. As a result, these are transformed into system requirements, which comprise a complete description of the system services, technical specifications, and design descriptions, as well as any conditions or constraints. Functional requirements are more specific and to-the-point than system

requirements, describing how a system should respond to a certain action or input. Every functional need has one or more non-functional requirements connected with it. Non-functional requirements might arise from the fulfilment of one functional requirement, whereas constraints are limits on the conditions (Sommerville, 2007). To elaborate further, a user requirement is provided as an example in Table 3- 1.

Table 3- 1: Mapping User Requirements into FRs, NFRs, and Constraints

Requirement Type	Example
User requirements	Any user who selects an option on the page should only make one click.
System requirements	The system should be interactive.
Functional requirements	Users should be able to select an option on the page.
Non-functional requirements	Efficiency
Constraints	Number of clicks

3.1.2 Challenges Faced by NFRs in Requirement Engineering

Stakeholders specify the design of the system and the limitations to the development process in the form of requirements described in natural language (Wilson et al., 1997). Stakeholder requirements could have properties of FRs and multiple different NFRs simultaneously (Dabbagh and Lee, 2014). The requirement analyst must identify those aspects and transform the operational need into a complete system specification and document in a formal software requirement specification document (SRS) (Cabral and Sampaio, 2008). The elicitation and formulation of the aforementioned specifications are called requirements engineering (Pohl and Rupp, 2015). All requirements are refined and documented clearly in those SRS documents that contain FRs and NFRs belonging to a software system. It works as a framework for consensus on what to do with the software system, a basis for evaluation, verification, enhancement, and estimation of implementation costs and schedules.

These requirements are interdependent with each other and may result in structural interdependencies, costs, or value interdependencies (Dahlstedt and Persson, 2003). There are also associations where one NFR helps to ensure another NFR. On the other

hand, one requirement will clash with another if they cannot coincide. Decisions made against one criterion have either a favourable or negative effect on other needs. (Tabassum et al., 2014). Prior to incorporating these needs into the design process, it is critical to identify what is essential (Felfernig et al., 2012). However, it appears that FRs are the primary focus, with NFRs being ignored (Barmi and Ebrahimi, 2011), which results in increased project failure rates.

3.1.3 The Role of Corpus in NLP

Automatic classification of software requirements has emerged as an increasingly exciting activity over the last decade through the use of machine learning techniques and rule-based approaches. These tasks largely depend on the domain corpus, a set of machine-readable text, and it should be authentic and representative of a specific language/domain (McEnery and Gabrielatos, 2006). Mainly supervised machine learning tasks require an annotated corpus (Tomanek et al., no date), as the evaluation of algorithms are based on those meaningful annotations (Silberztein, 2020). Human annotators add new information into the raw data (Hovy and Lavid, 2010) based on linguistic theory and guidelines (Pustejovsky and Stubbs, 2013). The recent trends in corpus development in NLP (Akhondi et al., 2014; Deleger et al., 2014; Mitrofan et al., 2018) suggest that an annotated corpora of gold standards are required for the development and evaluation of NLP systems (Mitrofan et al., 2018). Data is annotated independently by more than one annotator, and an inter-annotator agreement is computed to ensure quality (Wissler et al., 2014). This can help to minimise inconsistency and noisy annotation (Bhowmick et al., 2008) and improve supervised learning algorithms' performances (Zhao and Zhao, 2019). This chapter aims to define a representative sample for NFRs to create: 1) A single-label annotated NFR corpus and 2) A gold standard multi-label NFR corpus. The diversity of NFRs indeed leads to a divergent, practically impossible classification of NFRs. Therefore, it is vital to find critical NFRs that are considered necessary for the success of most software systems. This study uses software quality models to draw this sample and creates a single label representative NFR corpus based on requirements extracted from the SRS documents. This chapter further proposes a framework to create a gold standard multi- label NFR corpus that could identify various NFRs embedded within one requirement

and later be used to train a multi-label classifier system. Recently, crowdsourcing has emerged as a prevalent practise for this purpose. This practice gathers workers to one place where they can annotate the corpus according to the given research problem (Geiger, 2011) via dedicated platforms (Ghezzi et al., 2018).

3.2 Single Label NFRs Corpus Design

This section addresses the first objective of this study, which is to obtain a representative corpus for Non-functional Requirements. It begins with investigating the NFRs from the perspective of the software quality model to find the most critical NFR attributes.

3.2.1 Sampling for NFRs

To measure quality, various studies have put forth their quality measurement models. The most popular of them are the quality model of Boehm (Boehm et al., no date), the quality model of McCall (McCall et al., 1977), the quality model of FURPs (Florac et al., 1997), the quality model of Dromey (Dromey, 1995), and the ISO 9126 model, which was revised in 2007 and renamed ISO 25010 (ISO/ IEC CD 25010. 2008). These quality models are considered the basic models. To select the representative sample that could be generalised to the whole domain (Biber, 1993 a.p.244), the common NFRs in these models are conceived and described in Table 3- 2.

Table 3- 2:A Comparative Representation of Software Quality Models

NFRs	Boehm	McCall	ISO9126	FURPS	Dromey	ISO25010	Ranking
Reliability	1	1	1	1	1	1	6
Usability	0	1	1	1	1	1	5
Portability	1	1	1	0	1	1	5
Maintainability	1	1	1	1	1	1	6
Efficiency	1	1	1	1	0	1	5

The terms NFR and quality attributes are used interchangeably. All NFRs are listed in the left-most columns. While the sw quality models are mentioned in the top row, the remainder of the table contains entries against those NFRs in the cells. The availability of an NFR against a model is indicated by a "1", whereas the absence of an NFR is indicated by a "0". The NFRs common in at least five out of these six models have been selected as the sample in this study. Five NFRs, namely, reliability, usability, maintainability, portability, and efficiency are recognised as a representative sample from these quality models.

Table 3- 3: Definition of Non-functional Requirements

Category	Definition
Efficiency	To be able to perform the same functionality every time under any conditions.
Maintainability	To be able to make changes in a system in the future.
Portability	To use the system from one platform (hardware/software) to another.
Reliability	To be able to perform a specified function for a specified period.
Usability	To use a system with ease, effectiveness, and a user-friendly manner.

The selected NFR attributes are defined in Table 3- 3. These definitions will be used later in the study to train the annotators to achieve a gold standard annotated corpus.

3.2.2 Data Collection for Corpus

The first-hand data collection directly from stakeholders involved various limitations and ethical issues. Therefore, the software requirement specification (SRS) document is used as a data source. These SRS documents were downloaded from an online repository called SCRIBD with paid access. These requirements expand to five representative NFRs: efficiency, reliability, portability, maintainability, and usability. The requirements have been manually extracted from SRS documents. Moreover, SRS documents related to different software system domains are selected, providing diversity in the vocabulary used to define the requirements. Nevertheless, this selection is dependent on the available SRS documents on that online platform. According to our interpretation, this is the first dataset in the NFR domain designed on representative

samples (Pustejovsky and Stubbs, 2013). The table given below describes the requirement samples in our NFR corpus, which we named custom NFR corpus. Table 3- 4 provides the class-wise distribution of NFR samples for this corpus.

Table 3- 4: Class-Wise Distribution of Requirements in Custom NFR corpus

Category	Samples
Efficiency	480
Maintainability	240
Portability	156
Reliability	191
Usability	417
Total number of Samples	1484

As an output of this stage, it delivers a representative domain corpus for NFRs containing 1484 samples of requirements related to five NFRs: efficiency, maintainability, portability, reliability, and usability.

3.3 Gold Standard Multi-Label NFR Corpus Design

The development of the gold standard corpus is performed systematically and consists of three phases. It starts with the development of annotation guidelines, followed by the recruitment of annotators, and finally, an evaluation of the outcomes. To supervise the corpus annotation, a MAMA framework from the MATTER-MAMA has been used, as shown in Figure 3- 1 (Pustejovsky and Stubbs, 2013). MAMA (Model-Annotate, Model-Annotate) is an iterative process in which the annotators provide guidelines to perform annotations. These guidelines can be revised based on the annotation results until the required quality is achieved.

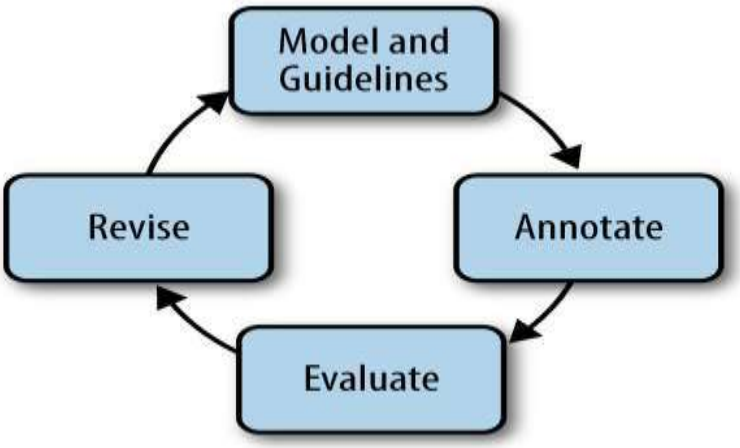

Figure 3- 1: MAMA Framework

The word "guidelines" Dipper et al.,(2004) applies to all knowledge, including linguistic theory, a derived model of an interesting phenomenon that describes how the specification should be applied to linguistic artefacts (Pustejovsky and Stubbs, 2013). A data item may be ambiguous and can fall into many categories. The annotations can be performed at different levels (Gries and Berez, 2017). However, in this experiment, sentence-level tagging has been performed based on the categories (e.g., reliability, efficiency, portability, usability, and maintainability) to a set of requirements. More than one label can be selected, as the objective is to create a multi- label corpus.

3.3.1 Corpus Annotation Through Crowdsourcing

Traditionally, domain-specific experts are hired with the transparency of the task, and they are remunerated for their services. However, expert-based acquisition is challenging. Hiring an expert is costly, and as this research is not externally funded, it limits this choice; it was also challenging to find suitable experts and raised many ethical issues. Moreover, it could introduce bias in the selection of an expert. Recently crowdsourcing [1]has emerged as a prevalent practice for this purpose. Crowdsourcing gathers workers in one place where they can annotate the corpus according to the given

[1] Jeff Howe of Wired magazine (Howe, 2006) first used the phrase, a combination of the terms "crowd" and "outsourcing", to refer to companies that performed a role by employees or hired employees to outsource a task to many people, particularly when enabled by online tools and venues.

research problem based on an online platform (Geiger et al., 2011). Crowdsourcing is typically carried out on dedicated platforms such as Amazon Mechanical Turk [1]or Figure Eight [2](Davis 2011), which enables interaction between contributors worldwide by providing a platform where many tasks can be distributed to human workers. The results are then aggregated together. An ethical assessment based on the PAPA (privacy, accuracy, property, and accessibility) approach (Mason, 1986) was conducted to perform this experiment. The use of the crowdsourcing approach involved minimum risk; therefore, an online platform Figure-eight tool was selected to perform annotation for the NFR corpus.

3.3.2 Data for a Multi-label NFR Corpus

The requirements extracted from SRS documents are exactly related to one class/label. These are in processed form, and the requirement analyst has invested time and effort in identifying and documenting them. At the same time, an automated requirement identification system aims to identify stakeholder requirements from raw data at the earliest stages of software development. The literature requirements in the raw form contain multiple functional and non-functional aspects. Labels are drawn based on the dependencies among NFRs mentioned in Table 3- 5. Based on this dependency relationship requirements are merged together to create a multi-label corpus.

The process starts by picking the data and manually mixing different requirements. We selected 200 samples from each NFR category and modifies them to represent multiple labels using the methods described and exemplified in Table 3- 5.

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

[2] https://appen.com/ The Figure-eight technology platform employs machine learning-assisted annotation methods to generate the high-quality training data that models require to perform in the real world (the platform is now called Appen and is under new administration.) Facebook, Twitter, Cisco Systems, GitHub, Mozilla, eBay, and Toyota are among the companies that use the platform. It facilitates the recruitment of annotators as well as the rapid expansion of annotation projects at a cheap cost and in a user-friendly manner).

Table 3- 5: Dependency Among NFRs

NFRs	Dependency Relation	NFRs
Reliability	Require	Maintainability
Reliability,	Conflict	Efficiency, Usability
Portability	Cost	Usability, Maintainability, Efficiency
Usability	Value	Efficiency

3.3.3 Annotation Framework Design and Settings

The platform has built-in templates related to various annotation tasks, including sentiment analysis, search relevance, image annotation, data categorisation, data collection, enrichment, etc. The template that best suits the problem at hand is data categorisation for text.

Recruitment of Annotators

The process is performed by annotators who decide based on both raw data and some information offered as instructions (Pustejovsky and Stubbs, 2013). The list given in Figure 3- 2 describes the criteria for the recruitment of the annotators.

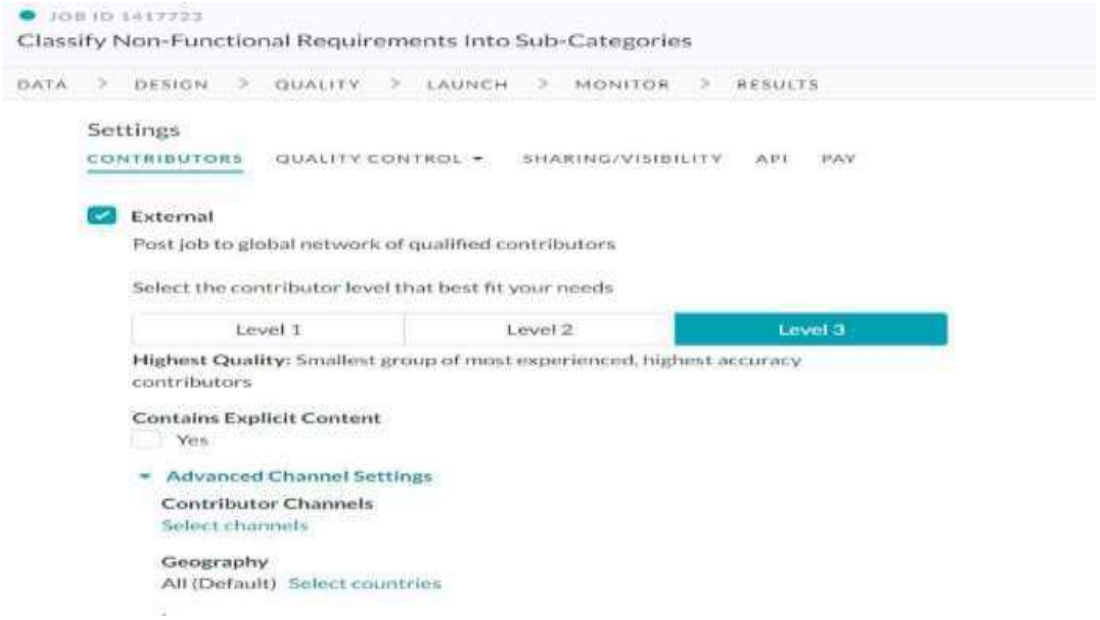

Figure 3- 2. Recruitment of Annotators

The job was open for worldwide crowd annotators on this platform. The members with expert level 3 were selected for this task.

Specifications and Guidelines

To begin the work of requirement annotation, specifications in the form of the participating categories' definitions are supplied (reliability, usability, maintainability, efficiency, and portability). A clear description of the job, unlabelled corpus file, and a set of rules were uploaded on the platform as shown in Figure 3- 3 , Figure 3- 4 and Figure 3- 5 respectively.

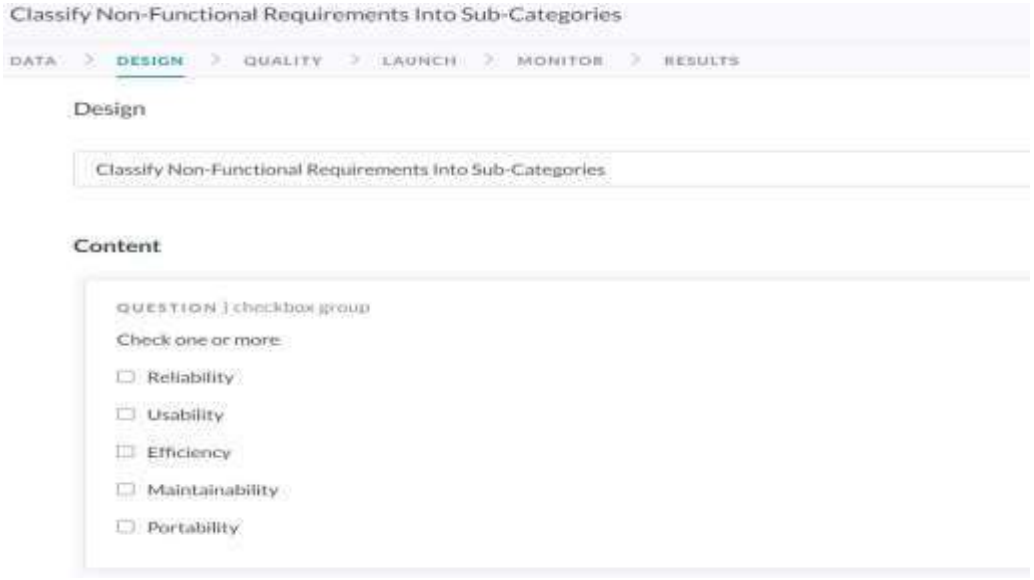

Figure 3- 3. Annotation Specification and Guidelines (a)

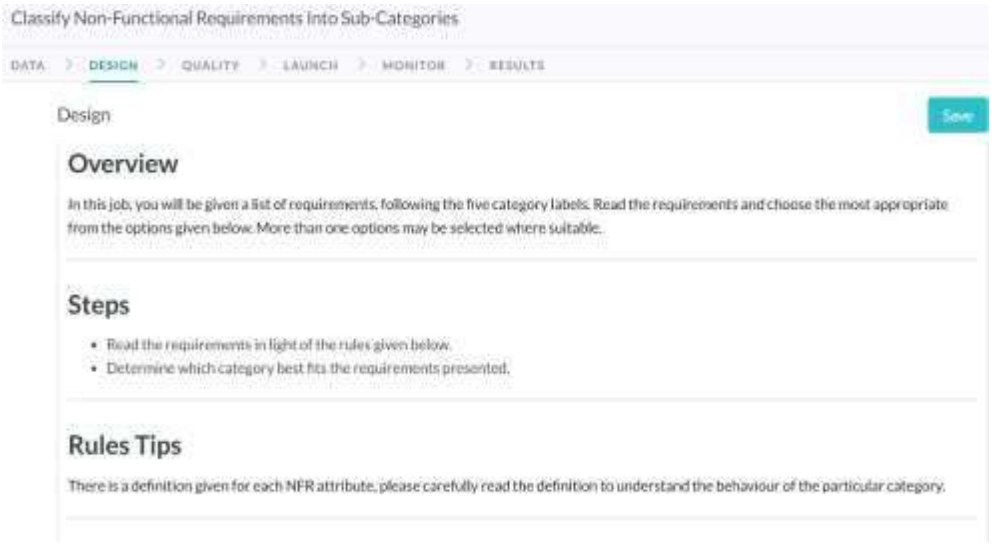

Figure 3- 4. Annotation Specifications and Guidelines (b)

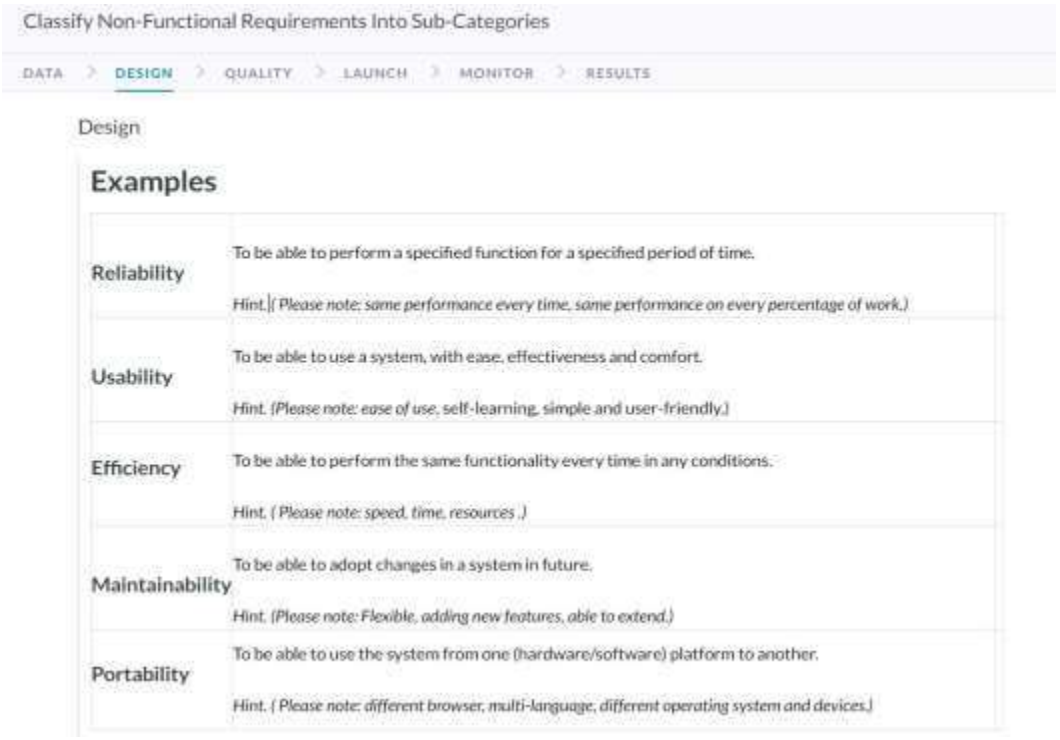

Figure 3- 5. Annotation Specifications and Guidelines (c)

3.4 Experiment (Corpus Annotation Procedure)

The experiment adopts a linguistic corpus construction scheme MAMA framework suggested by (Pustejovsky and Stubbs, 2013). The dataset consists of 1000 artificially generated sentences. The job was initiated with 100 samples of data which consist of 80% original requirements representing a single label and 20% artificially generated requirements. This initial set was used to train annotators. Once they became familiar with the job, the remaining artificially generated data were launched on the platform. The recruitment process starts with a test question containing a mixture of requirements belonging to all NFRs. Eleven human annotators showed interest in this test question to qualify for this job of assigning categories to requirements independently. Three participants who passed the test were selected for this job. The data was distributed between annotators, such that all three annotators annotated each requirement. Figure 3- 6 shows the steps involved in the procedure.

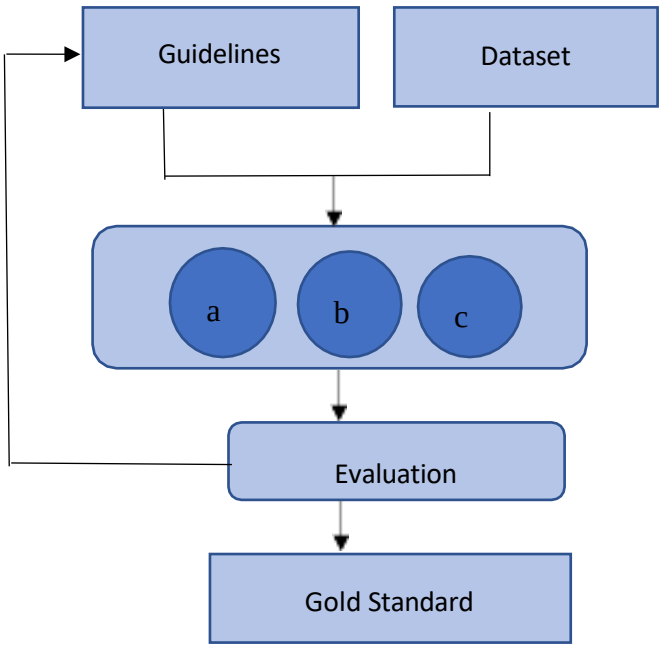

Figure 3- 6. Corpus Annotation Framework Design Procedure

3.4.1 Evaluation of Results

It is a process of analysing the results obtained from the annotators. The outcome is determined by applying the gold standard determination algorithm to the data given by three annotators to assess the accuracy of the annotation. The correct labelled set for requirements is decided by the inter-annotator agreement score calculated using Cohen's Kappa (Fleiss et al., 1969).

$$A_m = \frac{Po - Pe}{1 - Pe} \qquad \textit{Eq3- I}$$

A label is true if two out of three annotators agree on the same label. Randomly assigning items to a collection of categories, the observed agreement (P_o) is the percentage of items agreed on the label, for instance, by both of the annotators. The expected agreement (P_e) is the percentage of items on which agreement is expected by chance among the annotators. While A_m is measured to estimate the quality of the corpus, after predicted or chance agreement is taken out of the equation, it represents the percentage of agreement. Table 3- 6 shows the agreement between the pair of

annotators. All three annotators performed the task of annotation independent of each other, which resulted in 285 total annotations given by the annotators.

Table 3- 6: Results from Cohen's Kappa Agreement

Annotator pair	P_o	P_e	A_m Value
A-B	0.68	0.53	**0.31**
B-C	0.62	0.52	**0.20**
C-A	0.64	0.526	**0.24**

It is evident from Table 3- 6 that the highest agreement was found between annotators *A* and *B*, followed by *C* and *A*, whereas the lowest agreement was between *B* and *C*. However, the agreement value ranged between 0.2-0.31, which is considered a fair agreement according to Cohen's Kappa calculator. According to MAMA (Pustejovsky and Stubbs, 2013) framework, at this point, requirements can be given back to annotators with revised instructions to improve A_m value. The instruction and guidelines can be revised until the desired agreement is achieved. Later, the corpus can train a classifier for multi-label classification.

3.5 Conclusion and Future Recommendations

Section 3.4 suggested a framework to identify multi-label aspects of NFRs to create a multi-label gold standard corpus using a crowdsourcing web-based platform. The judgements were provided by three annotators using the figure-eight platform.

The annotation findings are based on inter-annotator agreements as determined by Cohen's Kappa. This methodology is intended to be iterative, and the initial findings indicate reasonable agreement between the annotators. However, the experiment demonstrates a variety of difficulties associated with utilising web-based platforms for domain-specific activities since they require advanced expertise from researchers and crowd annotators. Regrettably, the current study's examination is confined to a single repetition. One obvious reason for abandoning this experiment is its unavailability,

since the platform was taken over by new administration and confined to commercial usage only during the trial. Additionally, it was impossible to obtain a multi-label result in the initial iteration. The annotators worked on a single label (all annotators chose a single label for each need); this might be owing to the lack of direct contact between the researcher and the annotators. The annotators could only learn by referring to the job's instructions and specifications.

Many things can be improved in the context of a gold standard multi-label corpus, such as 1) how guidelines and rules are defined, 2) two-way communication for improved training, 3) a reliable platform for useful annotations, and 4) annotators with domain knowledge.

3.6 Summary

Corpus annotation is a vast topic of study. It is the first attempt to our knowledge to develop a representative corpus and a multilabel corpus in the domain of NFRs.

The contribution of this chapter is a representative domain corpus for NFRs termed the custom NFRs corpus, which is based on a sample chosen from software quality models. It has 1484 sentences divided into five NFR categories: efficiency, usability, reliability, portability, and maintainability. This corpus may train machine learning models to discover and categorize non-functional relationships. It will also be made public to stimulate more study.

Additionally, it offered an iterative strategy for obtaining a gold standard multi-label corpus for NFRs via a crowdsourcing platform on the web (figure-eight). The technique used three annotators, and the results were determined using Cohen's Kappa calculator. The initial data analysis reveals a high degree of agreement between the annotators. This study, however, is confined to a single repetition. The ultimate goal is to inspire future researchers to 1) train machine learning-based natural language processing systems and.2) assess the performance of natural language processing systems.

Chapter 4 Background and Experimental Settings for Deep Neural Networks

One of the drawbacks of traditional machine learning approaches for detecting NFRs is that features must be defined and retrieved manually or with the help of feature selection algorithms. Deep neural networks offer the advantage of not necessitating the creation of hand-crafted features. They could be able to learn semantic characteristics from word embeddings using context information during the training phase. These techniques are now the most successful solutions in the fields of image and audio classification, as well as natural language processing.

This chapter starts with a description of the deep neural network design appropriate for the classification of NFRs. Word embedding is discussed in Section 4.2, which leads to the supervised neural networks presented in Section 4.3. The classification process is described in section 4.4. The chapter concludes with an examination of how a model's architecture influences its representational capability, as well as hyperparameter and optimisation strategies.

4.1 Background of Deep Neural Network for Text Classification

Artificial intelligence (AI) encompasses machine learning as a subset that tries to create intelligent systems. DNNs can learn from data on their own, recognise patterns, and make decisions with little or no human intervention.

Supervised and unsupervised are two primary approaches in machine learning. The distinction is in the use of prior knowledge, which is represented by ground truth signals. Supervised learning aims to develop a mapping function that is the most accurate approximation of the link between the inputs and outputs in the data while employing ground truth values for samples. On the other hand, unsupervised learning is concerned with the underlying structure of the learning data rather than with the provision of output labels. Supervised learning is frequently used when it comes to

classification or regression tasks. However, unsupervised learning deals with tasks like clustering, representation learning (dimension reduction), density estimation, and other similar activities. In supervised learning applications, methods such as logistic regression, naive Bayes, support vector machines, artificial neural networks, and random forests are often employed.

Algorithms like k-means clustering, principal component analysis, autoencoders, and restricted Boltzmann machines are examples of unsupervised learning approaches. Specifically, in the context of this research, the focus is on the implementation of supervised learning in neural networks that are applied to textual data. DNNs have been effectively applied to natural language processing (NLP) tasks in recent years (Bengio et al., 2003; Collobert and Weston, 2008; Mikolov et al., 2013).

In 1943, neurologist Warren McCulloch and logician Walter Pitts created a relatively simplified first computational model of a neuron, in which they attempted to comprehend how the brain forms highly complex patterns by employing numerous linked fundamental cells (or neurons). The McCulloch-Pitts model serves as the foundation for neural network theory (McCulloch and Pitts, 1943; Piccinini, 2004).

Frank Rosenblatt made the next big development in the perceptron (Rosenblatt, 1958), which was announced in 1958. Neural networks use advanced mathematical models to handle data in a variety of ways to automatically learn and extract characteristics, resulting in improved accuracy and overall performance. These NNs are designed to approximate a specific function of interest, such as constructing an NFR classifier.

Such function maps an NFR's input x to a category ˆy. The model structure is primarily composed of three components (from input to output): (a) the word embedding layer, (b) the representation layer, and (c) the classification layer. In other words, a neural network is a collection of interconnected units with each link acting as a synapse and each unit having the form and function of a neuron. During the transmission of information between neurons, each synapse carries a weight that multiplies its inputs, and each neuron attached to it modifies the multiplied inputs to generate a correspondent output. In the following sections, each of these components is described in detail, particularly in terms of text classification, making it appropriate for the task at hand.

4.2 Distributed Word Representation in Neural Networks

Embedding is a fancy way of saying numerical values for words. Word embedding is a feature learning technique in which each vocabulary word or phrase is mapped to an N-dimensional vector of actual values. Word embeddings typically provide standard input representations of deep learning models, replacing traditional feature engineering. Multiclass text classification methods are commonly based on the bag-of-word representations technique discussed in Chapter 2. One of the fundamental limitations of such practices is that words are treated as independent features and do not retain any contextual information. However, an alternative method employing deep learning models for text classification can extract context-sensitive features from raw text. The development of distributed representations of words (Mikolov et al., 2013; Pennington et al., 2014), phrases (Socher et al., 2012) and sentences (Le and Mikolov, 2014; Kiros et al., 2015) has accompanied the success of deep learning-based natural language processing systems in recent years.

The distributed representations are real-valued vectors that flexibly represent the natural language's semantics. To address this issue utilising NFRs, this study provides a framework for the current situation that blends word2vec embeddings with deep learning to handle the problem.

Distributed representations, specifically “word embeddings”, follow a distributional hypothesis, which states that words that appear in similar situations have the same meaning. Consequently, each word is given a real-valued vector; furthermore, the vector space in which the words are stored is predetermined. Learning vector representations of words based on context have shown to be a success for neural networks (Bengio et al., 2003; Mikolov et al., 2010). Therefore, each word is mapped to a real-valued vector in a predefined vector space. On the other hand, each dimension in the bag-of-words representation of a sentence represents a word. When the document and/or vocabulary expands in size, these local representations become excessively sparse (many zeros). Additionally, the bag-of-words model does not account for word order, and the words are presented in a sequence.

4.2.1 Word2Vec Embedding Generation

Word to vector representation is a predictive model used to compute and generate a high-quality vector model created by Google in 2013 (Mikolov et al., 2013). The word2vec technique is used to represent the sentence in two ways, as shown in Figure 4- 1.

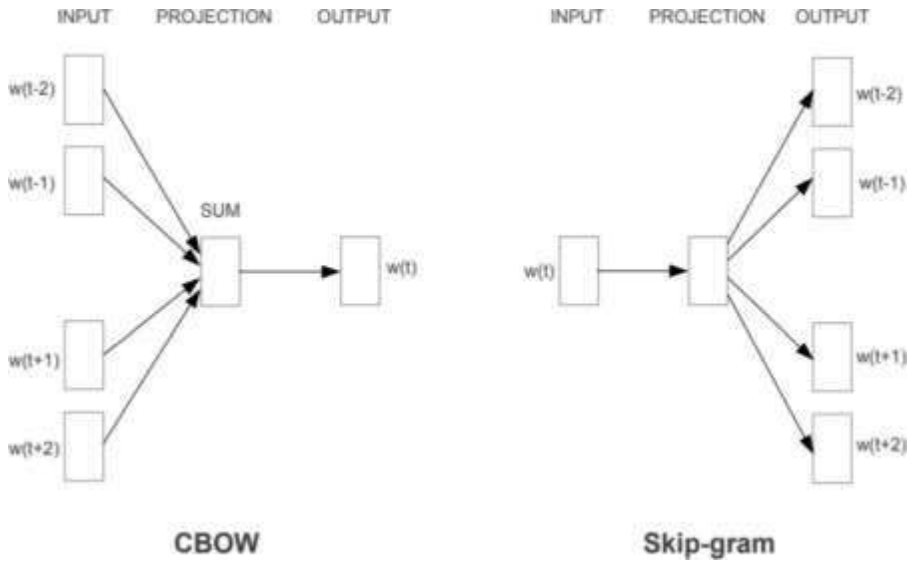

Figure 4- 1: Representation of Word2Vec Embeddings CBOW and Skip-gram Model (Mikolov et al. 2013)

Word embeddings starts with a vocabulary that stores all the corpus's unique words. The predicted outcome is calculated based on the context of the window size for the words (Levy and Goldberg, 2014). Figure 4- 1 shows word2vec's single-layer shallow neural network design, which is trained from scratch for the given dataset. Word2vec face two limitations: 1) sparse training data and 2) a large number of trainable parameters. Another option is to employ word embeddings that have been pre-trained on big datasets to capture semantic and syntactic information, which makes these models capable of boosting the performance of a classifier. Word embeddings are currently available for various techniques, including continuous skip-gram, continuous bag-of-words (CBOW), GloVe, and fast text (Bojanowski et al., 2017).

Continuous bag-of-words learns embeddings by predicting the current word based on its context. A basic CBOW model attempts to find a word based on previous words to find associations and similarities between terms in the text corpus. The input and output

layers share the same weight matrix (Mikolov et al., 2013). Unlike the traditional bag-of-words paradigm, CBOW uses a constantly distributed representation of the context. CBOW is more efficient with frequent words (Naili et al., 2017). The continuous skip-gram model (Skipgram) predicts the context words using the centre word. It tries to maximise a word's classification based on another term in the same sentence (Mikolov et al., 2013). These two architectures lower the complexity to

$$ND + D\log(V) and\ C(D + D\log(V)) \qquad \textit{Eq4- I}$$

For each training word per epoch, where N is the number of words in the context, V is the size of the vocabulary, C is the maximum distance of the words. More precisely, it uses each current word as an input to a log-linear classifier with a continuous projection layer and predicts words within a specific range and after the present word. The target word's input is fed; in this case, the hidden layer remains the same, and the neural network output layer is repeated numerous times to satisfy the desired amount of background terms. Skip gram is more efficient with infrequently used words (Naili et al., 2017). In semantic analogy tasks, skip-gram substantially surpasses representations generated by CBOW, whereas, in syntactic analogy tasks, skip-gram and CBOW perform equally.

4.2.2 Transfer Learning

It is expensive to train word embedding with large amounts of data. A different technique ensures that high-performance learners are trained with data from different domains that are more readily available. Transfer learning, or domain adaptation, are terms used to describe this method (Perera and Patel, 2019). The network is trained on out-of-distribution data first, then fine-tuned on domain-specific training data (Wolfe and Lundgaard, 2020). The final activation score is thresholded to determine novelty. These earnings could be either weights or embeddings. In the case of this research, learnings are the embeddings, and this concept is known as pre-trained word embeddings.

4.3 Feedforward Neural Network

Feedforward neural networks are those that do not form a cycle of connections. A single-layer perceptron (SLP) shown in Figure 4- 2(a) is the most basic type of feedforward neural network, as it has no hidden layer, and the inputs are connected directly to the outputs.

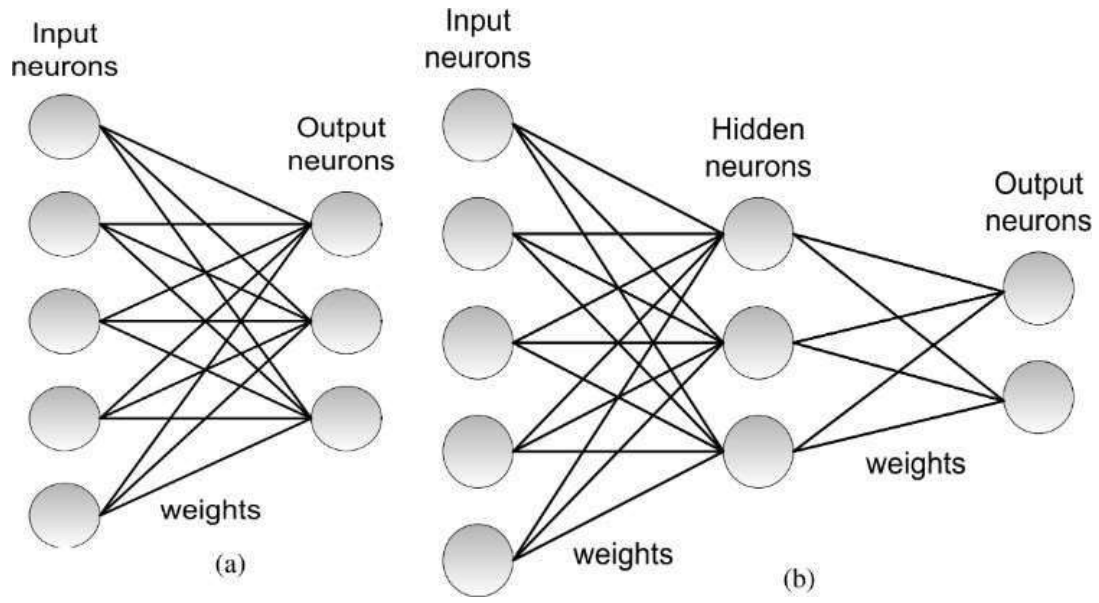

Figure 4- 2:Single Label Perceptron vs Multilabel Perceptron (Camuñas-Mesa *et al.*, 2019)

These SLPs can be stacked to form a multi-layer perceptron (MLP). An MLP's structure can be thought of as a series of layers, as shown in Figure 4- 2(b). It consists of three layers: an input layer that processes data; hidden layers that do mathematical computations on the input data to learn relationships; and an output layer that predicts output based on the learned relationships. Stochastic gradient descent with back-propagation can be used to train MLPs (Rumelhart et al. 1986).

4.3.1 Artificial Neural Networks (ANNs)

Artificial neuron networks (ANNs) are also known as feed-forward neurons. ANN tries to replicate the human brain's ability to self-learn in terms of adaptivity, defect tolerance, nonlinearity, and mapping improvement (Wang et al., 2018). It processes inputs only in the forward direction, through various input nodes, until it makes it to

the output node. ANNs are made up of neurons with weights between them, and throughout the learning process, they modify the weights depending on an error signal (or feedback) to obtain the desired output for a particular input. In the simplistic form of ANN, hidden nodes are optional, making their functioning more useful.

As shown in Figure 4- 3

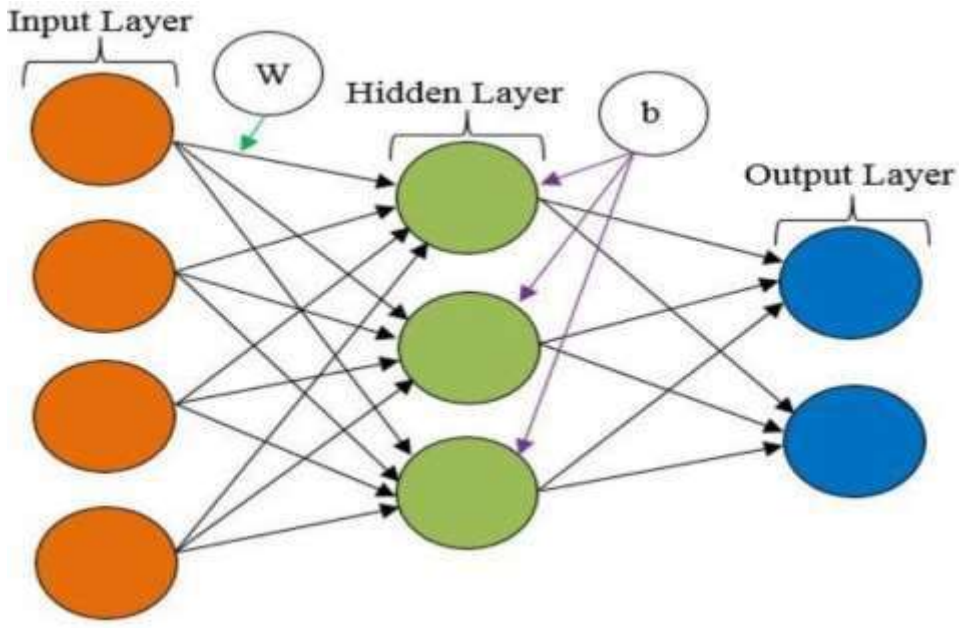

Figure 4- 3:A Simple Architecture of Artificial Neural Network (Rahman et al, 2019)

ANNs were developed and used for image recognition and, more recently, for natural language processing (Abiodun et al., 2018).

4.3.2 Convolution Neural Network (CNN)

Convolutional neural networks (CNNs) were first built based on Fukashima's neurocognition (Fukushima, 1980; Fukushima and Miyake, 1982). The name CNN is derived from the convolution operation in mathematics and signal processing. However, because of the limits of computer hardware for network training, it was not widely employed at first. In the 1990s, a gradient-based learning technique was used to solve the declining gradient problem and create highly optimised weights (LeCun et al.,1989). Feature extractors and a classifier are the two fundamental components of a CNN's overall architecture, as shown in Figure 4- 4.

A series of convolution and pooling pairs constitute the feature extraction layers, which are followed by a few fully connected layers that form the classification (Baker et al., 2017; Hadji and Wildes, 2018). It is a sort of feed-forward neural network that uses "convolutional filters" to improve performance. Each word is turned into a weighted vector with user-defined dimensions. A feature map is created by grouping the output nodes from the convolution and max-pooling layers (Krizhevsky, 2014). Features transmitted from lower-level layers are used to create higher-level features. In the convolutional and max-pooling techniques, the dimensions of features are lowered as they propagate to the highest layer, depending on the size of the kernel. The CNN's last layer is fed into a classification layer, which is a fully connected network. Similar to feed-forward neural networks, convolutional neural networks can also be trained using standard backpropagation (LeCun et al., 1989).

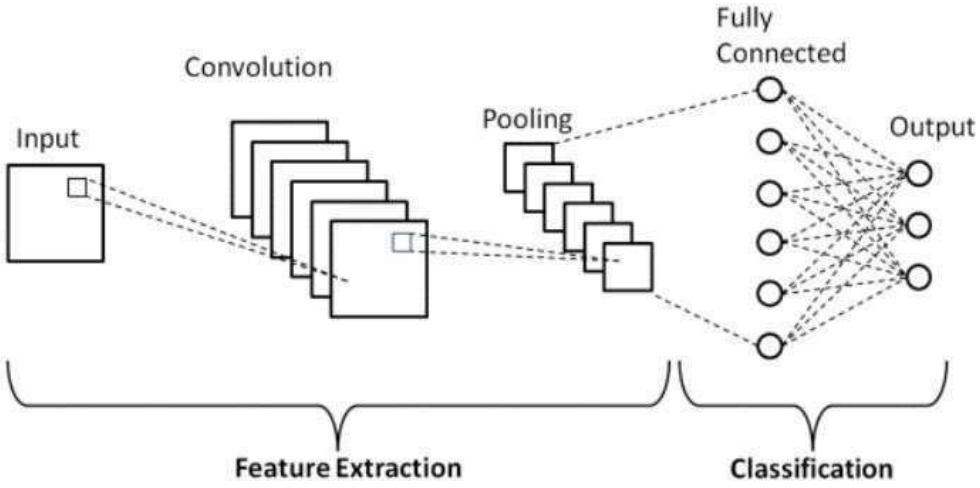

Figure 4- 4: A layered Architecture of Convolution Neural Network (Phung and Rhee, 2019)

Following that, researchers improved CNNs even further and reported cutting-edge outcomes in various recognition tasks. CNN was first exploited to create a semantic representation of the textual domain. Collobert et al. (2011) were pioneers of the use of CNNs for NLP tasks such as POS tags, chunks, and named-entity tags. Later CNNs were used for sentiment/opinion mining (Kalchbrenner et al., 2014; Kim, 2014) and in relation extraction (Zeng et al., 2014; dos Santos et al., 2015) with fairly balanced class distributions. The successful implementation of CNNs in the NLP domain makes it useful in mining semantic clues in contextual windows. CNN requires a broad set of

labelled data, making it challenging for the researcher to adopt data sparsity. CNN has a disadvantage in that it is unable to model long-distance contextual information while maintaining sequential order in its representations (Hu et al., 2015; Kalchbrenner et al., 2014).

4.4 Recurrent Neural Network (RNN)

RNNs were defined by Rumelhart et al. (1988; Elman, 1990) as "supervised neural networks". RNNs use internal memory to recollect their prior input every time a new input is brought into the network. As a result, they simulate sequential information through a series of feedback loops that recur over time. It does the same task for each sequence element, with the outcome decided by previous calculations, which generates a fixed-size vector to represent a series.

RNNs differ from feed-forward neural networks in that they can handle variable-length sequences in both input and output. RNNs can swap features collected over many time steps and record relationships in the sequential input to account for the direct flow of information. They can recall and reuse prior knowledge computations by applying them to the next element in the input sequences. RNNs are capable of capturing the fundamental sequential nature of language units such as letters, phrases, and even sentences, among other things. The semantical meaning of a sentence is inferred from the words that came before it in the sentence. They are capable of modelling text of varying lengths, including highly long phrases, paragraphs, and even whole manuscripts (Tang et al., 2015). In the NLP area, it has been effectively used for tasks such as point of sale tagging (Zhang et al., 2016) and, more recently, text classification (Chen et al., 2017) and multimodal sentiment analysis (Zhang et al., 2017), among others (Poria et al., 2017). As a result of these capabilities, the RNN has become a well-known neural architecture for solving sequential tasks, such as language modelling (Mikolov et al., 2010; Peters et al., 2018), named entity recognition (Ma and Hovy, 2016), relation extraction (Vu et al., 2016; Gupta et al., 2019), textual similarity (Gupta and Schütze, 2018), and sentiment analysis (Tang et (Zhang and Lapata, 2014).

Gradient descent with back-propagation through time (BPTT) is the usual approach for training an RNN (Rumelhart and McClelland, 1986). However, the vanishing gradient problem affects RNN networks. Its descendants, such as long short-term memory (LSTM) and gated recurrent units (GRUs), overcame this restriction by efficiently back-propagating error gradients (Hochreiter and Schmidhuber, 1997).

4.4.1 Long-Short Term Memory (LSTM)

Hochreiter and Schmidhuber (1997) suggested that long short-term memory (LSTM) is a form of RNN (Elman, 1993). An LSTM unit has a "memory" cell that can keep its state value for an extended period. It uses a gating mechanism with three non-linear gates: an input, an output, and a forget gate. Since most NLP tasks depend on words or other elements, such as phonemes or sentences, it is helpful to remember the previous details when processing new ones (Mikolov et al., 2015).

LSTM can use long memory as the input to the hidden layer of the activation function. Input data is pre-processed to reshape data for the embedding matrix. The LSTM, which contains cells, is the next layer, followed by a completely connected layer. Unlike the vanilla RNN, LSTM enables the error to backpropagate through an infinite number of time phases. Figure 4- 6 illustrates how LSTM works with a gating mechanism. LSTM has three gates. The gating mechanism is what allows LSTMs to explicitly model long-short term dependencies. As shown in Figure 4- 5, the network learns how its memory should behave by learning the parameters for its gates. Each vanilla LSTM module comprises a central value that acts as memory c t at time t. Input gate I t, output gate o t, and forget gate f t are all available. Combinations of c t, input x t, and output h t-1 result in these gates. The gated input and gated c t-1 produce a new value, c t. The output gate o t controls the module's c t output.

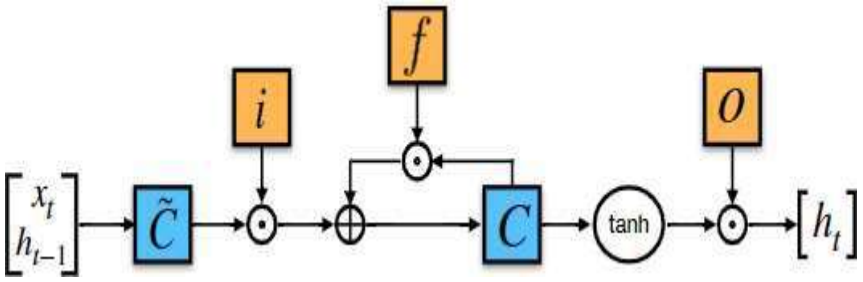

Figure 4- 5: An Illustration of Long-Short term Memory (Chung et al. 2016)

Many changes have been made to the LSTM unit since its conception to increase its performance. SGD with BPTT can be used to train weight matrices and bias vectors. LSTM has been adopted by several state-of-the-art NLP systems, such as dialogue systems (Sutskever et al., 2014), tweet encoding (Wang et al., 2015), and language modelling (Shen et al., 2018). Due to the four-times increase in the number of parameters compared to a simple RNN, LSTMs have higher memory needs. LSTMs use many memory cells. Therefore, they have a far greater computational complexity.

4.4.2 Gated Recurrent Unit (GRU)

GRU (Chung et al., 2014) is a slightly simpler variant of the LSTM. The cell state and concealed states are combined into a single memory content. Other than that, the GRU has no control over the memory content's accessibility to other network units. A GRU has two gates: a reset gate that determines how to integrate the incoming input with the old memory and an update gate that specifies how much of the last memory should be kept. Figure 4- 6 shows the gating mechanism. A gating mechanism, like an LSTM, learns long-term dependencies, but it is different from LSTM due to an output gate with controlled exposure.

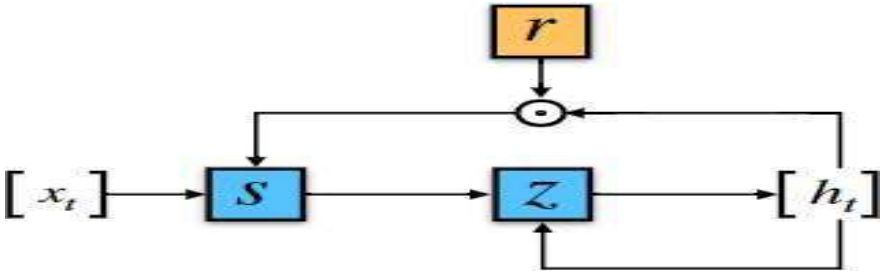

Figure 4- 6: An Illustration for GRU (Figure Source: Chung et al. 2016)

GRU has shown competitive performance but suffers from vanishing gradient problems via a gating mechanism.

4.5 Classification Layer

The feature maps in the final layer are represented as vectors, with scalar values supplied to the fully linked layers as they are combined. The output is generated with the association of an activation function. Figure 4- 7 describes the classification process in a DNN, where x_1, x_2, and x_m represent the input values, w_1, w_2, and w_m are the weights calculated in the internal layers and $\sum$ represents the sum of those values. A classification layer receives a sum of weights from the internal layers and predicts a value of Y with the help of an activation function.

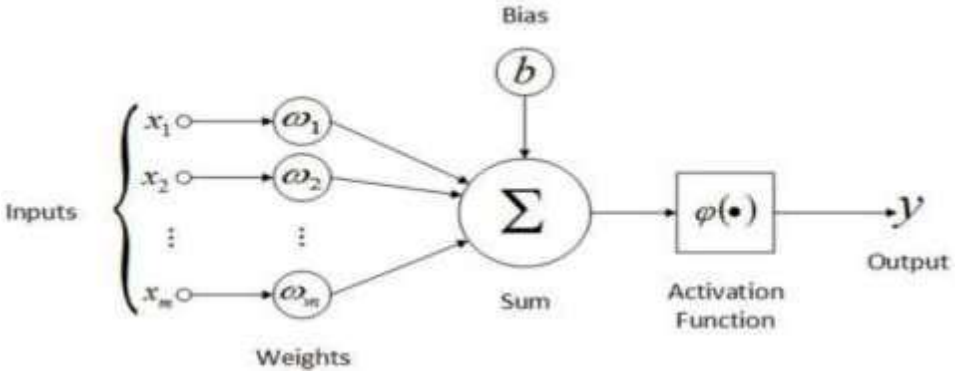

Figure 4- 7: An Internal Function Involved in Classification (Tzanis and Alimissis, 2021)

4.5.1 Activation function

The DNN's output layer might be considered the final layer. Linear, sigmoid, Tanh, and SoftMax are employed as output layers in the DNN for classification. It determines how close the parameters are to the training and validation data's ground truth labels.

4.1.5.1 Softmax Classifier

The softmax function is the gradient log normaliser of the categorical probability distribution (Joo et al., 2019). It begins the same way as the standard layer, which forms the weighted inputs for the DNN. The framework is made to return N values, one for each category in the classification.

task. The softmax function is used to normalise the outputs by requiring the input values of the last layer to be positive and their total to be 1. Each number in the softmax function's output is interpreted as the likelihood of belonging to each class. Linear and sigmoid activation functions are ineffective for multi-class classification problems. Softmax, on the other hand, can be utilised to solve probabilistic multiclass classification issues (Zhu et al., 2020).

4.1.5.2 Rectified Linear Unit (ReLU)

The rectified linear unit (ReLU) was first proposed by Nair and Hinton 2010 as an activation function (Nair and Hinton, 2010). In deep learning, the ReLU outperforms the Sigmoid and Tanh activation functions in terms of performance and generalisation. The ReLU depicts a nearly linear function, preserving the features of linear models. When the input value is positive, the derivative of the input is always 1. As a result, it addresses the vanishing gradient problem (Zeiler et al., 2013).

4.1.5.3 Hyperbolic Tangent Function (Tanh)

The Tanh function's key benefit is that it generates zero-centred output that has a range of -1 to 1. It aids the back-propagation process in comparison to the sigmoid function (Nwankpa *et al.*, 2018). The tanh function, on the other hand, could not solve the sigmoid functions' vanishing gradient problem (Karlik and Olgac, 2011). The function has been utilised primarily on neural networks with recursion and provides higher training performance for multi-layer neural networks in the field of Speech and natural language processing.

4.5.2 Loss Function

In a neural network, a loss is nothing more than the predicted error of the network, and the mechanism used to compute the loss is referred to as the Loss Function. Put another way, the loss is utilised in the calculation of the gradients. In addition, gradients are employed to update the weights of the neural network. When a model is parametrized and then translated into the real domain, the loss function (also known as a cost function) is computed as a function. It assesses how inaccurate the current valuation of the parameters is at the time of the measurement. For faulty forecasts, the loss function

should return high values, whereas, for successful predictions, the loss function should return low values. Optimisation processes are approaches for determining the input that will result in the lowest amount of loss.

4.2.5.1 Cross-Entropy

Cross entropy loss, also called logistic regression loss, is an alternative measurement of a probability distribution. In order to assess the performance of a neural network model, the cross-entropy function allows the network to evaluate such minor errors (Mannor et al., 2005). The averaged cross-entropy is represented as a loss function that can interpret the softmax classifier. The cross-entropy between actual distribution p and a predicted distribution q is represented as:

$$H(p, q) = -\sum_x p(x) \log q(x) \qquad \textit{Eq4- II}$$

Hence, the task of the softmax classifier is to minimise the cross-entropy between the actual distribution and the predicted distribution.

In summary, the softmax classifier can be interpreted in a probability view. Given a sample (x_i, y_i) and parameters W, it can compute the normalised probability:

$$P(y_i|x_i ; W) = e^{fyi}|\sum_j e^{fj} \qquad \textit{Eq4- III}$$

Where fy_i is the score predicted by the model with weights W. Therefore, the normalised probabilities are computed by exponentiating the values and dividing by the sum of all values. In some circumstances, the error function e reflects some assumptions about the data distribution as well as an adequate notion of quality (Golik et al., 2013). Minimising cross-entropy refers to increasing the likelihood of the correct label or decreasing the dissimilarity between the network distribution estimation and the suitable distribution to approach c.

4.5.3 Back-propagation

Backpropagation is a popular approach for training feedforward neural networks. It computes the gradient of the loss function for the network weights for a single input-

output pair at the time of training a neural network. Usually, gradient descent, or stochastic gradient descent, trains multilayer networks and updates weights to minimise the loss. After introducing the loss function, the neural network can learn with gradient descent. The loss function represents the error function, and the weights are the function's variables. The gradients of the error function with respect to the weights are called error gradients. It passes out the output from the activation function to the next hidden layer. When the result differs significantly from the actual value, the process of determining the error value and updating the weights to adjust those biases based on that value is known as backpropagation.

4.6 Optimisation in Deep Learning Architectures

An optimiser guides the weights associated with layers in order to reduce loss while forecasting the labels for training and validation data. The optimiser is used to adjust weights based on the loss experienced during the network's training stage. Training accuracy is typically higher than validation accuracy. The overfitting phenomenon occurs when the discrepancy between the two accuracies is too significant (Cogswell et al., 2016). Put another way, the DNN utilises its weights to memorise the training set instead of looking for distinguishing traits that might help it learn. The DNN’s sophisticated design is another contributing factor to overfitting. Regularisation approaches can be used to make the DNN as complex as desired while limiting overfitting (Kukaka et al., 2017; Minar and Naher, 2018). Because of this, it supports the use of simple models to generalise previously unobserved data points (Nusrat and Jang, 2018).

4.6.1 Scholastic Gradient

The scholastic gradient is an expansion of the gradient descent. A variation of gradient descent known as stochastic gradient descent (SGD) has been widely employed in CNNs. During each epoch of the gradient descent, the algorithm calculates the gradient and updates the network weights before evaluating the output of each single data point in the training set. Instead of single data points, mini batches can be used to reduce the

gradient variability. Even if the training set contains only a few hundred photos, calculating the gradient descent will take considerable time. A tiny random sample from a training set is used to estimate the gradient decline in the SGD.

4.6.2 ADAM Optimiser

To calculate each parameter's adaptive learning rates, the ADAM (Adaptive Moment Estimation)-Optimiser is one of the most prominent adaptive step size approaches (Kingma and Ba, 2017). ADAM is a more complex version of stochastic gradient that preserves an average of past gradients as well as an exponentially declining average of past squared gradients, similar to the momentum approach (Perin and Picek, 2021).

4.6.3 Adaptive Methods

There are two primary adaptive method approaches. Structural stabilisation changes the adaptive parameters in hidden layers, such as neuron numbers (Liu and Liao, 2014). Structural stabilisation can be approached from two directions. Structural stabilisation starts from a small network and increases layer numbers or utilises neuron numbers in the training process to arrive at a significant neural network architecture. The other is to start from an extensive network and prune out layers or neurons in the training process to achieve the optimal neural network architecture (Gupta and Raza, 2018).

The architecture of a deep learning network is determined by selecting hyperparameters for each layer (Caselles-Dupré et al., 2018). The majority of deep learning algorithms explicitly provide hyperparameters that regulate various aspects such as memory and execution cost. Levy et al. (2015) illustrated that careful optimisation of hyperparameters is often more important than the chosen embedding algorithm itself.

The primary goal of hyperparameter selection is to fine-tune a model's capacity to match the difficulty of the target task. Since it is impossible to learn from the training set, a setting is often modelled as a hyperparameter (Aghaebrahimian and Cieliebak, 2019). Usually, the hyperparameters are determined by human intuition, experience or trial and error (Andonie, 2019). There are two primary approaches to selecting and

optimising hyperparameters: manual and automatic selection (Chan et al., 2013). The decision to use one over the other usually reflects a trade-off between a deep understanding of the model needed to manually pick hyperparameters versus the high cost of computing required by automated selection algorithms (Luo, 2016). Some typical hyperparameters must always be considered (Hutter et al., 2019). These include **1) Learning rate:** the learning progress of a model in a way that can be used to optimise its capacity, 2) **Number of hidden units:** The number of hidden units is vital to regulating model representative capacity, 3) **Division of the dataset:** For a complete epoch, the data set is separated into training and validation sets. It is helpful to compute and compare the training and validation accuracy after each epoch is completed. An unpredictable number of settings can play the role of hyperparameters for specific models (Balaprakash et al., 2018). The number and divergence of hyperparameters in machine learning algorithms are precise to each model.

4.7 Regularisation in Deep Learning

In general, in the context of refining model design, certain solutions are given to reduce overfitting. A pooling layer is regularly added between successive convolutional layers to minimise the spatial size of the representation and the number of parameters. To provide more information, a favourable explicit regularisation type is adopted. In all training data, a dropout layer is employed to eliminate neuron interactions and gain more robust features.

4.7.1 Drop out

Drop out is a helpful tool to enhance generalisability (Srivastava et al.,2014). The first technique, known as naïve or straightforward drop-out, was proposed to remove the connections between deep layers. The key idea is to drop neural network units randomly during training to prevent units from being too co-adapted (Hernández-García and König, 2018). Applying dropout means randomly dropping a unit out or temporarily removing it from the network. This zeros the activation of randomly selected nodes with a certain probability during the training process. It also helps avoid

overfitting in DNNs, another differential feature of the neural network. It disables some neurons at each training iteration to prevent them from being too dependent on each other.

4.7.2 Early Stopping

The model's accuracy in fitting unseen data is evaluated after each epoch when using iterative gradient descent to train a neural network. Early stopping is a technique that is introduced if the model's performance on the validation data is not improving (Prechelt, 1998; Song et al., 2020). Early stopping stops the model's training process before reaching the lowest training error, thereby ensuring that the variance of the estimator is not too high.

4.7.3 Weight decay

The capacity of a neural network to generalise depends on the balance between training examples and the system's complexity. A way to restrict a network and thereby reduce its complexity is to limit weight growth by some form of weight loss (Krogh and Hertz, 1992). This is commonly done by L2 regularisation, which adds a penalty for high weights to the network's cost function. Weight decay, on the other hand, is an extra term in the weight update rule that causes weights to fall exponentially to zero without any other changes. The cost function can be controlled to effectively restrict the number of free parameters in the model to prevent overfitting. A practical course of action to apply a regularisation term to the energy function is to design a Gaussian zero average overweight, similar to changing the cost function. The regularisation term L2-norm is used to penalise huge weight values; the weight-decay coefficient can be used to directly adjust the regularisation effect (Nakamura and Hong, 2019). The weight-decay coefficient can be adjusted by hand or learned through Bayesian optimisation; layer-wise weight-decay has just recently been addressed.

4.7.4 Data Augmentation

Data augmentation is a method of artificially producing data from the existing training data by making small adjustments to the dataset (Wong et al., 2016). According to Taylor and Nitschke (2018), it is common knowledge that misalignment might lead to

a shortage of data on the adversary's side. Second, the DA is controlled, which means that the data deformations are chosen and thus fully defined. As a result, the classification problem's complexity can be determined to its entire extent.

4.8 Summary

The literature on deep neural networks and text classification was reviewed in Section 4.1. Section 4.2 presented techniques related to word representation and transfer learning. Section 4.3 and 4.4 defines the neural networks, multi-layer perceptron model as well as feed forward and recurrent neural networks elated to this study. Moreover, classification of DNN, some output functions, back propagation process and gradient of different levels were highlighted in Section 4.5.

In deep learning architectures optimisation is an issue, various approaches to addressing this are described in Section 4.6. Lastly some regularisation strategies were discussed towards the end of this chapter.

Chapter 5 Framework Design for an NFR Classification System

The goal of this chapter is to develop a suitable architecture based on deep learning techniques for the classification of NFRs. It provides step-by-step details for the design and implementation of these neural networks, while highlighting the possible benefit of using a framework that eliminates the need for human based feature engineering.

The chapter begins with a problem statement and a possible solution. Section 5.2 provides the detailed design of the architecture. The model's implementation is given in Section 5.3. This chapter concludes with the preliminary results obtained from the experiment.

5.1 . Problem Formulation

The aim of this study is to design an optimal framework for the classification of NFRs that includes a method of data augmentation, word2vec embeddings, and a deep learning model based on a representative NFR corpus. The problem of NFR representation has been formulated as a multi-class classification.

For the given repository of requirements, Let $X = \{x^1, x^2, x^3, \ldots x^N\}$ is a set of N requirements, such that each requirement is defined as; $x \in R^D$ where D is the maximum length of the dataset. In the case of supervised learning, the problem is to estimate a function f_e that transforms the requirement into M-dimensional latent space as:

$$z = f_e(x, \theta_e, \beta_e) \qquad \textit{Eq5- I}$$

where $z \in R^M$ is the extracted representation θ_e and β_e are the weights and biases of the estimation function f_e, respectively. These representations are then associated with different categories using a softmax classification defined as:

$$\tilde{t} = \sigma(z) \qquad \textit{Eq5- II}$$

Function f_e is learned by minimising the error between actual label t and predicted class label $\tilde{t}$ using:

$$Jc = -\sum t \log\tilde{t} + \qquad \textit{Eq5- III}$$

For an automatic NFR classification, this study proposes to use four-phase-deep neural network frameworks starting with 1) pre-processing, 2) embedding generation, 3) feature learning, and 4) classification, as shown in Figure 5- 1.

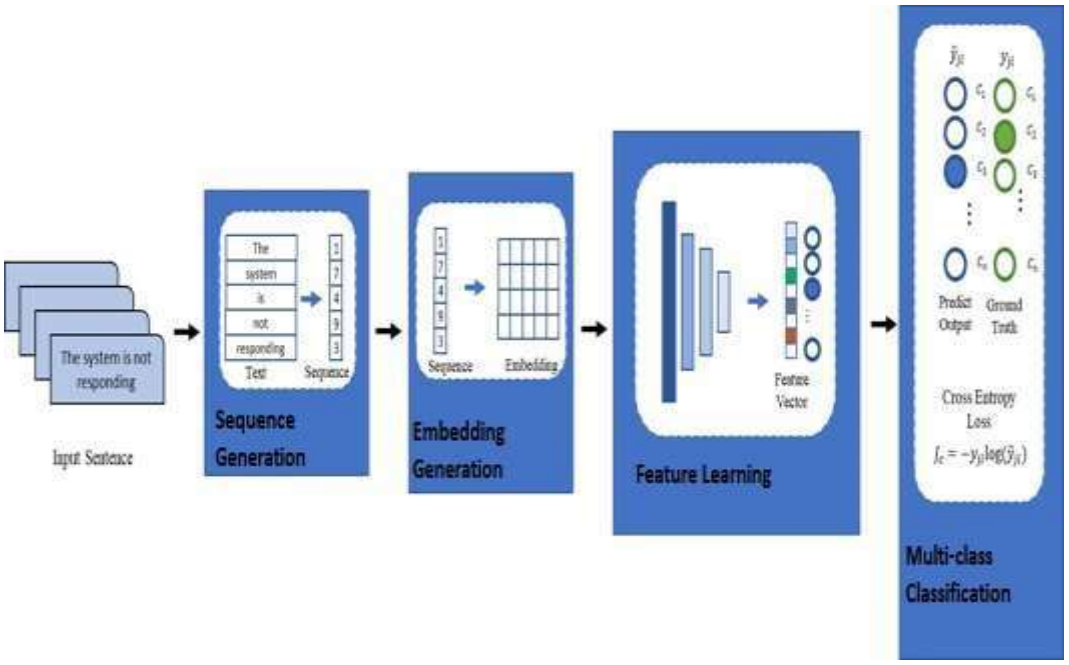

Figure 5- 1: Framework for NFR Classification, including Phases of Pre-processing, Embedding Generation, Feature Learning, and Classification

5.2 Training Configuration for a Baseline Classification Model

This section formalises the proposed approach for NFR classification, one of the contributions of this thesis mentioned in the introduction of this chapter. It provides the step-by-step procedure to train four deep neural networks: CNN, ANN, GRU, and LSTM, to design a classifier for this experiment.

5.2.1 Corpus for Training

This experiment uses a benchmark text corpus for NFRs that has recently been developed[1]. The corpus consists of 1484 sentences taken from the SRS documents.

Table 5- 1: Class-wise Distribution for Custom NFRs Corpus

Category	Total Samples
Efficiency	480
Maintainability	240
Portability	156
Reliability	191
Usability	417
Total Documents	1484

The corpus consists of five classes: efficiency, maintainability, portability, reliability, and usability. Each category contains a different number of sentences. The class-wise distribution of samples has been mentioned in Table 5- 1. The data ratio between the training and validation sets remains 80:20 for all four DNNs.

5.1.2.1 Pre-processing

Pre-processing is required for the initial phase of training, which is to refine the data. In section 2.4 of Chapter 2, various pre-processing approaches were addressed. This experiment does not necessitate any of the syntactic or semantic labelling outlined earlier. However, some essential data cleansing is still required. Therefore, a word embedding strategy is used for semantic learning in this study.

1 NFRs Corpus created because of research question 1, details can be found in Chapter 3.

5.1.2.2 Sequence Generation

As neural networks could take only numbers as input, training deep learning models on the given dataset requires the sentences to transform into sequences of numeric values. Numeric values are taken from the dictionary indices generated for all the corpus words. For this purpose, the sentences are then tokenised into words considering white spaces as delimiters and replaced with the characters with small letters for each word. In addition, it filters out the list of punctuation marks, including ``! #$%& () *+, -. /: ;<=>? @ [] \^ _` {|} ~ \t\n" from the sentences. The dictionary of the corpus is generated so that a unique index number represents each word. These indices are used to convert sentences into sequences. The total number of unique words in the dictionary plus a stop word (usually placed at index 0) comprises the vocabulary of the corpus. An example of the sentence conversion into sequences is shown in Table 5- 2.

Table 5- 2: Conversions of NFRs Sentence into Sequences

Sentences	Post Symbol Removal	Sequence
The system shall refresh the display every 60 seconds.	the system shall refresh the display every 60 seconds	[6, 1, 21, 247, 6, 75, 20, 745, 46]
The system will support 60HZ display	the system will support 60HZ display	[1, 7, 91, 746, 75]
The system will refresh display every time automatically	the system will refresh the display every time automatically	[1, 7, 247, 75, 20, 10, 113]
Users can set auto timing in the system	user can set auto timing in the system	[2, 4, 576, 222, 747, 12, 6, 1]

5.1.2.3 Sequence Padding

DNNs are best suited for a limited set of inputs. However, due to the variety of terms used in different sentences, the length of sequences in this scenario varies dramatically. As a result, the maximum length of a phrase in the corpus is determined as 46, and pad zeros are assigned to sequences that are shorter than 46. Table 5- 3 shows an example of the post-padding method that is adopted in this study.

Table 5- 3: Step-by-Step Conversion of Sentences into Sequence and Padding

Sentences	Sequence	Post-sequence Zero Padding (Maximum Length = 46)
The system shall refresh the display every 60 seconds.	[6, 1, 21, 247, 6, 75, 20, 745, 46]	[6 1 21 247 6 75 20 74 5 46 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
The system will support 60HZ display	[1, 7, 91, 746, 75]	[1 7 91 746 75 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
The system will refresh display every time automatically	[1, 7, 247, 75, 20, 10, 113]	[1 7 247 75 20 10 113 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Users can set auto timing in the system	[2, 4, 576, 222, 747, 12, 6, 1]	[2 4 576 222 747 12 6 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

5.1.2.4 Embedding Generation

After sequence generation the next step is to acquire embeddings of these sequences. Word embeddings techniques were discussed in the previous chapter in section 4.3. This experiment uses word2vec skip-gram model where these word embeddings are trained from scratch on the custom NFR corpus.

5.3 Training Configuration for DNNs

Deep neural networks used for NLP include the embedding layer as the first layer of the system. Second layer receive embeddings of the sentences to learn discriminant features of the text by transforming them from embedding space to feature space. For multiclass classification, a feature for each text is discovered to maximise the probability of belongingness to the actual class. Principally, gradient descent with backpropagation executes this successfully. Figure 5- 2 is created to illustrates the transformation of sentences to word embeddings and their representative features. It is important to note that features in each sentence are mapped to multiple semantic concepts expressed by distributed words. As a result, one or more semantic notions are included in a single class.

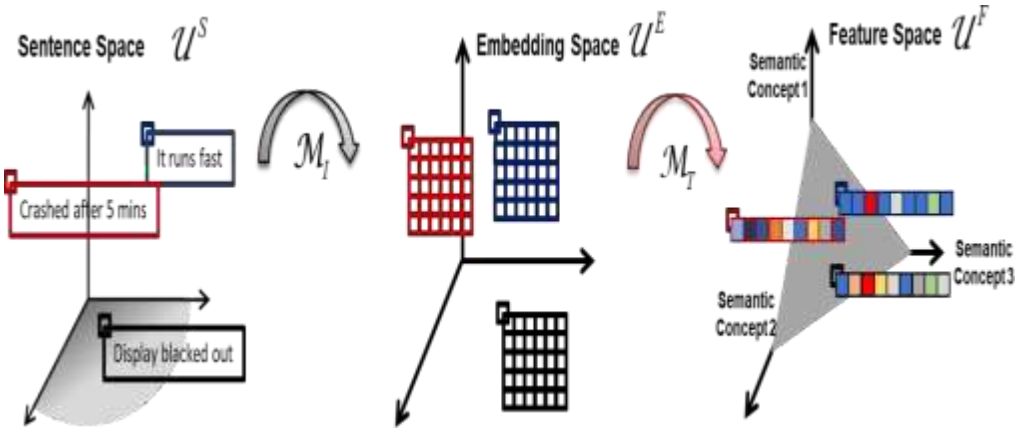

Figure 5- 2: Transformations from Sentences to Word Embeddings and Features

As feature learners, this experiment employs four neural networks: ANN, CNN, GRU, and LSTM. Each network's details are listed below.

5.3.1 ANN Representation Learner

After the embedding layer, a five-layer ANN with 1024, 512, and 256 units in each first, second, and third layer is formed, as illustrated in Figure 5- 3.

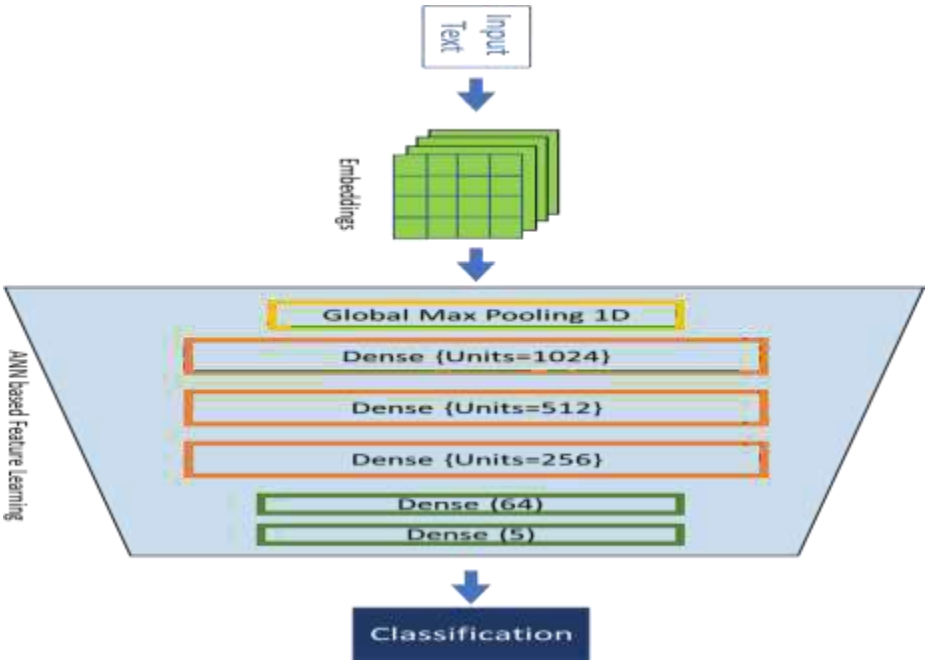

Figure 5- 3: ANN Architecture for NFR Classification

Before the first dense layer, a global max-pooling layer was utilised to provide a one-dimensional output from the embedding layer. Two more dense layers of 64 and 5 neurons were employed to generate the classifier's output.

5.3.2 CNN Representation Learner

This representation explores the development of features with context information between the words retained with 1D convolution in combination with ReLU.

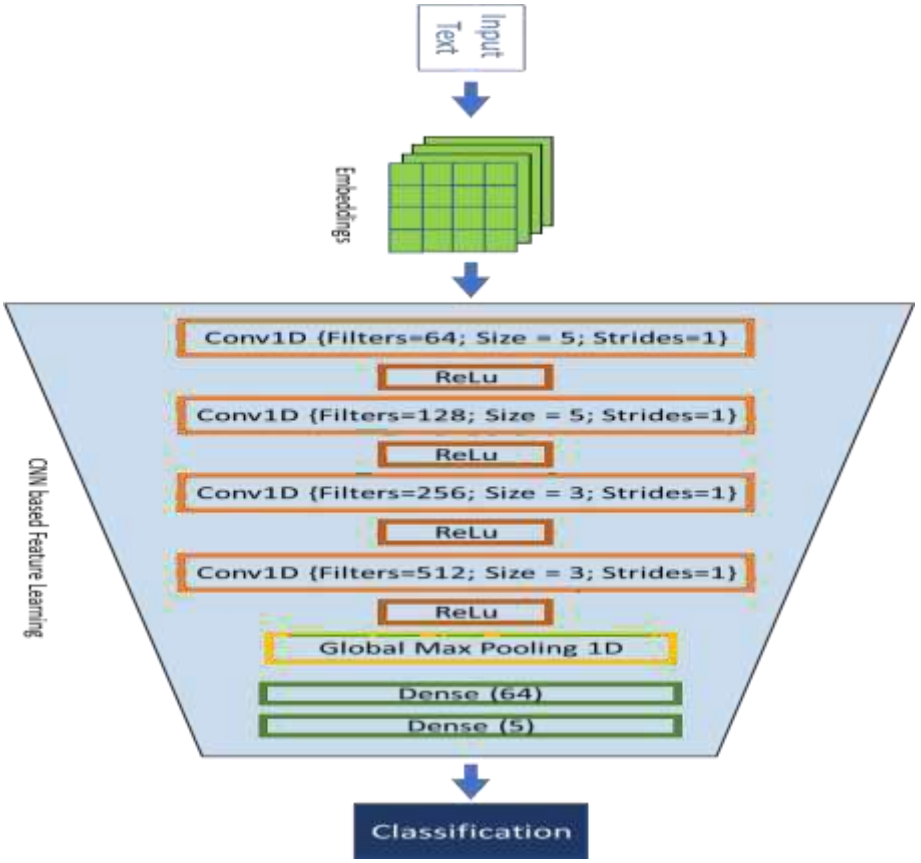

Figure 5- 4: CNN Architecture for NFR Classification

activations. Figure 5- 4 shows four layers with an increasing number of filters from 64 to 512 that have been used in this network. To obtain a low dimensional feature, the filter size of 5 is reduced to 3 in the other layers. A global max pooling is applied to the features before feeding them to the two dense layers for classification.

5.3.3 GRU Representation Learner

Recurrence plays a significant role in sequenced input. Therefore, this architecture explores the performance of a GRU-based feature learner employing 4 GRU layers followed by Tanh activation. A similar strategy used in CNN that increases units in each GRU layer is applied here, as shown in Figure 5- 5. Global max pooling with two dense layers concludes the feature learning network for classification.

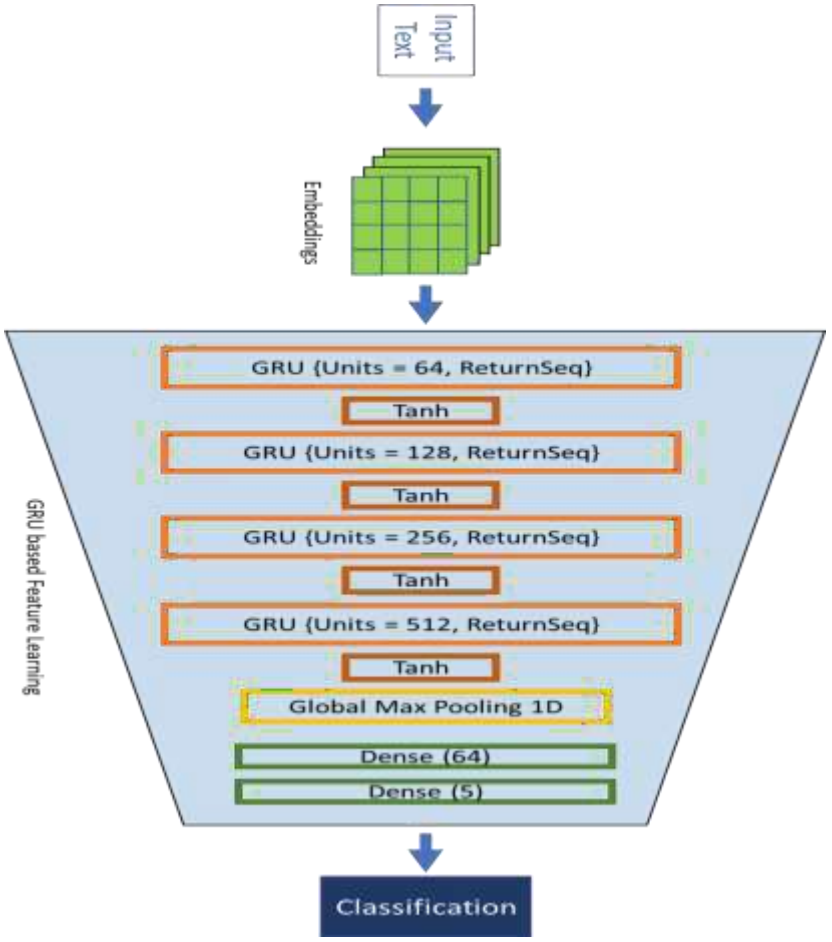

Figure 5- 5: GRU Architecture for NFR Classification

5.3.4 LSTM Representation Learner

LSTMs have recently beaten GRU-based networks' performance for many applications, including action recognition. In this experiment, LSTM-based feature learners have been used. Additionally, this network uses a dropout of 0.4 in each LSTM layer, as shown in Figure 5- 6.

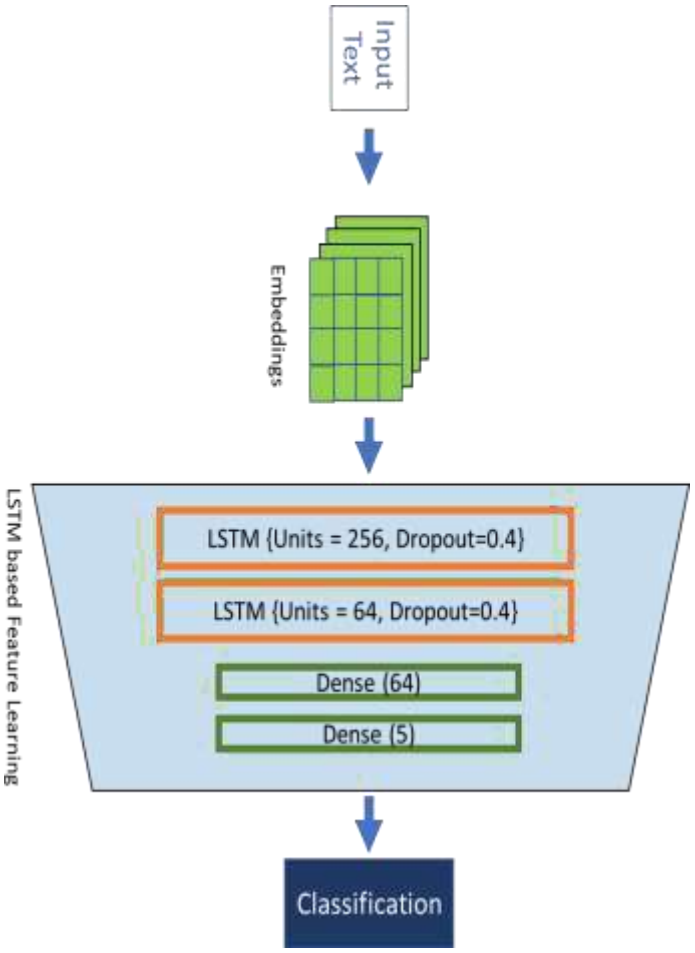

Figure 5- 6: LSTM Architecture for NFR classification

5.4 Classification

In related literature, well-known classifiers, such as Naive Bayes and SVM, were used to classify requirements. However, the softmax classifier has proven superior in multiclass classification problems. To categories the NFRs, this experiment uses a softmax activation in the dense layer of the feature learner. While the loss between correct and predicted class labels is reduced using categorical cross-entropy loss. The use of neurons equal to the number of classes in the dataset proved significant in this experiment.

5.4.1 Hyperparameter Settings

Hyperparameter tuning was discussed in Chapter 4, section 4.4. For the experimental setting in this experiment all DNNs have similar hyper-parameter configurations. For instance, an Adam optimiser was used to optimise all the networks with an initial learning rate of 0.001 and a rating decay of 0.2. Additionally, this experiment uses early stopping criteria, which causes the training process to cease after ten consecutive epochs of no improvement as opposed to using a predetermined number of epochs.

5.4.2 Hardware and Software Settings

All the experiments were conducted on a system with an Intel Core 2.80 GHz i7-7700HQ processor with 16 GB RAM and Nvidia GeForce GTX 1050Ti GPU. All the experiments were performed using Python 3.6.5 and TensorFlow 1.12.0.

5.4.3 Performance Metrics

This experiment evaluates the effect of learning on four deep neural networks, and the model's ability to learn and predict the classification results from the given data is assessed based on two parameters: accuracy [1]and loss[2]. Convergence graphs are generated to check the convergence rate of different approaches and plot the variation of loss and accuracy concerning increasing epochs. Furthermore, precision, recall, and f1-score is used as key indicators to evaluate the performance of the proposed framework for the NFR corpus.

[1] Accuracy is the ratio of number of correct predictions to the total number of input samples.
[2] Loss is false classifications

5.5 Experimental Results

As these models require an extensive training set, it would be interesting to see which model can learn and produce a satisfactory result with a small dataset, such as a custom NFR corpus. The aim here is to avoid overfitting when the DNN experiences a large difference in the performance over the training set compared to the validation set. experiment aims to minimise this gap and achieve higher accuracy for the classification of NFRs.

To provide a get a good picture of the NFR corpus to the reader, it is converted into a word cloud. Figure 5- 7 shows terms in the corpus generated after tokenisation and stop-word removal. The magnitude of each phrase (or two linked words) indicates how frequently a word appears in the corpus. For example, the most-used words in the corpus from the given figure are “user” and “system”. Other words (i.e., “easy”, “easily”, “must”, “help”, and “product”) have also been used extensively in the corpus.

Figure 5- 7: Word Cloud Generation for the Words in the Corpus

In terms of training accuracy and loss, CNN outperformed all four neural networks but accuracy decreased to the '60s for the validation set. Further investigation reveals that LSTM failed to perform well on such data due to many trainable parameters and insufficient data to train. However, the GRU network outperformed other neural networks on the validation set. Figure 5- 9 shows how the validation loss for GRU and

ANN approaches 1.66 and 1.70 respectively. The LSTM network, on the other hand, sees a significant increase. In Figure 5- 8, similar behaviour has been noticed for the accuracy of these networks.

It is clear that LSTM was underperforming. In addition to the four models Conv-LSTM, a hybrid architecture that combines CNN and LSTM, was examined. Convolution layers are utilised to extract features, which are then fed into the LSTM. Table 5- 4 demonstrates that it began to function better under these network parameters, although no noticeable benefit was observed.

Table 5- 4: Comparison of Various Representation Learning Approaches based on Statistical Performance Measure

Representation Learner	Augmentation	Precision	Recall	F1-Score
ANN	None	0.62	0.62	0.61
CNN	None	0.61	0.60	0.60
GRU	None	0.67	0.68	0.67
LSTM	None	0.06	0.25	0.10
Conv-LSTM	None	0.44	0.46	0.44

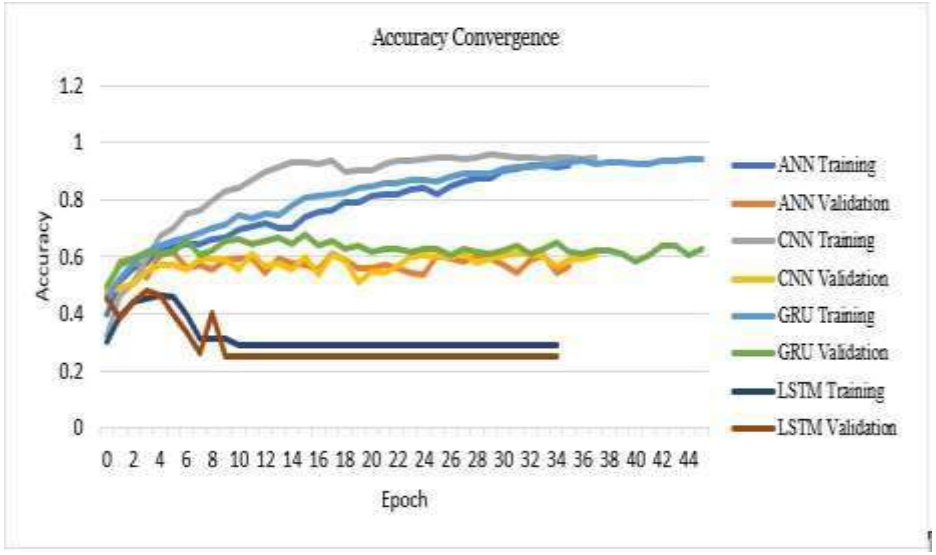

Figure 5- 8: Convergence Plots Concerning the Number of Epochs Vs Accuracy for the Baseline Models

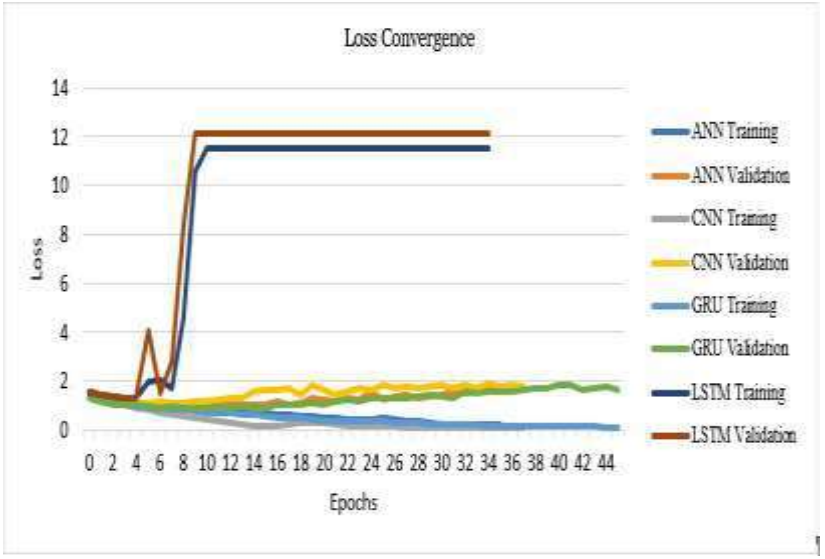

Figure 5- 9: Convergence Plots Concerning the Number of Epochs Vs Loss for the Baseline Model

5.6 Summary

This chapter explored the design considerations necessary for constructing an effective DNN based classifier. This chapter's focus was initially established as a multiclass classification, which was subsequently operationalised as an experiment. It showed how to categorise NFRs using DNNs. Furthermore, statistical measurements were utilised to evaluate the performance of the created models. Under the recommended settings, CNN and GRU outperformed ANN and LSTM, as shown in Table 5.1. The provision of an effective approach to stabilise the classification system is explained in this chapter, which is an important contribution of this thesis.

Chapter 6 Custom Data Augmentation Approach and Experimentation

The previous chapter demonstrated the classification of NFRs using four deep neural networks. The results indicate that, while some neural networks are capable of minimising validation loss, on average, they do not converge to a superior solution. This chapter discusses the issue of overfitting and motivates the use of a data augmentation strategy to solve data sparsity in deep learning-based classification. This chapter provides a major contribution to the research by investigating the effect of the data augmentation scheme to improve the performance of the baseline NFR classification system.

Section 6.1 provides a background to the data augmentation approach. It discusses work related to text augmentation techniques and the design considerations that must be addressed when developing an effective optimisation algorithm. Section 6.2 proposes a new data augmentation approach to probe DNNs for effectively classifying NFRs. The chapter further extends the investigation and formulate the training of the DNN models under these modifications. Lastly, it concludes with the results of the experiment.

6.1 Background

Data augmentation (DA) is a method of artificially producing new data by making minor adjustments to the existing training data (Wong et al., 2016). This study adopts data augmentation (Taylor and Nitschke, 2018) among other regularisation strategies (described in chapter 4 in section 4.6) for two main reasons. First, the deformation leads to an increase in acquisitions. Second, the data augmentation is controllable, the data modifications are chosen, and the data is thus precisely defined. It is, therefore, possible to fully determine the addition of complexity induced to the classification problem. DA inflates the dataset size artificially by either data warping or oversampling. Existing data is warped in such a way that the label is preserved.

Whereas oversampling generates synthetic instances and adds them to the training set (Wong et al., 2016).

Text augmentation is a relatively recent field that has evolved to deal with overfitting in DNN based text classification. The current data augmentation methods that are widely used in NLP are covered in this section.

A study by Zhang et al. (2016) uses a thesaurus to replace words with their synonyms to improve the training performance of a character-level convolutional neural network.

Rosario (2017) classifies short texts based on SVM. Whereas our work focuses on requirement classification based on a sentence level and concentrates on deep learning-based techniques.

Quijas (2017) proposed data augmentation techniques that use shuffling, noise injection, and paddling techniques to augment the textual data, to train convolutional and recurrent neural networks for text classification.

A year later, Kobayashi (2018) proposed an approach suggesting contextual augmentation which replaces the words in a sentence by its counterparts generated with a bi-directional language model. The experiment was performed with RNN and CNN trained across multiple datasets with favourable results.

Another work proposed by Coulombe (2018) is based on textual noise, spelling errors, synonyms replacement, paraphrase generation (using regular expressions or syntax trees), and back-translation techniques to generate more data.

Abulaish and Sah (2019) proposed a data augmentation approach that combines n-grams and LDA techniques to identify class-specific phrases to augment the corpus. They have evaluated the performance of the convolutional neural network on an original and augmented corpus and obtained positive results.

In another study, an easy data augmentation technique (EDA) was proposed using operations such as synonym replacement, random insertion, random swap, and random deletion. The researchers experimented with five tasks. Their findings imply that when training on smaller datasets, EDAs can enhance performance and decrease overfitting. Moreover, this study claims to improve performance for convolutional and recurrent neural networks (Wei and Zou, 2019).

In data augmentation, usually only the training set (containing images/videos/text) is augmented to address ML data thirsty algorithms (Wang et al., 2017). Nevertheless, lately, DA has emerged as a common practice to apply test-time augmentation (Ayhan and Berens, 2018; Shanmugam et al., 2020). The primary objective to augment test data is to reduce variance but not make the test data bigger or more accurate. However, this is typically so that the input data from the test set resemble the input data from the training set. For instance, in the case of the training images, AlexNet (Krizhevsky et al., 2017) and ResNet (Wang et al., 2021) performed the 10-crop technique in training and performed augmentation on the testing set. The training images were cropped in different locations/offsets. In contrast, only a single centred crop was performed at test time or in the second approach, and an average was taken after multiple random crops.

In previous studies, data augmentation was primarily performed with noise injections (Quijas, 2017), rotations, reverse translation, swapping, and random deletion (Wei and Zou, 2019). Techniques like these may lose valuable data and information.

A domain-specific method is another way of using a synonym thesaurus. These techniques were employed based on co-occurrence or semantic networks to create a synonym thesaurus. DA that uses a thesaurus replaces domain jargon or keywords with synonyms. Such strategies may be suitable for language-related tasks, which depend on an external dictionary. However, they require a long computation time and a high cost of implementation relative to performance gain (Zhang et al., 2016; Kobayashi, 2018). Furthermore, in a lexical-based method or the hand-crafted system, a domain expert develops a thesaurus (i.e., WordNet) by hand. Others, such as (Kobayashi 2018) bi-directional translations, can maintain the sense of the sentence but lose the jargon specific to the domain.

The real need for data augmentation lies in a domain-specific task, where it is expensive to gain a large corpus. However, the existing techniques of data augmentation in the text domain lack practices to handle domain knowledge. This gap provides a motivation for this study to discover a new technique to handle domain-specific data. The proposed approach is inspired by the data augmentation technique in practice in the image domain.

Blending photos by averaging their pixel values is a rather counterintuitive technique to data augmentation. The images formed by doing so do not appear to be beneficial to a human viewer.

In a study, Ionue (2019) demonstrated how sample matching can be turned into a useful enhancement approach. He randomly cropped and flipped images horizontally, resulting in a jumbled image that was used to train a classification model. The new image's label was like the original image, which was chosen at random. Another finding of the study was that merging photos from the full training set rather than instances strictly belonging to the same class yielded superior outcomes. This sample pairing creates a dataset of size N2+N from a size N training set.

Similar to this, Summer and Dinneen (2019) further explored the notion of combining images in an unintuitive way into a new training set using nonlinear methods. In like manner, Liang et al. (2018) employed GANs to create mixed images. Mixed images reduced training time and increased the variety of GAN samples in the training outcomes. Takahashi and Matsubara (2020) have demonstrated another method of picture mixing in which photos are randomly cropped, and the cropping is concatenated to produce new images. However, this strategy has a severe flaw in that it makes no sense from a human perspective.

6.2 Custom Data Augmentation (CDA) Approach

A new data augmentation strategy has been suggested, which is inspired by a query expansion process (Porter et al., 2020). It manipulates the data points themselves, in the creation of fresh data points resulting into a more comprehensive dataset.

This approach uses a “sort and concatenate” strategy for this purpose. The set of sentences from a class is sorted and then concatenated to the original sentences. As a result, it combines two sentences from each group to form more sentences. The class label is then assigned to the newly generated sentence. It is expected that mixing the requirements can keep the sentence semantic and maintain the syntactic structure. By adding information generated from a distinct set of requirements, this augmentation provides value to the base data. It can also significantly improve data quality by

reducing the amount of manual human input needed to add third-party libraries. Figure 6- 1 describes the process involved in this augmentation in the proposed approach.

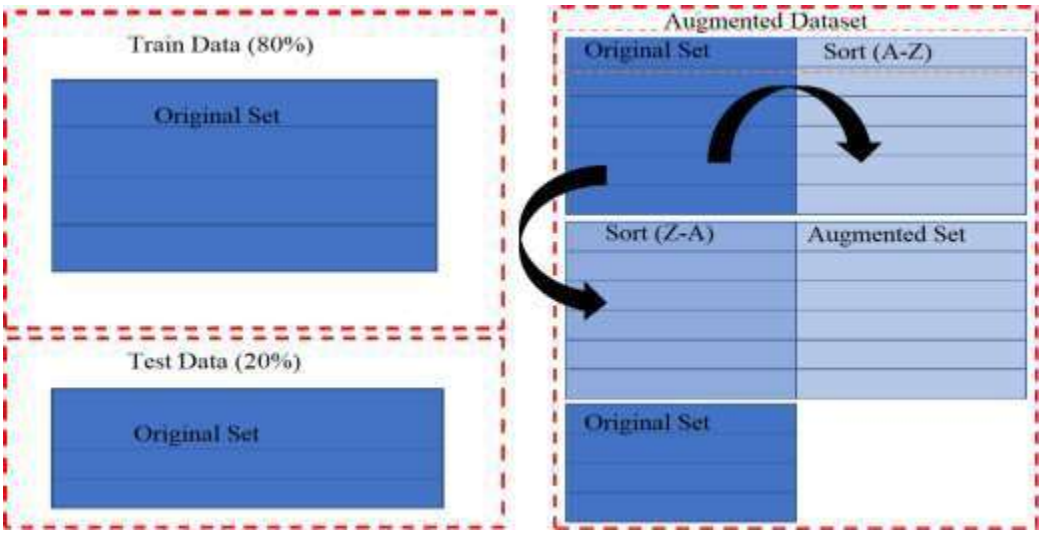

Figure 6- 1: A Framework Representing the Procedure for Custom Data Augmentation

This section presents a formalism for the custom data augmentation approach. It transforms the NFRs corpus to a distribution Q, used for training instead.

Steps for custom data augmentation (CDA) approach:

Input: *Original dataset D*

Output: *Augmented dataset Q*

Divide D into N classes

Sort *O in ascending order as* A_a

Concatenate *O with* A_a *as* OA_a *(Where O remains unchanged)*

Sort *O in descending order as* B_d

Concatenate *O with* B_d *as* OB_d *(Where O remains unchanged)*

Concatenate OA_a *and* OB_d

Repeat steps *2 to 7 until it reaches N.*

In the example described above, let 'D' denotes a dataset. Where *'N'* represents the number of classes. '*D'* can be divided with requirements R_n in each class, where n is the limit of the number of sentences for concatenations in each group. It takes one subset from the original data *'O'*. and creates A_a when sorted in ascending order and

B_d when in descending order. When performing a transformation on requirements, R_n, *'O'* remains unchanged. Lastly, all subsets are combined to develop *Q*.

This process represents the generic augmentation procedure; requirements belonging to every label/class have been concatenated within the same category to form a new set of samples in a dataset. Merging the content of one requirement with another one affects the entirely new requirements with the characteristic of both requirements.

The next step is to determine how the number of generated augmented sentences per original sentence yield performance boosts. The proposed approach increases the size of data in N^2. This dataset will be divided into a subset of equal size. Furthermore, it will be synthetically increased to double its size for every subset.

6.3 Training Configuration for DNNs with Data Augmentation and Pretrained Word Embeddings

This section presents an experiment to operationalise the proposed data augmentation approach. Similar to the baseline experiment, it trains four DNN architectures for the classification of NFRs. The pre-processing and sequence padding have been performed in similar manner. This experiment uses two data augmentation techniques:
1) easy data augmentation (EDA) and 2) the newly proposed custom data augmentation (CDA). The EDA approach used random insertion, deletion, random, swap, and synonym replacement operations. The CDA technique, on the other hand, concatenates two sentences from the same class to create more samples from that class. Both strategies were used to supplement the data over the whole corpus. Then, in each case, 80% of the augmented data was utilised for training, and the remaining 20 % was used for validation. The distribution for the augmented data is provided in Table 6- 1.

Table 6- 1: A Class-wise Distribution of Augmented Data Samples for the NFR corpus

Category	Original Data	Data augmented with EDA	Data augmented with CDA
Efficiency	480	1920	960
Maintainability	240	960	480
Portability	156	624	320
Reliability	191	764	382
Usability	417	1668	834
Total Documents	1484	5936	2976

6.3.1 Pre-trained Word Embeddings

In the baseline experiment, skipgram word embeddings were used. The NFRs corpus was used to train this skipgram model from the start. This experiment, however, employed a pre-trained weight on the Eng-CoNLL 17 corpus and finetune them while learning the feature representations for the provided data.

The Conference on Computational Natural Language Learning (CoNLL) is a collaborative effort to learn dependency parsers for a variety of languages in a real-world scenario without the use of gold-standard input annotations (Zeman et al., 2017). All the test sets used the same annotation scheme, which was called universal dependencies. The major purpose of the challenge was to learn syntactic dependency parsers that can be employed in a real-world setting and can handle a wide range of typologically diverse languages. These embeddings used 100-dimensional pre-trained word2vec continuous skip-gram vectors trained for an English CoNLL17 corpus with a vocabulary size of about 4.02 million words and fine-tuned the pre-trained weight on the NFRs augmented data.

6.3.2 Hardware/ Software Settings

The experiment is performed with the same neural network architecture as the previous one, but with a new augmented data distribution, as shown in Table 6- 1. After

initialising the embedding layer with pre-trained weights, the embedding layer is left trainable to maintain the accelerated pace of convergence. Instead of employing a fixed number of epochs an early stopping criterion is used, which terminates the training process if there are no improvements for ten consecutive epochs. All the networks were optimised using an Adam optimiser with an initial learning rate of 0.001 and a rating decay of 0.2

6.4 Experimental Results

The results of the baseline NFR classification model designed in Chapter 5 showed that there was not enough data to train the designated embedding layer. Therefore, to solve this issue, this chapter introduced a data augmentation strategy. Additionally, pre-trained word embedding technique is employed in this experiment. In section 6.3, an experiment was conducted with pre-trained skip-gram word embeddings, and the entire corpus was augmented before splitting into a train/validation set with two data augmentation strategies: state of the art EDA and newly proposed CDA, respectively.

In this section the results are evaluated based on accuracy and loss on the training and validation sets. Most of the models perform well on the training set. However, the problem under analysis is in generalising on the validation dataset and minimising the loss. The results obtained from the EDA approach shows, this modification could not drastically improve the results; however, the training time was reduced. Figure 6- 4 shows the accuracy and loss values while indicating that CNN and GRU pre-emptively stopped under 20 epochs. However, in the instance of the CDA approach, it was discovered that, in addition to the ANN, CNN, and GRU networks that were already performing well, the LSTM network also performed well with this augmentation technique. However, until the first ten epochs, it showed no signs of learning, as shown in Figure 6- 5.

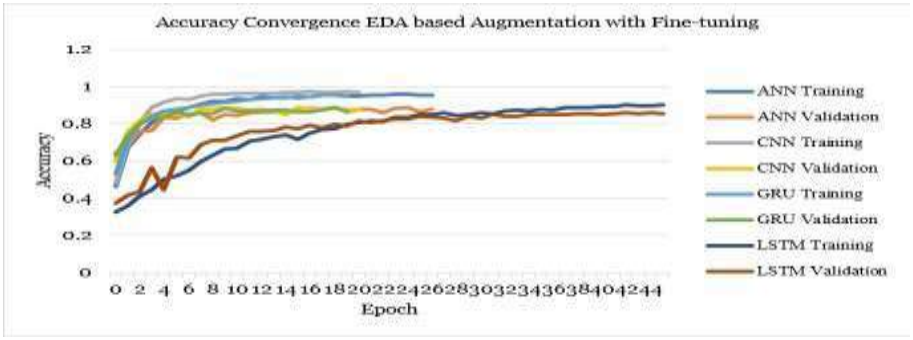

Figure 6- 2: Convergence Plots Concerning Several Epochs for Accuracy on Entire Corpus with EDA and Pre-trained Word Embeddings

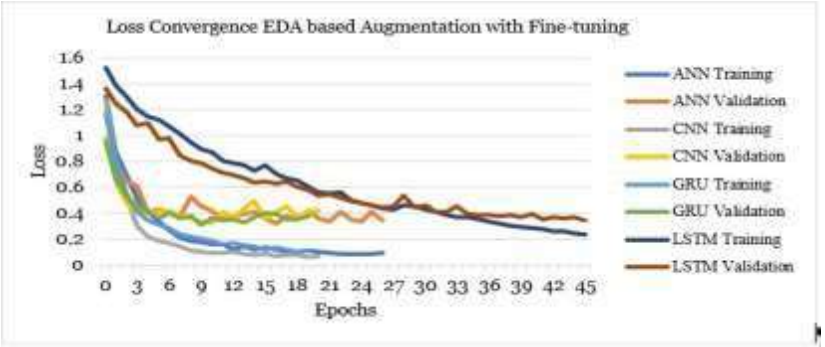

Figure 6- 3: Convergence Plots Concerning the Number of Epochs for Loss on Entire Corpus with EDA and Pre-trained Word Embeddings

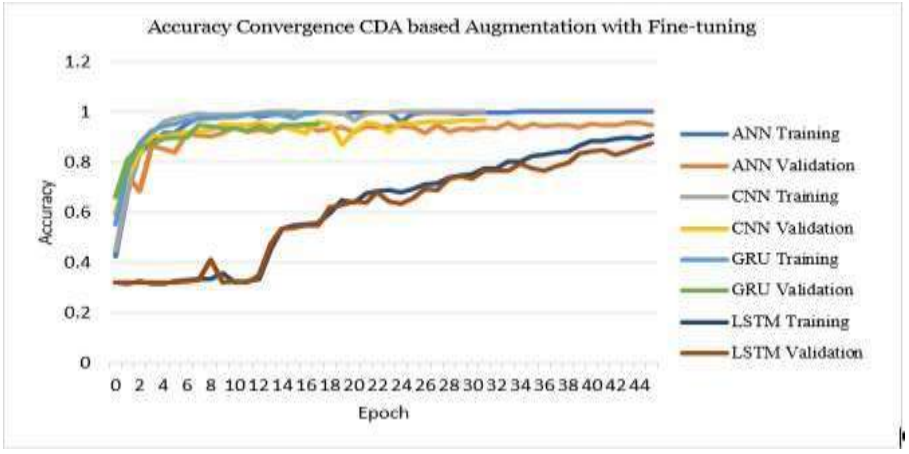

Figure 6- 4: Convergence Plots Concerning Several Epochs for Accuracy on the Entire Corpus with CDA and Pre-trained Word Embeddings

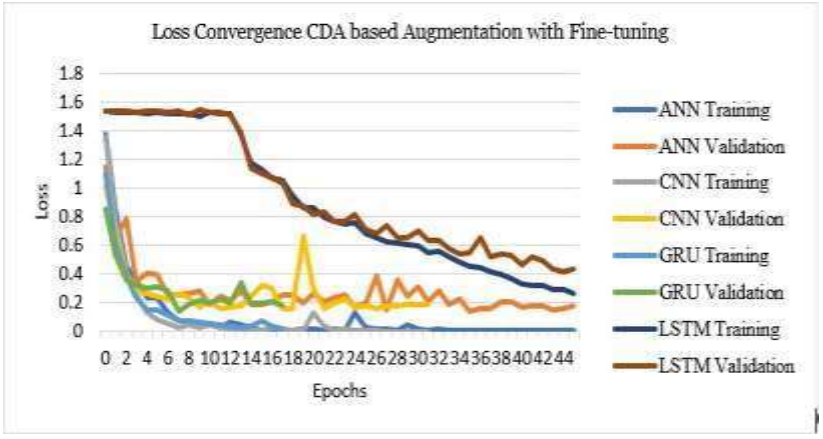

Figure 6- 5: Convergence Plots Concerning Several Epochs for Loss on the Entire Corpus with CDA and Pre-trained Word Embeddings

6.5 Analysis of Classification Models

The efficacy of the proposed method relies on the data augmentation technique, data distribution set for augmentation, word embedding, and model's architecture. These models' results are calculated based on precision, recall, and f1-score.

Table 6- 2: Comparative Analysis of the Results Among the Baseline, EDA, and CDA Approaches

Method	Baseline Model with skip-gram trained from scratch			EDA Approach with Pre-trained Embeddings			CDA Approach with Pre-trained Embeddings		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
CNN	0.61	0.60	0.60	0.89	0.88	0.88	0.95	0.96	0.95
GRU	0.67	0.68	0.67	0.86	0.86	0.86	0.93	0.94	0.93
ANN	0.62	0.62	0.62	0.88	0.88	0.88	0.95	0.94	0.94
LSTM	0.06	0.25	0.15	0.84	0.85	0.85	0.87	0.89	0.88

In the baseline experiment, the results from Table 6- 2 suggest that GRU generalised extremely well with minimum loss over the validation set, which indicates that it can work with a small dataset such as a custom NFR corpus. In contrast, the LSTM fails to perform with a small dataset. However, CNN also performed satisfactorily. Furthermore, by closely observing the results, it is concluded that using augmentation in a combination of pretrained embedding improves the results from the baseline model. At the same time, neural networks such as GRU and CNN performed well with EDA based augmentation used with pre-trained word embeddings. Table 6- 2 shows that the same representation outperformed all the previous approaches when used with the CDA approach. The model's overall performance for EDA improved from the 60s to 80s. Whereas, the CDA approach achieved the highest results for CNN, reaching 95% for the classification of NFRs. At the same time, LSTM appeared to be the worst-performing model in each experiment. The CDA method resulted in a 2x increase in data. However, in EDA, this was not feasible. As previously stated, the CDA technique retains domain vocabulary, which provides some evidence that the pre-trained model

learned greater semantic relatedness in the custom enhanced corpus. However, with supplemented data using EDA, its actions (random deletion/random insertion) provided more data but could not retain sentence semantic relatedness; this is shown in this experiment as pre-trained word embeddings could not recognise a comparable pattern. As a result, the CDA strategy outperformed the EDA and baseline models, even with fewer enhanced samples.

6.6 Summary

At the chapter's outset, a thorough examination of existing data augmentation was presented. In Section 6.2, an innovative method of custom data augmentation was proposed with possible benefits of combining data augmentation frameworks with transfer learning.

This chapter discussed how to apply the custom data augmentation strategy as an optimisation technique for addressing the issue of data sparsity and overfitting while training DNNs. The performance of the baseline classifier and EDA was compared when training DNN models on the NFRs corpus. This chapter made a significant contribution in terms of a data augmentation by describing a practical approach for stabilising DNNs for the classification of NFRs. The results in Table 7-3 showed that the suggested method is more successful than the EDA approach and baseline classifier under the proposed settings

Chapter 7 Extended Experiment and Detailed Analysis of Results

This chapter discusses the effects of two methods of data augmentation and pre-trained word embeddings on an NFR classification system. Previously, for DNN training, the data augmentation technique was used under a different data distribution where the full corpus was enhanced before being separated into train/validation sets. In contrast, this chapter explores an alternative method to data augmentation in order to develop a classification system for NFRs. Additionally, it gives a comprehensive analysis of the results using a variety of statistical performance measures for the various experimental setups mentioned in this thesis and explores this study's final objective to analyse the generalisability of the designed NFR classification system for the selected NFRs.

A discussion on all previous experiments performed in Chapters 5, and 6, are discussed and analysed in section 7.1. Section 7.2 explores an alternative optimisation approach in the form of a new experiment. Section 7.3 extends the investigation to the best-performing model based on all experiments and further explores the generalisability of the model, hence, finally providing the classification insight for each NFR attribute.

7.1 Background

This chapter provides an extended experiment to optimise the design and develop an automatic system to classify non-functional requirements in multiple classes based on deep learning techniques. DNNs require a large, annotated corpus and are prone to overfitting; therefore, to overcome these shortcomings, a unique approach for data augmentation named custom data augmentation (CDA) approach was proposed in chapter 6. In the initial experiment, data augmentation was performed on the entire NFR corpus before splitting the data into train/validation sets to acquire a set of sentences that belong to a consistent niche. The performances were analysed based on the comparison among the baseline model (no augmentation) and a state-of-the-art EDA with pre-trained word embeddings on Eng-CoNLL 17 corpus. The overall results

indicate that all the DNNs improved with both data augmentation strategies. However, the proposed CDA approach performed relatively well compared to the EDA due to its significantly rich class representative data.

These experiments address the multiclass classification based on the custom NFRs corpus, which appears to have an imbalanced distribution of samples against each class. When a simple train/validation split is performed randomly, it is basic machine learning nature to divide the train and validation set disregarding the distribution or proportion of the classes. There is a chance that all variations of one data will end up in the same set, which can cause bias in the results. Therefore, a counterstrategy in the second round of the experiment is adopted to ensure that this technique is valid. In this chapter, the study aims to perform a train/validation split manually. According to the results from the previous experiment, the CNN turned out to be the best performing model; therefore, the CNN's convergence and learning behaviour will also be studied under the new settings, and a comparison with previous results will be provided in detail.

7.2 Training Configuration for a Baseline Classification Model

To perform an experiment for NFR classification, the same four deep neural networks, ANN, CNN, GRU, and LSTM, were selected, along with the custom NFR corpus. In this experiment, the difference is that the corpus is manually divided into training and validation sets based on the 80 and 20 ratios. Table 2 provides the train/validation distribution used in this experiment.

To design a baseline classifier, DNNs were designed with the original train set shown in Table 7- 1, with skip-gram word embeddings trained from scratch. All these DNNs were tested with the original validation set. When data is manually divided into train/validation sets, it ends up in a train/validation set different from the train/validation split that was automatically performed in Chapter 5, section 5.3.

Table 7- 1: Data Distribution for the Custom NFR corpus with Augmented Data

Category	Original Data	Original Training Set	Custom Augmented Training Set	EDA Training Set	Original Testing Set	Custom Augmented Testing Set
Efficiency	480	384	768	1536	96	192
Maintainability	240	192	384	768	49	98
Portability	156	124	248	496	32	64
Reliability	191	152	304	608	39	78
Usability	417	330	660	1320	84	168
Total samples	1484	1182	2364	4738	300	600

7.2.1 Experimental Results

The classifier results can be seen in Table 7- 2 An interesting finding is that GRU performed better than other models with such a small dataset, and it is observed that LSTM appeared to be the least favourable for this task. These results are in line with the results from Table 6- 2, where these DNNs were trained with data distributed automatically between train and validation sets. The accuracy falls at approximately 50% for ANN, CNN, and GRU training.

7.3 Training Configuration for DNNs with Data Augmentation on the Train Set

Similar to the previous experiment this approach aims to see if any data augmentation strategy, specifically the CDA technique can help to boost the performance of the proposed classification model. The augmentation is performed on the training set only to validate the efficacy of the suggested approach. Whereas, the original validation set, as stated in the Table 7- 1 is used for testing the model's performance.

In the case of the EDA, training samples were increased from 1182 to 4738, whereas, with the CDA approach, they increased to 2364 from 1182. An original validation set, which is the most convenient method in ML, is used for validation. This experiment

uses the same pre-trained word embeddings as the previous experiments performed in section 6.3.

7.3.1 Experimental Results

The results for the experiment can be seen from Table 7- 2 which shows that in a case where only the training set is augmented, the EDA-based approach does not give any boost to the results. With the introduction of this augmentation strategy, only the performance of LSTM was improved over the baseline model. On the other hand, the CDA approach performed better than EDA in the previous settings (as can be seen from 6.3), and CNN outperformed all the previous results for the classification of NFRs. However, in the current approach shown in Table 7- 2 using data augmentation on training set alone is marginally better in some cases than not using data augmentation. The model using the baseline and EDA approach produced similar results for GRU, but the performance appeared to be declined when trained with the CDA approach. Contrastingly, for ANN, CNN, and LSTM, a slight improvement was observed. The results show that even when using the same pre-trained word embeddings, DNNs could not generalise under this data distribution. These deep learning models could not identify and learn features on the validation data, as there is absence of the same representation found and learned from the training data. Overall, CNN trained with CDA approach turned out to be favourable under both data situations with these settings. Therefore, the classification of NFRs is analysed based on the CNN model.

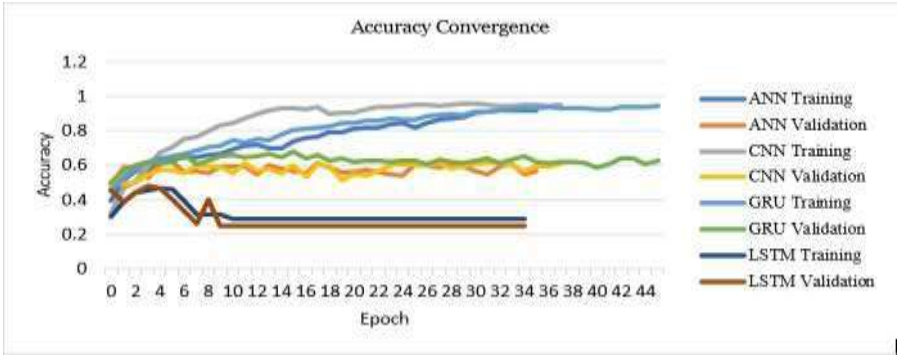

Figure 7- 1: Convergence Plots Concerning the Number of Epochs for Accuracy with the Baseline Models

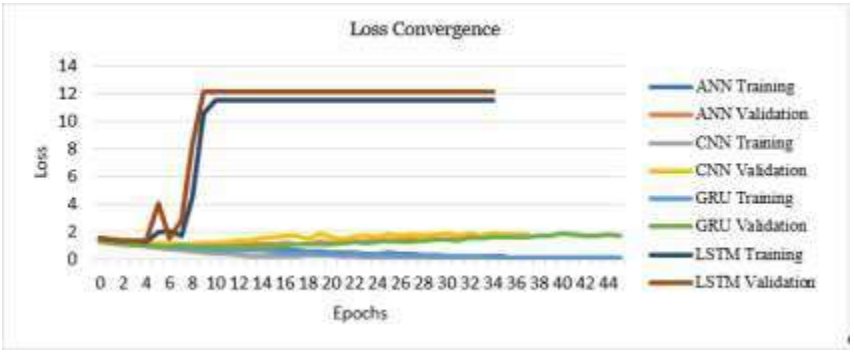

Figure 7- 2. Convergence Plots Concerning the Number of Epochs for Loss with the Baseline Models

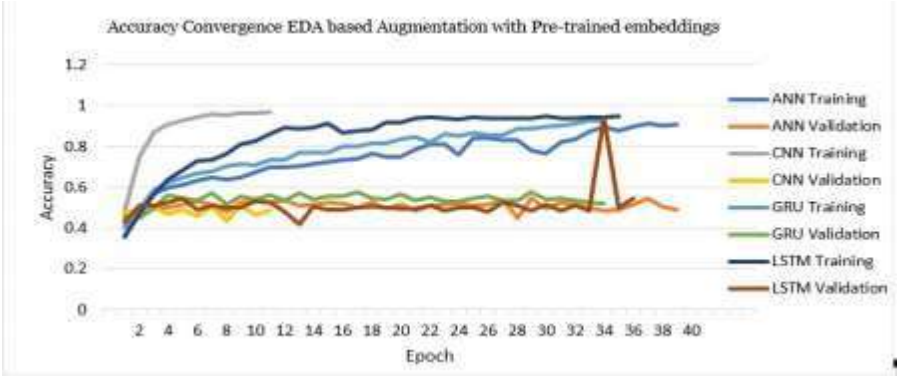

Figure 7- 3:Convergence Plots Concerning the Number of Epochs for Accuracy on the Trainset with EDA and Pre-trained Word Embeddings

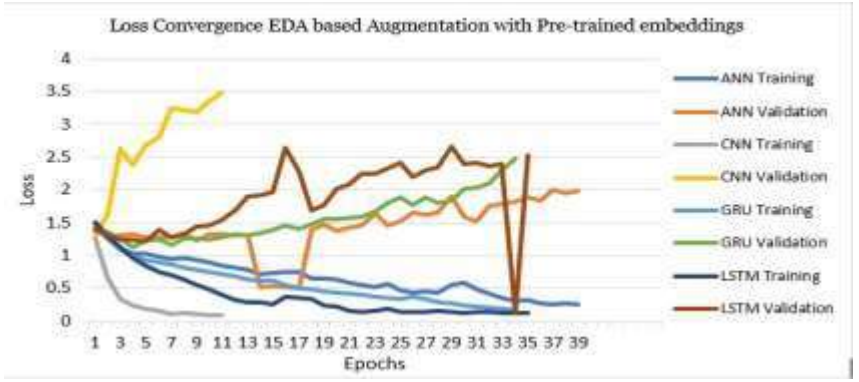

Figure 7- 4: Convergence Plots Concerning the Number of Epochs for Loss on the Trainset with EDA And Pre-trained Word Embeddings

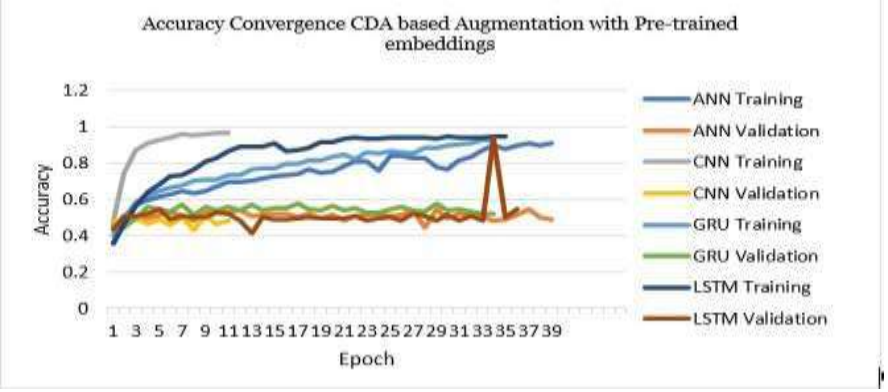

Figure 7- 5: Convergence Plots Concerning the Number of Epochs For. Accuracy on the Trainset with CDA and Pre-trained Word Embeddings

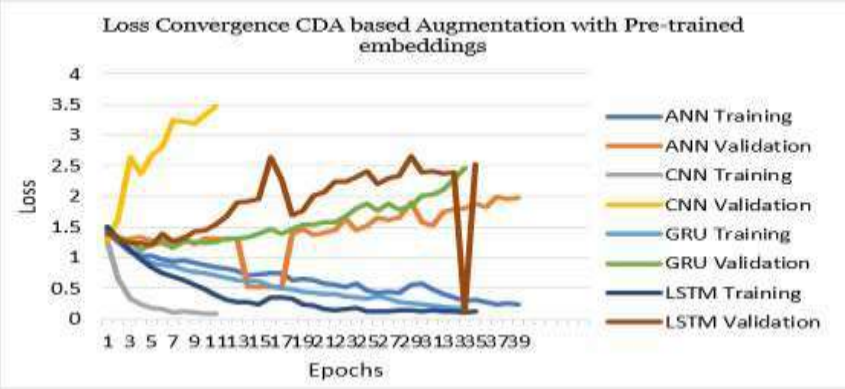

Figure 7- 6: Convergence Plots Concerning Several Epochs for Loss on the Trainset with CDA and Pre-trained Word Embeddings

Table 7- 2: Comparative Analysis of the Results Among the Baseline, EDA, and CDA Approaches

Model	No Augmentation, No Pretrained Embeddings			EDA with Pre-trained Embeddings			Custom Augmentation with Pre-trained Embeddings		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
ANN	0.50	0.49	0.49	0.48	0.42	0.45	0.48	0.46	0.47
CNN	0.48	0.49	0.49	0.44	0.43	0.43	0.51	0.44	0.48
GRU	0.53	0.55	0.54	0.51	0.50	0.50	0.47	0.48	0.47
LSTM	0.03	0.16	0.09	0.45	0.41	0.43	0.47	0.46	0.46

7.4 Training Configuration for CNN with Custom Data Augmentation on Train/Validation Sets Separately

To check whether the entire classifier training and testing process was adequately developed, another experiment was conducted with CNN. In this experiment, the CNN was trained with the data augmented with the CDA approach, where the validation set was also separately augmented with the same approach. Table 7- 1 shows that the amount of data increased for training and testing split for each class using Custom data augmentation.

7.4.1 Experimental Results

In this section, data augmentation is performed on the NFR corpus with the manual distribution of the corpus into train and validation sets where both sets were augmented separately. Table 7- 3 shows the results for this approach; it increased the model's performance by an average factor of 10% giving us the precision of 0.59, recall of 0.55, and F1-score of 0.57. These results suggest that to produce better results, the validation set also requires some level of augmentation to find a similar feature to the training set.

7.5 Analysis of CNN Classification Model

This section compares and evaluates the effectiveness and performance of the data augmentation under two distributions for the NFR corpus. As identified in the previous section and shown in Table 6- 2, CNN with pre-trained embedding appeared to be the best classifier for NFRs when the entire corpus was augmented with the CDA approach. This comparison aims to present an analysis to explore how NFRs (efficiency, reliability, usability, maintainability, and portability) were individually learned by the CNN classifier. The efficacy of CNN performance when trained with pre-trained word embedding and data augmentation performed on a different set of data with CDA approach to provide answers to the last objective of this thesis.

Table 7- 3: Comparative Analysis of the Results for CNN Augmented with the CDA Approach on Train Sets vs Train/validation sets

Non-functional requirements	CNN results for CDA on the entire corpus			CNN Results for CDA with manual data distribution					
				CNN Results with CDA on train/validation sets separately			CNN Results with CDA on the train set only		
	Precision	Recall	F1-Score	Precision	Recall	F1-score	Precision	Recall	F1-score
Efficiency	0.95	0.98	0.97	0.64	0.65	0.65	0.45	0.50	0.47
Maintainability	1	0.92	0.96	0.57	0.54	0.56	0.60	0.24	0.42
Portability	0.88	1	0.94	0.90	0.58	0.74	0.67	0.37	0.52
Reliability	0.93	0.98	0.95	0.19	0.12	0.15	0.28	0.43	0.36
Usability	0.98	0.92	0.95	0.63	0.85	0.74	0.57	0.64	0.61
Average	0.95	0.96	0.95	0.59	0.55	0.57	0.51	0.44	0.48

Finally, Table 7- 3 compares the CNN results for the classification of all five NFRs. The performance of the CNN with the previous approach reached 95% precision. In contrast, it reached only 50% when only the train set was augmented. However, it slightly improved, reaching nearly 60% when the validation set was also augmented with the CDA approach.

Another interesting observation from these results is that when tested with original data and when the validation set was augmented separately, CNN showed similar learning behaviour concerning all classes, reaching its highest recall for usability in both cases. Similarly, portability reached the highest precision under both settings as compared to the rest of the classes, whereas when trained with augmented data before dividing it into train/validation sets, CNN seemed to have completely different behaviour. Only 88% of positive class predictions that belong to the positive class were created out of all positive examples in the dataset, where portability received 100% recall, but appeared with the lowest f1score, balancing the concerns of both accuracy and recall.

7.5.1 Convergence of the CNN Network

The comparison of the convergence rate of the CNN trained with the previous approach and the one presented in this chapter can be seen in the following figures.

Figure 7- 7 gives a clear indication that the CNN could not minimise the loss for validation when only the validation set was augmented with the CDA approach. Figure 7- 8 describes the results from the experiment where augmentation was performed on both train/validation sets individually. Finally, Figure 7- 9 shows the results of the previous approach where data was augmented before being divided into the train/validation sets. The figure shows that the CNN converged around the 40^{th} epoch, achieving the minimum validation loss.

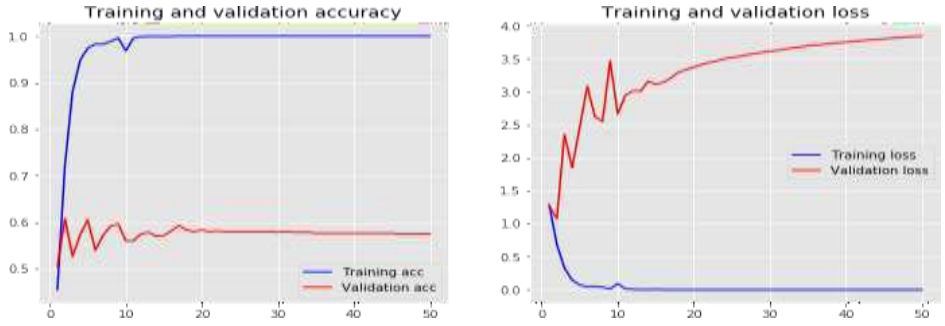

Figure 7- 7: CNN Convergence with Only Train Set Augmentation

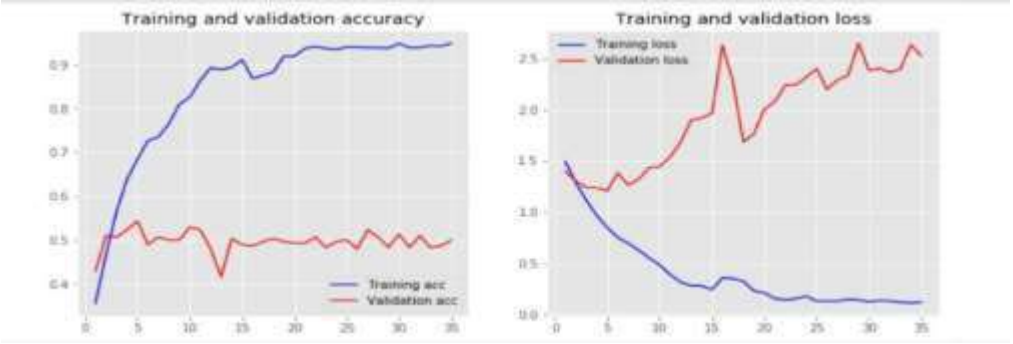

Figure 7- 8: CNN Convergence with Both Train/Validation Augmentation Performed Individually

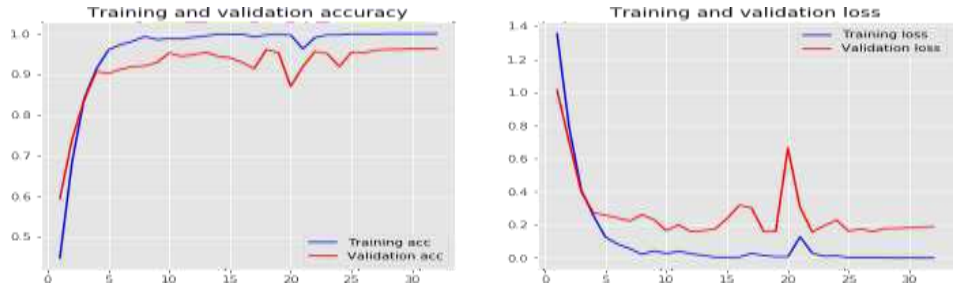

Figure 7- 9: CNN Convergence with Train/Validation Augmentation Performed Before Data Split

7.6 Generalisability of the CNN Classifier for NFRs

This section delves deeper into the outcomes of a CNN-acquired confusion matrix, comparing the results from the experiments in which the train/validation split has been acquired before augmentation without augmenting the test set versus the one in which the test set has also been augmented, as shown in Figure 7- 11 and Figure 7- 12 While Figure 7- 10 represents the previous approach with superior results in which the augmentation has been carried out before train/test split using the CDA approach.

To gain a closer understanding of the results, in this section, a confusion matrix is used to evaluate the category-wise performance for the classification over the best performing classifier. This confusion matrix demonstrates the difference between ground truth labels and the predicted labels of the experiment for each class. Another thing that is quite visible through confusion matrices is the number of misclassified samples in each category. Looking closely at the confusion matrix, the values given in diagonal represent the correctly classified sample, also known as the true positive.

7.6.1 CNN Results for Custom Data Augmentation on the Entire Corpus

This experiment achieved 592 samples in the validation set as a result of train/validation split after data augmentation. It can be seen from the confusion matrix that 22 out of 592 samples were incorrectly classified, spreading over various NFRs classes, as evident from Figure 7- 10. Upon further analysis, it was observed that classes, (i.e., maintainability and usability) had the most inaccurate predictions. In contrast, efficiency and reliability had only minor errors. Portability was learned extremely well; not a single sentence belonging to this class was classified incorrectly.

7.6.2 CNN Results for Custom Data Augmentation on the Train Set Only

To investigate whether the relative data augmentation approach is not constrained to data selection, augmentation was only performed on the training set using the CDA approach. Table 7- 3 compares the results, showing that the earlier method achieves a relatively higher result.

Figure 7- 11 reveals that usability received the highest true positive. Out of a total of 84 usability samples, 55 were accurately classified. However, usability was confused with efficiency 24 times. Only 48 instances were classified successfully, while it was misclassified as reliability 26 times.

7.6.3 CNN Results for Custom Data Augmentation on the Train/Validation set Separately

The Figure 7- 12 shows the result for these settings. The CNN learned features to identify usability and correctly classified it 142 out of 168 times. Efficiency was correctly classified 124 times from 192 samples. efficiency was classified as reliability 28 times. The worst performance is observed with reliability, which was incorrectly classified as usability the most frequently.

An interesting observation from these two confusion matrices is that the CNN learned all five classes similarly under both data variations. The overall performance improvement is observed due to the increment in the size of the data validation sample.

This indicates that CNN based model can even produce improved results if the train and validation set have more data; hence more learnable features can produce higher results. The overall results indicate that data augmentation alone on the train set does not work well in this situation. The CNN cannot learn the relationship between the two, and the validation set is considered entirely new for the classifier. In second scenario, where the CNN was trained and tested with data augmented on both (train/validation) sets, it started to improve, though not to the level of the first

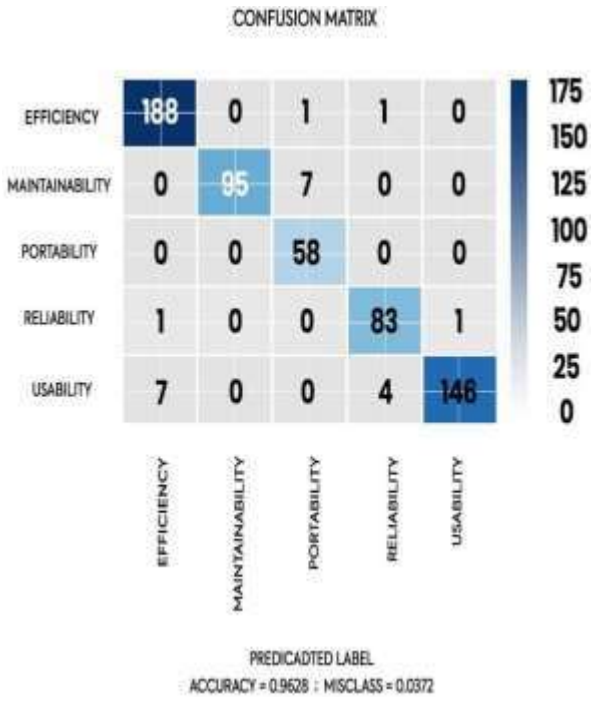

Figure 7- 10: Confusion Matrix for CNN Results for Custom Data Augmentation on the Entire Corpus

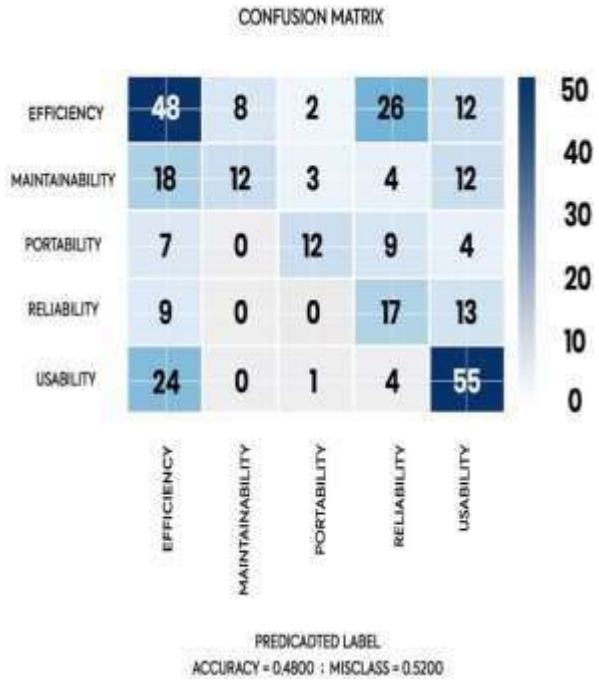

Figure 7- 11: Confusion Matrix for CNN Results for Custom Data Augmentation on the Train Set Only

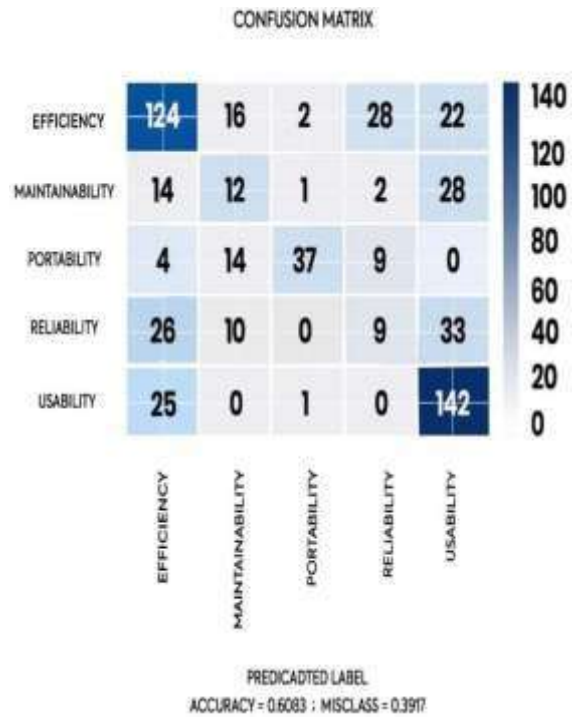

Figure 7- 12: Confusion Matrix for CNN Results for Custom Data Augmentation on Train/Validation Set Separately

experiment's findings. In conclusion, these models need some augmentation on the validation set to find relationships to improve classification results.

Another influencing factor on the performance of these models is the imbalanced number of instances in the dataset, as this imbalance affects the overall accuracy of the classifier. It is seen that the classes with more samples were learned better than those with a smaller number of examples.

7.7 Summary

This chapter explored the design considerations necessary for constructing an effective DNN based classification model and demonstrated how to optimise this approach using a data augmentation technique. The chapter began by implementing a new approach for classifying NFRs based on two data augmentation methods applied to the training set exclusively. Using this alternate formulation, it was demonstrated that CDA outperforms EDA with marginal difference. Compared to the prior approach, which enriched the full corpus before separating it into training and validation sets, the earlier method was shown to be more successful, even though both methods use identical pre-trained word embeddings and DNN architectures.

Chapter 8 Conclusion, Limitations, and Future Recommendations

A synopsis of the study and the contributions of this thesis are outlined in this last chapter. A critical study of the completed work is carried out, with an emphasis on the strengths and shortcomings of the research. It also emphasises potential future advancements in this discipline and the direction in which researchers in this rapidly expanding field are moving.

Section 8.1 summarises the approaches that have been presented based on the aim and objectives of this research. The critical findings of the study conducted are discussed in Section 8.2. Finally, in section 8.3, various future research directions are suggested.

8.1 Summary of the Thesis

Chapter 1 initially proposed the multiclass classification of NFRs for this research. This chapter described the motivation for this study and the research objectives to be further investigated later in the study.

Chapter 2 provided an extensive literature review of existing methods used to classify FRs and NFRs. The literature was discussed in terms of four aspects of classification: the corpus, feature engineering techniques, representation algorithms used for training, and evaluation parameters used to analyse the classification results.

Chapter 3 provided a theoretical background in defining NFRs. This chapter helped to identify critical NFRs that served as base labels for this study. It further collected data to create an NFR corpus, and finally, it presented a framework for corpus annotation to obtain the gold standard through a crowdsourcing approach.

Chapter 4 discussed a detailed background for the deep learning model, architecture design, and hyperparameters specific to the experiment.

In **Chapter 5**, four deep learning models under the specified network parameters were trained to design a baseline classifier.

Chapter 6 addressed a background and related work for text augmentation to provide the theoretical background for the proposed method. Pre-trained word embeddings were introduced as part of a data augmentation strategy. The proposed augmentation approach and EDA approach were used to augment the NFRs corpus, which was then followed by another round of tests to assess the implications of the representation model performance.

In **Chapter 7**, A revised experiment was performed under manual data distribution of the corpus. All the experiments performed in chapter 6 were evaluated and compared against the research aim (described in chapter 1).

It is noted that the focus of this thesis was on the use of deep learning models to classify non-functional requirements. The core idea presented in this thesis is 1) The creation of domain corpus for non-functional requirements and 2) A multi-class classification system for NFRs.

This thesis included a range of experiments to address the study objectives listed below. As a result, related to the presentation of a response to the research aim, the following key findings were reached:

Objective 1: To obtain a single-label NFR corpus based on a representative sample.

The NFRs used in this study were found in at least five of the six basic software quality models: Dromey's, FURPS/FURPS+, ISO9126, ISO 52010, Boehm, and McCall's. As a result, this study came up with five NFRs: reliability, usability, maintainability, portability, and efficiency. Data was acquired from an online library (SCRIBD) in the form of Software requirements specification (SRS) documents, and then requirements for these selected classes were manually extracted from these SRS documents. As a result, the Custom NFRs corpus was created.

Objective 2: To design a framework for building a gold standard multilabel NFR corpus.

The study proposed a framework for crowdsourcing the creation of a gold standard multi-label NFR corpus.

The experiment was performed using an online crowdsourcing platform to annotate the NFRs. The task was assigned to three annotators with the data and instructions. Finally, the gold standard quality was determined using Cohen's Kappa calculator based on inter-annotator agreement. Regrettably, this experiment is restricted to one iteration owing to the figure-eight platform's unavailability. However, the theoretical contribution of this task is still significant. It emphasises that while crowdsourcing systems are cost-effective for generic annotation tasks, using them for domain-specific problems poses hurdles.

Objective 3: To design a deep learning architecture most appropriate for the classification of NFRs.

The experiment involves four neural networks to create a classification system for NFRs: ANN, CNN, GRU, and LSTM. The distribution of datasets was kept consistent throughout all networks. The findings indicated that GRU performed better on the training set and generalised similarly well on the validation set when trained with such a small dataset. LSTM, on the other hand, appeared to be the least suitable for this purpose.

Objective 4: To investigate the effect of data augmentation approach and pretrained word embeddings over the performance of the baseline NFR classification system.

This experiment addresses the constraints regarding the size of the NFR corpus while using word embeddings for semantic feature learning. To enhance the efficacy of the feature learning process the suggested data augmentation technique using pre-trained word embeddings. This experiment used the NFR corpus augmented with CDA and EDA, as well as skip gram word embeddings pretrained on the Eng-CoNLL corpus to train four neural networks.

The entire training was conducted under two modifications in the augmentation process; 1) the entire corpus was augmented before diving into the train/validation set. 2) Only the train set was augmented.

The results suggests that the first approach, when these pre-trained embeddings were used with EDA, this modification not only improved the results, but the training time

was also reduced. Even with a smaller number of generated augmentations, the proposed CDA approach performed reasonably well in comparison to the EDA-based system due to its significantly rich class representative data. The CNN trained with CDA, and pre-trained embeddings outperformed all the previous results for the classification of NFRs.

On the other hand, in the second experiment, we only saw a minor difference in comparison to the baseline and EDA On further investigation, when the validation set was also supplemented with the CDA technique, a modest improvement was noticed for CNN.

Objective 5: To analyse the generalisability of the designed NFR classification system over the selected NFRs.

The corpus was separated into two sets to measure classification performance: a training set and a validation set. All models were trained with varied settings, word embeddings, and hyperparameter tunings on the training set before being tested on the other set. The accuracy and loss for both the training and validation sets were used to calculate their classification performance. For each NFR class, a confusion matrix was utilised to assess the findings based on precision, recall, and F1-score.

Under the first data augmentation settings, it achieved a 96% precision overall for CNN with the CDA approach. Efficiency received the highest F1 score. The recall is affected by maintainability and usability, as these two NFR attributes had the most inaccurate predictions, while efficiency and reliability had the least errors. At the same time, portability had the highest recall value. When only the train set was augmented, the results could only fall to the 50^{th} percentile range.

8.2 Limitations of the Study

All of the experiments in this thesis contributed significantly to the work at hand; yet some of them produced unexpected findings or revealed something new. According to the researcher’s knowledge, this was the first attempt in this sector to generate a multilabel NFR corpus. Additionally, the study used a crowdsourcing tool to select the gold standard annotations. The annotation of the NFRs was based on the judgement of three annotators', and the annotation quality was determined by an inter-annotator

agreement. However, the experiment was pre-emptively suspended due to the platform's unavailability. The findings of the first iteration indicate that annotators did not adhere to standards and failed to categorise the supplied requirements using multiple labels. Therefore, it resulted with annotations based on a single label. Additionally, the agreement achieved is fair according to Cohen's kappa scale; indicating that there are chances to obtain higher agreement if the process is repeated with improved guidelines.

1. Two data augmentation techniques were used in this study; however, the class imbalance is not addressed through any of these data augmentation techniques. The CDA approach that was tested doubled the data size. However, it does not address how to balance samples for each class, which means that even after augmentation, the class with a smaller number of samples in the original data will have a smaller number of instances.
2. One of the significant drawbacks of the CDA approach is that it can't effectively supplement datasets with a small number of documents per category. One of the reasons is that it principally depends on the combinations of the records to develop new records. In contrast, data with fewer records is not expected to generate many unique varieties.

8.3 Unexpected Results

One unexpected finding of the research was that LSTM failed to perform under both experiments. Although the training dataset was imbalanced, in the experiment with data augmentation on the entire corpus, it was observed that portability had the least number of samples for training. It was surprising that portability was learned extremely well, with not a single sentence belonging to this class classified incorrectly. Another intriguing observation in the same experiment was that the LSTM network performed satisfactorily when using the CDA technique in combination with pre-trained embeddings. However, no learning behaviour was observed until the first ten epochs. For EDA, we noticed that CNN and GRU early stopped under 20 epochs.

Unfortunately, when only the train set was augmented, none of the data augmentation techniques performed well. However, some improvements were observed when the test set was also augmented separately for CNN.

8.4 Future Recommendations

The research described in this thesis has indicated several promising directions for future research. There are numerous areas where more work may be done to improve classification or expand the study to include other corpora, data augmentation, and classification models.

- **Experiment the Custom Data Augmentation Approach with Alternate Corpus**

The CDA techniques are presented in Chapter 5. It suggests a new algorithm that increased the corpus's size yet kept the domain vocabulary. Future research will find it intriguing to see if the proposed approach provides equally better results with a variable dataset length.

- **Improvements in Design of a Gold standard Corpus Annotation**

The suggested framework for corpus annotation was designed using a crowdsourcing web-based platform, which brought many challenges. Corpus annotation is an incredibly vast field of research, and it is not only a time-consuming task, but it demands high skills for both researcher and crowd annotators. However, in the context of a gold standard corpus, many things can be improved, such as 1) how guidelines and rules are defined, 2) two-way communication for improved training, 3) the use of a reliable platform for useful annotations, and 4) annotators with domain knowledge. Therefore, it is predicted that this will give another fertile area for future research.

- **Techniques to Handle Unbalanced Data**

The study produced a corpus based on representative samples of NFRs and has more examples for each class than the previous corpus in this domain. However, it still has unbalanced samples in this corpus; the proposed CDA approach does not address this issue. Therefore, future research should focus on developing a more efficient approach to solve the problem of uneven data.

- **Classification of Software Requirement through Multi-label Classification**

The methodologies given in this thesis for examining classification methods for NFRs were based on multiclass classification. However, the researcher views software needsas a multi-label problem, which has yet to be addressed in multilabel classification. Although this study started with creating a multi-label corpus, it was not successfully completed. Therefore, due to the complexity of the nature of this problem, it has not been addressed in this study. Consequently, it is suggested that future work is required to address this issue comprehensively. It would be fascinating to discover the effect of deep learning models for the classification of FRs and multiple NFRs when the training is performed with the multi-label corpus.

References

Abad, Z. S. H., Karras, O., Ghazi, P., Glinz, M., Ruhe, G. and Schneider, K. (2017) What Works Better? A Study of Classifying Requirements, ArXiv:1707.02358 [Cs]. Available from: http://arxiv.org/abs/1707.02358 [Accessed 28 September 2020].

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A. and Arshad, H. (2018) State-of-the-art in artificial neural network applications: A survey, Heliyon, 4 (11), pp. e00938. DOI: 10.1016/j.heliyon. 2018.e00938.

Abulaish, M. and Sah, A. K. (2019) A Text Data Augmentation Approach for Improving the Performance of CNN, in: 2019 11th International Conference on Communication Systems & Networks (COMSNETS). Bengaluru, India: IEEE, pp. 625–630.

Afrin, T. and Litman, D. (2018) Annotation and Classification of Sentence-level Revision Improvement, in: Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications. New Orleans, Louisiana: Association for Computational Linguistics, pp. 240–246.

Agarwal, S. and Yu, H. (2009) Automatically classifying sentences in full-text biomedical articles into Introduction, Methods, Results and Discussion, Bioinformatics, 25 (23), pp. 3174–3180. DOI:10.1093/bioinformatics/btp548.

Aghaebrahimian, A. & Cieliebak, M. (2019), Hyperparameter tuning for deep learning in natural language processing, CEUR Workshop Proceedings. https://doi.org/10.21256/zhaw-18993

Andonie, R. (2019) Hyperparameter optimisation in learning systems, Journal of Membrane Computing, 1 (4), pp. 279–291. DOI:10.1007/s41965-019-00023-0.

Badave, M., Casamayor, A., Godoy, D., Campo, M., Chung, L., Cleland-Huang, J., et al. (2015) Classification of Non-Functional Requirements Using Semantic-FSKNN

Based ISO/IEC 9126, TELKOMNIKA (Telecommunication Computing Electronics and Control), 13 (4), pp. 1456. DOI:10.12928/telkomnika. v13i4.2300.

Baharudin, B., Lee, L. H. and Khan, K. (2010) A Review of Machine Learning Algorithms for Text-Documents Classification, Journal of Advances in Information Technology, 1 (1), pp. 4–20. DOI:10.4304/jait.1.1.4-20.

Baker, B., Gupta, O., Raskar, R. and Naik, N. (2017) Accelerating Neural Architecture Search using Performance Prediction, ArXiv:1705.10823 [Cs]. Available from: http://arxiv.org/abs/1705.10823 [Accessed 10 November 2020].

Baker, C., Deng, L., Chakraborty, S. and Dehlinger, J. (2019) Automatic Multi-class Non-Functional Software Requirements Classification Using Neural Networks, in: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). Milwaukee, WI, USA: IEEE, pp. 610–615.

Balaprakash, P., Salim, M., Uram, T. D., Vishwanath, V. and Wild, S. M. (2018) DeepHyper: Asynchronous Hyperparameter Search for Deep Neural Networks, in: 2018 IEEE 25th International Conference on High Performance Computing (HiPC). Bengaluru, India: IEEE, pp. 42–51.

Baudat, G. and Anouar, F. (2000) Generalized Discriminant Analysis Using a Kernel Approach, Neural Computation, 12 (10), pp. 2385–2404. DOI:10.1162/089976600300014980.

Bengio, Y., Ducharme, R., Vincent, P. and Jauvin, C (2003) A Neural Probabilistic Language Model, pp. 19.

Biber, D. (1993) Representativeness in Corpus Design, pp. 31.

Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T. (2017) Enriching Word Vectors with Subword Information, ArXiv:1607.04606 [Cs]. Available from: http://arxiv.org/abs/1607.04606 [Accessed 10 November 2020].

Breck, E., Polyzotis, N., Roy, S., Whang, S. E. and Zinkevich, M. [no date] Data Validation for Machine Learning, pp. 14.

Cagli, E., Dumas, C. and Prouff, E. (2017) Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures, in: Fischer, W. and Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. Cham: Springer International Publishing,10529, pp. 45–68.

Casamayor, A., Godoy, D. and Campo, M. (2010) Identification of non-functional requirements in textual specifications: A semi-supervised learning approach, Information and Software Technology, 52 (4), pp. 436–445. DOI: 10.1016/j.infsof.2009.10.010.

Caselles-Dupré, H., Lesaint, F. and Royo-Letelier, J. (2018) Word2Vec applied to Recommendation: Hyperparameters Matter, ArXiv:1804.04212 [Cs, Stat]. Available from: http://arxiv.org/abs/1804.04212 [Accessed 10 November 2020].

Challita, N., Khalil, M. and Beauseroy, P. (2016) New feature selection method based on neural network and machine learning, 2016 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), pp. 81–85. DOI:10.1109/IMCET.2016.7777431.

Chan, S., Treleaven, P. and Capra, L. (2013) Continuous hyperparameter optimisation for large-scale recommender systems, in: 2013 IEEE International Conference on Big Data. Silicon Valley, CA, USA: IEEE, pp. 350–358.

Chen, G., Ye, D., Xing, Z., Chen, J. and Cambria, E. (2017) Ensemble application of convolutional and recurrent neural networks for multi-label text categorization, in: 2017 International Joint Conference on Neural Networks (IJCNN). Anchorage, AK, USA: IEEE, pp. 2377–2383.

Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., et al. (2014) DaDianNao: A Machine-Learning Supercomputer. DOI:10.1109/MICRO.2014.58.

Cheng, H. G. and Phillips, M. R. (2014) Secondary analysis of existing data: opportunities and implementation, 26 (6), pp. 5.

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014) Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, ArXiv:1412.3555 [Cs]. Available from: http://arxiv.org/abs/1412.3555 [Accessed 10 November 2020].

Chung, L., Nixon, B. A., Yu, E. and Mylopoulos, J. (2000) Non-Functional Requirements in Software Engineering. Boston, MA: Springer US. DOI:10.1007/978-1-4615-5269-7.

Cleland-Huang, J., Settimi, R., Xuchang Zou and Solc, P. (2006) The Detection and Classification of Non-Functional Requirements with Application to Early Aspects, in: 14th IEEE International Requirements Engineering Conference (RE'06). Minneapolis/St. Paul, MN: IEEE, pp. 39–48.

Cleland-Huang, J., Settimi, R., Zou, X. and Solc, P. (2007) Automated classification of non-functional requirements, Requirements Engineering, 12 (2), pp. 103–120. DOI:10.1007/s00766-007-0045-1.

Cogswell, M., Ahmed, F., Girshick, R., Zitnick, L. and Batra, D. (2016) Reducing Overfitting in Deep Networks by Decorrelating Representations, pp. 13.

Collobert, R. and Weston, J. (2008) A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning, pp. 8. https://doi.org/10.1145/1390156.1390177

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. and Kuksa, P. (2011) Natural Language Processing (Almost) from Scratch, NATURAL LANGUAGE PROCESSING, pp. 45.

Coulombe, C. (2018) Text Data Augmentation Made Simple by Leveraging NLP Cloud APIs, pp. 33.

Dhurandhar, A. and Dobra, A. (2008) Probabilistic Characterization of Random Decision Trees, 9 (10), pp. 28.

Dipper, S., Götze, M., & Skopeteas, S., (2004) Towards user-adaptive annotation guidelines. Presented at the COLING Workshop on Linguistically Interpreted Corpora (LINC-2004), Geneva, Switzerland.

Santos, C. N. dos, Xiang, B. and Zhou, B. (2015) Classifying Relations by Ranking with Convolutional Neural Networks, ArXiv:1504.06580 [Cs]. Available from: http://arxiv.org/abs/1504.06580 [Accessed 18 November 2020].

Elman, J. L. (1993) Learning and development in neural networks: the importance of starting small, Cognition, 48 (1), pp. 71–99. DOI:10.1016/0010-0277(93)90058-4.

Feng, P.-M., Ding, H., Chen, W. and Lin, H. (2013) Naïve Bayes Classifier with Feature Selection to Identify Phage Virion Proteins, Computational and Mathematical Methods in Medicine, 2013, pp. 1–6. DOI:10.1155/2013/530696.

Fukushima, K. (1980) Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biological Cybernetics, 36 (4), pp. 193–202. DOI:10.1007/BF00344251.

Fukushima, K. and Miyake, S. (1982) Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position, Pattern Recognition, 15 (6), pp. 455–469. DOI:10.1016/0031-3203(82)90024-3.

Gao, B. and Pavel, L. (2018) On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning, ArXiv:1704.00805 [Cs, Math]. Available from: http://arxiv.org/abs/1704.00805 [Accessed 18 November 2020].

Golik, P., Doetsch, P. and Ney, H. (2013) Cross-Entropy vs. Squared Error Training: A Theoretical and Experimental Comparison, pp. 6.

Griffiths, T. L., Steyvers, M. and Tenenbaum, J. B. (2007) Topics in semantic representation., Psychological Review, 114 (2), pp. 211–244. DOI:10.1037/0033-295X.114.2.211.

Gupta, P. and Schütze, H. (2018) LISA: Explaining Recurrent Neural Network Judgments via Layer-wIse Semantic Accumulation and Example to Pattern Transformation, in: Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP. Brussels, Belgium: Association for Computational Linguistics, pp. 154–164.

Gupta, P., Yaseen, U. and Schütze, H. (2019) Linguistically Informed Relation Extraction and Neural Architectures for Nested Named Entity Recognition in BioNLP-OST 2019, in: Proceedings of the 5th Workshop on BioNLP Open Shared Tasks. Hong Kong, China: Association for Computational Linguistics, pp. 132–142.

Gupta, T. K. and Raza, K. (2018) Optimising Deep Neural Network Architecture: A Tabu Search Based Approach, pp. 15.

Hadji, I. and Wildes, R. P. (2018) A New Large Scale Dynamic Texture Dataset with Application to ConvNet Understanding, in: Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y. (eds.) Computer Vision – ECCV 2018. Cham: Springer International Publishing,11218, pp. 334–351.

Haury, A. C., Gestraud, P. and Vert, J. P. (2011) The influence of feature selection methods on accuracy, stability, and interpretability of molecular signatures, PLoS ONE, 6 (12), pp. 1–12. DOI: 10.1371/journal.pone.0028210.

Hernández-García, A. and König, P. (2018) Do deep nets really need weight decay and dropout? ArXiv:1802.07042 [Cs]. Available from: http://arxiv.org/abs/1802.07042 [Accessed 10 November 2020].

Hochreiter, S. and Schmidhuber, J. (1997) Long Short-Term Memory, Neural Computation, 9 (8), pp. 1735–1780. DOI:10.1162/neco.1997.9.8.1735.

Hovy, E. and Lavid, J. (2010) Towards a ‘Science’ of Corpus Annotation: A New Methodological Challenge for Corpus Linguistics, pp. 26.

Hu, B., Tu, Z., Lu, Z., Li, H. and Chen, Q. (2015) Context-Dependent Translation Selection Using Convolutional Neural Network, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers). Beijing, China: Association for Computational Linguistics, pp. 536–541.

Hussain, I., Kosseim, L. and Ormandjieva, O. (2008) Using Linguistic Knowledge to Classify Non-functional Requirements in SRS documents, in: Kapetanios, E., Sugumaran, V., and Spiliopoulou, M. (eds.) Natural Language and Information Systems. Berlin, Heidelberg: Springer Berlin Heidelberg,5039, pp. 287–298.

Hutter, F., Kotthoff, L. and Vanschoren, J. (eds.) (2019) Automated Machine Learning: Methods, Systems, Challenges. Cham: Springer International Publishing. DOI:10.1007/978-3-030-05318-5.

Inoue, H. (2018) Data Augmentation by Pairing Samples for Images Classification, ArXiv:1801.02929 [Cs, Stat]. Available from: http://arxiv.org/abs/1801.02929 [Accessed 17 September 2020].

Joachims, T. (2002) Optimising Search Engines using Clickthrough Data, pp. 10. DOI: https://doi.org/10.1145/775047.775067.

Jolliffe, I. T. and Cadima, J. (2016) Principal component analysis: a review and recent developments, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 374 (2065), pp. 20150202. DOI:10.1098/rsta.2015.0202.

Joo, W., Lee, W., Park, S. and Moon, I.-C. (2019) Dirichlet Variational Autoencoder, ArXiv:1901.02739 [Cs, Stat]. Available from: http://arxiv.org/abs/1901.02739 [Accessed 11 November 2020].

Kalchbrenner, N., Grefenstette, E. and Blunsom, P. (2014) A Convolutional Neural Network for Modelling Sentences, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Baltimore, Maryland: Association for Computational Linguistics, pp. 655–665.

Karlik, B. and Olgac, A. V. (2011) Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks, pp. 13.

Kamal, N., Andrew, M. and Tom, M. (2006) Semi-Supervised Text Classification Using EM, in: Chapelle, O., Scholkopf, B., and Zien, A. (eds.) Semi-Supervised Learning. The MIT Press, pp. 32–55.

Kang, N., Singh, B., Afzal, Z., van Mulligen, E. M. and Kors, J. A. (2013) Using rule-based natural language processing to improve disease normalization in biomedical text, Journal of the American Medical Informatics Association, 20 (5), pp. 876–881. DOI:10.1136/amiajnl-2012-001173.

Khalid, S. and Nasreen, S. (2014) A Survey of Feature Selection and Feature Extraction Techniques in Machine Learning, (October 2016). DOI:10.1109/SAI.2014.6918213.

Kilimci, Z. H. and Akyokus, S. (2018) Deep Learning- and Word Embedding-Based Heterogeneous Classifier Ensembles for Text Classification, Complexity, 2018, pp. 1–10. DOI:10.1155/2018/7130146.

Kim, Y. (2014) 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, pp. 1746–1751.

Kingma, D. P. and Ba, J. (2017) Adam: A Method for Stochastic Optimisation, ArXiv:1412.6980 [Cs]. Available from: http://arxiv.org/abs/1412.6980 [Accessed 12 November 2020].

Kobayashi, S. (2018) Contextual Augmentation: Data Augmentation by Words with Paradigmatic Relations, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers). New Orleans, Louisiana: Association for Computational Linguistics, pp. 452–457.

Krizhevsky, A. (2014) One weird trick for parallelizing convolutional neural networks, ArXiv:1404.5997 [Cs]. Available from: http://arxiv.org/abs/1404.5997 [Accessed 11 October 2020].

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2017) ImageNet classification with deep convolutional neural networks, Communications of the ACM, 60 (6), pp. 84–90. DOI:10.1145/3065386.

Krogh, A. and Hertz, J. A. (1992) A Simple Weight Decay Can Improve Generalization, pp. 8.

Kukačka, J., Golkov, V. and Cremers, D. (2017) Regularization for Deep Learning: A Taxonomy, ArXiv:1710.10686 [Cs, Stat]. Available from: http://arxiv.org/abs/1710.10686 [Accessed 10 November 2020].

kumar, Y. and Sahoo, G. (2012) Analysis of Parametric & Non-Parametric Classifiers for Classification Technique using WEKA, International Journal of Information Technology and Computer Science, 4 (7), pp. 43–49. DOI:10.5815/ijitcs.2012.07.06.

Kurtanovic, Z. and Maalej, W. (2017) Automatically Classifying Functional and Non-Functional Requirements Using Supervised Machine Learning, pp. 6.

Lecun, Y. (1998) Gradient-Based Learning Applied to Document Recognition, PROCEEDINGS OF THE IEEE, 86 (11), pp. 47.

LeCun, Y., Bengio, Y. and Hinton, G. (2015) Deep learning, Nature, 521 (7553), pp. 436–444. DOI:10.1038/nature14539.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D. (1989) Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1 (4), pp. 541–551. DOI:10.1162/neco.1989.1.4.541.

Levy, O. and Goldberg, Y. (2014) Dependency-Based Word Embeddings, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Baltimore, Maryland: Association for Computational Linguistics, pp. 302–308.

Lewis, D. D. (1998) Naive (Bayes) at forty: The independence assumption in information retrieval, in: Nédellec, C. and Rouveirol, C. (eds.) Machine Learning: ECML-98. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 4–15.

Liu, Y. and Liao, S. (2014) Preventing Over-Fitting of Cross-Validation with Kernel Stability, in: Calders, T., Esposito, F., Hüllermeier, E., and Meo, R. (eds.) Machine Learning and Knowledge Discovery in Databases. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 290–305.

Lu, M. and Liang, P. (2017) Automatic Classification of Non-Functional Requirements from Augmented App User Reviews, in: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering - EASE'17. Karlskrona, Sweden: ACM Press, pp. 344–353.

Luo, G. (2016) A review of automatic selection methods for machine learning algorithms and hyper-parameter values, Network Modeling Analysis in Health Informatics and Bioinformatics, 5 (1), pp. 18. DOI:10.1007/s13721-016-0125-6.

Ma, X. and Hovy, E. (2016) End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF, ArXiv:1603.01354 [Cs, Stat]. Available from: http://arxiv.org/abs/1603.01354 [Accessed 10 November 2020].

Mahmoud, A. and Williams, G. (2016) Detecting, classifying, and tracing non-functional software requirements, Requirements Engineering, 21 (3), pp. 357–381. DOI:10.1007/s00766-016-0252-8.

Mahmoud, M. (2017) Software Requirements Classification using Natural Language Processing and SVD, International Journal of Computer Applications, 164 (1), pp. 7–12. DOI:10.5120/ijca2017913555.

Mairiza, D., Zowghi, D. and Nurmuliani, N. (2009) Managing Conflicts among Non-Functional Requirements, pp. 10.

Mannor, S., Peleg, D. and Rubinstein, R. (2005) The cross-entropy method for classification, in: Proceedings of the 22nd international conference on Machine learning - ICML '05. Bonn, Germany: ACM Press, pp. 561–568.

Maron, M. E. (1961) Automatic Indexing: An Experimental Inquiry, Journal of the ACM, 8 (3), pp. 404–417. DOI:10.1145/321075.321084.

Mcculloch, W. S. And Pitts, W. (1990) A Logical Calculus of The Ideas Immanent in Nervous Activity, 52 (1–2), pp. 17.

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013) Efficient Estimation of Word Representations in Vector Space, ArXiv:1301.3781 [Cs]. Available from: http://arxiv.org/abs/1301.3781 [Accessed 10 November 2020].

Mikolov, T., Joulin, A., Chopra, S., Mathieu, M. and Ranzato, M. (2015) Learning Longer Memory in Recurrent Neural Networks, ArXiv:1412.7753 [Cs]. Available from: http://arxiv.org/abs/1412.7753 [Accessed 10 November 2020].

Mikolov, T., Karafiat, M., Burget, L., Cernocky, J. and Khudanpur, S. (2010) Recurrent Neural Network Based Language Model, pp. 4.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. (2013) Distributed Representations of Words and Phrases and their Compositionality, pp. 9.

Minar, M. R. and Naher, J. (2018) Recent Advances in Deep Learning: An Overview, ArXiv:1807.08169 [Cs, Stat]. DOI:10.13140/RG.2.2.24831.10403.

Mirończuk, M. M. and Protasiewicz, J. (2018) A recent overview of the state-of-the-art elements of text classification, Expert Systems with Applications, 106, pp. 36–54. DOI: 10.1016/j.eswa.2018.03.058.

Naili, M., Chaibi, A. H. and Ben Ghezala, H. H. (2017) Comparative study of word embedding methods in topic segmentation, Procedia Computer Science, 112, pp. 340–349. DOI: 10.1016/j.procs.2017.08.009.

Nakamura, K. and Hong, B.-W. (2019) Adaptive Weight Decay for Deep Neural Networks, ArXiv:1907.08931 [Cs, Stat]. Available from: http://arxiv.org/abs/1907.08931 [Accessed 12 November 2020].

Nair, V. and Hinton, G. E. (2010) Rectified Linear Units Improve Restricted Boltzmann Machines, pp. 8.

Nematzadeh, A., Meylan, S. C. and Griffiths, T. L. (2017) Evaluating Vector-Space Models of Word Representation, or The Unreasonable Effectiveness of Counting Words Near Other Words, pp. 6.

Nusrat, I. and Jang, S.-B. (2018) A Comparison of Regularization Techniques in Deep Neural Networks, Symmetry, 10 (11), pp. 648. DOI:10.3390/sym10110648

Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S. (2018) Activation Functions: Comparison of trends in Practice and Research for Deep Learning, ArXiv:1811.03378 [Cs]. Available from: http://arxiv.org/abs/1811.03378 [Accessed 17 January 2022].

Ormandjieva, O. (2013) Ontology-Based Classification of Non-Functional Requirements in Software Specifications: A new Corpus and SVM-Based Classifier, (ii). DOI:10.1109/COMPSAC.2013.64.

Perera, P. and Patel, V. M. (2019) Deep Transfer Learning for Multiple Class Novelty Detection, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Long Beach, CA, USA: IEEE, pp. 11536–11544.

Perin, G. and Picek, S. (2021) On the Influence of Optimisers in Deep Learning-based Side-channel Analysis, pp. 22.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. and Zettlemoyer, L. (2018) Deep contextualized word representations, ArXiv:1802.05365 [Cs]. Available from: http://arxiv.org/abs/1802.05365 [Accessed 10 November 2020].

Peters, M., Neumann, M., Zettlemoyer, L. and Yih, W. (2018) Dissecting Contextual Word Embeddings: Architecture and Representation, pp. 11.

Pohl, K. and Rupp, C. (2015) Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam, foundation level, IREB compliant. Second edition. Santa Barbara, CA: Rocky Nook.

Poria, S., Cambria, E., Hazarika, D., Majumder, N., Zadeh, A. and Morency, L.-P. (2017) Context-Dependent Sentiment Analysis in User-Generated Videos, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vancouver, Canada: Association for Computational Linguistics, pp. 873–883.

Porter, N. D., Verdery, A. M. and Gaddis, S. M. (2020) Enhancing big data in the social sciences with crowdsourcing: Data augmentation practices, techniques, and opportunities, PLOS ONE, 15 (6), pp. e0233154. DOI: 10.1371/journal.pone.0233154.

Prechelt, L. (1998) Early Stopping | but when? pp. 15.

Pustejovsky, J. and Stubbs, A. (2013) Natural language annotation for machine learning. Sebastopol, CA: O'Reilly Media.

Quijas, J. (2017) Analysing the Effects of Data Augmentation and Free Parameters for Text Classification with Recurrent Convolutional Neural Networks, pp. 54.

Ramadhani, D. A., Rochimah, S. and Yuhana, U. L. (2015) Classification of Non-Functional Requirements Using Semantic-FSKNN Based ISO/IEC 9126,

TELKOMNIKA (Telecommunication Computing Electronics and Control), 13 (4), pp. 1456. DOI:10.12928/telkomnika. v13i4.2300.

Rashwan, A., Ormandjieva, O. and Witte, R. (2013) 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. Kyoto, Japan: IEEE, pp. 381–386.

Roman (1985) A taxonomy of current issues in requirements engineering, Computer, 18 (4), pp. 14–23. DOI:10.1109/MC.1985.1662861.

Rosario, R. R. (2017) A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Statistics, pp. 210.

Rumelhart, D. E. and McClelland, J. L. (1986) Parallel distributed processing: explorations in the microstructure of cognition. Cambridge, Mass: MIT Press.

Sabir, M., Chrysoulas, C. and Banissi, E. (2020) Multi-label Classifier to Deal with Misclassification in Non-functional Requirements, in: Rocha, Á., Adeli, H., Reis, L. P., Costanzo, S., Orovic, I., and Moreira, F. (eds.) Trends and Innovations in Information Systems and Technologies. Cham: Springer International Publishing, pp. 486–493.

Sharma, V. S., Ramnani, R. R. and Sengupta, S. (2014) A framework for identifying and analyzing non-functional requirements from text, in: Proceedings of the 4th International Workshop on Twin Peaks of Requirements and Architecture - TwinPeaks 2014. Hyderabad, India: ACM Press, pp. 1–8.

Silberztein, M. (2020) Using Linguistic Resources to Evaluate the Quality of Annotated Corpora, pp. 11.

Singh, P., Singh, D. and Sharma, A. (2016) 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). Jaipur, India: IEEE, pp. 620–626.

Slankas, J. and Williams, L. (2013) Automated extraction of non-functional requirements in available documentation, 2013 1st International Workshop on Natural Language Analysis in Software Engineering, NaturaLiSE 2013 - Proceedings, pp. 9–16. DOI:10.1109/NAturaLiSE.2013.6611715.

Sommerville, I. (2007) Software engineering. 8th ed. Harlow, England; New York: Addison-Wesley.

Song, H., Kim, M., Park, D. and Lee, J.-G. (2020) How does Early Stopping Help Generalization against Label Noise? ArXiv:1911.08059 [Cs, Stat]. Available from: http://arxiv.org/abs/1911.08059 [Accessed 11 November 2020].

Sorzano, C. O. S., Vargas, J. and Pascual, A. (2014) A survey of dimensionality reduction techniques, pp. 35.

Sun, D., Zhao, S., Zhang, Z. and Shi, X. (2017) A match method Based on Latent Semantic Analysis for Earthquake hazard Emergency Plan, Isprs - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XLII-2/W7, pp. 137–141. DOI:10.5194/isprs-archives-XLII-2-W7-137-2017.

Sunner, D. and Bajaj, H. (2016) Classification of Functional and Non-functional Requirements in Agile by Cluster Neuro-Genetic Approach, International Journal of Software Engineering, and Its Applications, 10 (10), pp. 129–138. DOI:10.14257/ijseia.2016.10.10.13.

Sutskever, I., Vinyals, O. and Le, Q. V. (2014) Sequence to Sequence Learning with Neural Networks, ArXiv:1409.3215 [Cs]. Available from: http://arxiv.org/abs/1409.3215 [Accessed 18 November 2020].

Sze, V., Chen, Y.-H., Yang, T.-J. and Emer, J. (2017) Efficient Processing of Deep Neural Networks: A Tutorial and Survey, ArXiv:1703.09039 [Cs]. Available from: http://arxiv.org/abs/1703.09039 [Accessed 25 September 2020].

Tabassum, M. R., Md. Saeed Siddik, Shoyaib, M. and Khaled, S. M. (2014) Determining interdependency among non-functional requirements to reduce conflict, in: 2014 International Conference on Informatics, Electronics & Vision (ICIEV). Dhaka, Bangladesh: IEEE, pp. 1–6.

Takahashi, R., Matsubara, T. and Uehara, K. (2020) Data Augmentation using Random Image Cropping and Patching for Deep CNNs, IEEE Transactions on Circuits and Systems for Video Technology, 30 (9), pp. 2917–2931. DOI:10.1109/TCSVT.2019.2935128.

Tang, D., Qin, B. and Liu, T. (2015) Document Modeling with Gated Recurrent Neural Network for Sentiment Classification, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. Lisbon, Portugal: Association for Computational Linguistics, pp. 1422–1432.

Taylor, L. and Nitschke, G. (2018) Improving Deep Learning with Generic Data Augmentation, in: 2018 IEEE Symposium Series on Computational Intelligence (SSCI). Bangalore, India: IEEE, pp. 1542–1547.

Tissier, J., Gravier, C. and Habrard, A. (2017) Dict2vec: Learning Word Embeddings using Lexical Dictionaries, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. Copenhagen, Denmark: Association for Computational Linguistics, pp. 254–263.

Tomanek, K., Wermter, J. and Hahn, U. (2007) An Approach to Text Corpus Construction Which Cuts Annotation Costs and Maintains Reusability of Annotated Data, pp. 10.

Tomar, D. and Agarwal, S. (2014) A Survey on Pre-processing and Post-processing Techniques in Data Mining, International Journal of Database Theory and Application, 7 (4), pp. 99–128. DOI:10.14257/ijdta.2014.7.4.09.

Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K.-L. A., Elkhatib, Y., Hussain, A. and Al-Fuqaha, A. (2017) Unsupervised Machine Learning for Networking: Techniques,

Applications and Research Challenges, ArXiv:1709.06599 [Cs]. Available from: http://arxiv.org/abs/1709.06599 [Accessed 5 August 2019].

Vapnik, V. N. and Lerner, A. Y. (1963) Recognition of Patterns with help of Generalized Portraits, pp. 8.

Varghese, N. (2012) A Survey of Dimensionality Reduction and Classification Methods, International Journal of Computer Science & Engineering Survey, 3 (3), pp. 45–54. DOI:10.5121/ijcses.2012.3304.

Vlas, R. E. and Robinson, W. N. (2012) Two Rule-Based Natural Language Strategies for Requirements Discovery and Classification in Open-Source Software Development Projects, Journal of Management Information Systems, 28 (4), pp. 11–38. DOI:10.2753/MIS0742-1222280402.

Vu, N. T., Adel, H., Gupta, P. and Schütze, H. (2016) Combining Recurrent and Convolutional Neural Networks for Relation Classification, in: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. San Diego, California: Association for Computational Linguistics, pp. 534–539.

Walowe Mwadulo, M. (2016) A Review on Feature Selection Methods for Classification Tasks, International Journal of Computer Applications Technology and Research, 5 (6), pp. 395–402. DOI:10.7753/IJCATR0506.1013.

Wang, D., He, H. and Liu, D. (2018) Intelligent Optimal Control with Critic Learning for a Nonlinear Overhead Crane System, IEEE Transactions on Industrial Informatics, 14 (7), pp. 2932–2940. DOI:10.1109/TII.2017.2771256.

Wang, J.-H., Liu, T.-W., Luo, X. and Wang, L. (2018) An LSTM Approach to Short Text Sentiment Classification with Word Embeddings, pp. 10.

Wei, J. and Zou, K. (2019) EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International

Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics, pp. 6381–6387.

Wiegers, K. E. and Beatty, J. (2013) Software requirements. Third edition. Redmond, Washington: Microsoft Press, s division of Microsoft Corporation.

Wolfe, C. R. and Lundgaard, K. T. (2020) E-Stitchup: Data Augmentation for Pre-Trained Embeddings, ArXiv:1912.00772 [Cs, Stat]. Available from: http://arxiv.org/abs/1912.00772 [Accessed 10 November 2020].

Wong, S. C., Gatt, A., Stamatescu, V. and McDonnell, M. D. (2016) Understanding data augmentation for classification: when to warp? ArXiv:1609.08764 [Cs]. Available from: http://arxiv.org/abs/1609.08764 [Accessed 11 November 2020].

Wu, J., Gupta, S. and Bajaj, C. (2016) Higher Order Mutual Information Approximation for Feature Selection, ArXiv:1612.00554 [Cs]. Available from: http://arxiv.org/abs/1612.00554 [Accessed 18 November 2020].

Xu, X., Zheng, J., Yang, J., Xu, D. and Chen, Y. (2017) Data classification using evidence reasoning rule, Knowledge-Based Systems, 116, pp. 144–151. DOI: 10.1016/j.knosys.2016.11.001.

Younas, M., Wakil, K., N., D., Arif, M. and Mustafa, A. (2019) An Automated Approach for Identification of Non-Functional Requirements using Word2Vec Model, International Journal of Advanced Computer Science and Applications, 10 (8). DOI:10.14569/IJACSA.2019.0100871.

Young, T., Hazarika, D., Poria, S. and Cambria, E. (2017) Recent Trends in Deep Learning Based Natural Language Processing, ArXiv:1708.02709 [Cs]. Available from: http://arxiv.org/abs/1708.02709 [Accessed 17 February 2019].

Zeman, D., Popel, M., Straka, M., Hajic, J., Nivre, J., Ginter, F., et al. (2017) CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, in: Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw

Text to Universal Dependencies. Vancouver, Canada: Association for Computational Linguistics, pp. 1–19.

Zena M. Hira and Gillies, D. F. (2015) A Review of Feature Selection and Feature Extraction Methods Applied on Microarray Data, Advances in Bioinformatics, 2015 (1). DOI:10.1155/2015/198363.

Zeng, D., Liu, K., Lai, S., Zhou, G. and Zhao, J. (2014) Relation Classification via Convolutional Deep Neural Network, pp. 10.

Zeiler, M. D., Ranzato, M., Monga, R., Mao, M., Yang, K., Le, Q. V., et al. (2013) On rectified linear units for speech processing, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. Vancouver, BC, Canada: IEEE, pp. 3517–3521.

Zhang, W., Yang, Y., Wang, Q. and Shu, F. (2011) An empirical study on classification of non-functional requirements, … of the 23rd International Conference on …, (January).

Zhang, X. and Lapata, M. (2014) Chinese Poetry Generation with Recurrent Neural Networks, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, pp. 670–680.

Zhang, X., Zhao, J. and LeCun, Y. (2016) Character-level Convolutional Networks for Text Classification, ArXiv:1509.01626 [Cs]. Available from: http://arxiv.org/abs/1509.01626 [Accessed 16 September 2020].

Zhu, Q., He, Z., Zhang, T. and Cui, W. (2020) Improving Classification Performance of Softmax Loss Function Based on Scalable Batch-Normalization, Applied Sciences, 10 (8), pp. 2950. DOI:10.3390/app10082950.

Appendix- A: Comparison of Existing Methods for Requirements Classification

Table A- 1: Comparison of Existing Methods for Requirements Classification

Selected Feature	Algorithm	NFR attributes	Dataset	Tool	Study
Keywords, **Indicator term**	SIG (soft goal interdependency graph)	1FR 9 NFR attributes	PROMISE corpus, Integrated engineering toolset (IET) under development at Siemens Logistics and Automation plant	Un-known	(Cleland et al., 2007)
Five Syntactic features, **Nine POS based features,** **keyword features,** **Smoothed/Unsmoothed Probability Measure, (SPM)/ (UPM)**	Decision tree	PROMISE corpus	15 SRS problem statements, all from different domains, with a total of 765 sentences: 495 (65%) of them were annotated as "NFR", while 270 (35%) of them as "FR".	Stanford Parser (equipped with Brill's POS tagger, Morphological stemmer)	(Hussain et al., 2008)
Users feedback Expectation maximisation, **TF/IDF**	Naïve Bayesian KNN, Expectation maximisation with naïve bayes/tfidf	1FR 11 NFR attributes	PROMISE corpus	Porter algorithm for stemming	(Casamayor et al., 2010)
Ngram (one word), **Multiword expression,** **Information gain (IG)**	SVM	12 NFR attribute	PROMISE[1] corpus	Un-known	(Zhang et al., 2011)

Multilevel ontology-based design **POS Tagging, SAO approach** **Delimiter based approach**	Rule base (RCNL) Classifier	Binary classification	Dataset from Source forge	GATE JAPE	(Vlas and Robinson, 2012)
Ontology-based	SVM+OWL ontology	7 NFR attributes	Concordia corpus, 3064 sentences	GATE	(Rashwan et al., 2013)
Distance function	KNN classifier Multinomial-naïve Bayes Sequential minimal optimisation (SMO)	9 NFR attributes in addition reliability, recoverability. performance and scalability as a single category 78access control and audit measured as separate entity from security	11 EHR document PROMISE corpus CCHIT Ambulatory Requirements iTrust	Weka	(Slankas and Williams, 2013)
Syntactic and Semantic pattern,	Fully rule based.	6 NFR attributes extracted and redefined from PROMISE categories	PROMISE Corpus[1]	Java JRE 1.6, wordnet for lemma generation	(Sharma et al., 2014)

Semantic factors, K distance function from 10 to 55	KNN classifier Nearest neighbours	Multilabel NFR attributes	PROMISE corpus Itrust, CCHIT, World Vista US Veterans Health Care System Documentation, Online Project Marking System SRS, Mars Express Aspera-3 Processing and Archiving Facility SRS.	Un-known	(Ramadhani et al., 2015)
Thematic role	Rule based system for annotation	1FRl+11 NFR attributes	PROMISE and Concordia	GATE, JAPE rules, ANNIE tokenizer, POS tagger, chunker, snowball stemmer, , Multilingual Noun Phrase Extractor (MuNPEx) GATE Morphological Analyzer, Number Tagger (for tagging numbers), Measurement Tagger	(Singh et al., 2016)
Tokenisation, Stop-word removal, stemming, TF-IDF, RBF kernel with SVM, UPGMA with K-means	K-means, SVM, Neural network with genetic algorithm	Unknown	PROMISE, EU Procurement online system	Python, MATLAB	(Sunner and Bajaj, 2016)

Semantics of FR to identify Quality concerns, LSI, Vector space model	Unsupervised approach Hierarchical clustering algorithm	security, performance, accessibility, accuracy, portability, safety, legal, privacy, reliability, availability, and interoperability	SafeDrink, SmartTrip and BlueWallet	Java API WordNet, Open NLP lemmatizer Porter stemmer WordNet	(Mahmoud and Williams, 2016)
POS Taggin, Ngram, uni, bi, and trigram	Decision Tree	1FR 11 NFR attributes	PROMISE	Un-known	(Kurtanovic and Maalej, 2017)
POS, Temporal Tagging, Entity Tagging	Naïve Bayes LDA, BTM Hierarchical, K-Means, Hybrid	1FR 11 NFR attributes	Tera-PROMISE	Java, Weka	(Abad et al., 2017)
Five representation models TF, TF-IDF, TFIDF-CF, Bigram and Trigram, LSA with cosine distance	SVD model with NLP	FR, A=Availability, L = Legal, LF = Look and feel, MN = Maintainability, O = Operational, PE = Performance, SC = Scalability, SE = Security, US = Usability,	Used tera PROMISE dataset which includes, PROMISE, Itrust, CCHIT, World Vista US Veterans Health Care System Documentation, Online Project Marking System SRS, Mars Express Aspera-3 Processing and Archiving Facility SRS.	Eclipse for Java and Java SE Development Kit 7u79	(Mahmoud, 2017)

		FT = Fault tolerance, and PO = Portability			
Keywords, Similarity distance	Word2vec	1FR 11 NFR attributes	PROMISE corpus	Unknown	(Younas et al., 2019)
Unsupervised Learning	ANN, CNN	Maintainability, operability, performance, security, and usability	PROMISE corpus	Unknown	(Baker et al., 2019)

Appendix- B: Corpus Design

Table A- 2: Steps to Design a Gold Standard Corpus

Input: Data, Guidelines **Output**: Gold standard Annotated Corpus

1 Initialize r where D∈ dataset

2 Define guideline and rules for participants a, b, c ∈n, worker ids

3 Create test as 't' for participants, if a, b, c ∈ n pass test 't' then move to step4 else exit.

4 Select a, b, c ∈n

5 for each r ∈ D do

6 For each label l_i ∈L do

7 U= set of annotators who have assigned l_i label to r.

8 W= set of annotators who have not assigned l_i for r.

9 If cardinality(U)>cardinality(W) then

10 Assign l_i to r

11 J=J+1 where J∈U

12 End

13 Else if cardinality(U)< cardinality(W)

14 Then

15 Do not assign l_i to r.

16 J=J+1 where j ∈W

17 End

18 Else if then

19 Assign label l_i to r

20 End

21 ENDS

End

D represents the dataset, whereas *r* denotes the requirements belonging to *D*. *A, b,* and *c* are the annotators, whereas '*t'* denotes the test question. If a participant passes the test *'t'*, select it as annotator and grant it access to the complete annotation task. The *U* is the annotator who assigns the label *'l_i'* to a requirement *'r.'* (where li is the label or pair of labels). If two out of three annotators assign a label 'l_i' to a requirement *'r'* then label it as l_i otherwise, do not consider it as true.

Appendix- C: The Custom Augmentation Approach

Table A- 3: The Custom Augmentation Approach

'A' Sorted set (Ascending order)		**Augmented subset (OA_a)**	**Augmented subset (OA_aOB_d)**
1	System can handle user different request at same time	System can handle user different request at same time The system should perform every task in less than 6 seconds.	System can handle user different request at same time The system should perform every task in less than 6 seconds.
2	The system shall refresh the display every 60 seconds.	The system shall refresh the display every 60 seconds User login time must be less than 1 minutes	The system shall refresh the display every 60 seconds User login time must be less than 1 minutes
3	The system should perform every task in less than 6 seconds.	The system should perform every task in less than 6 seconds System can handle user different request at same time	The system should perform every task in less than 6 seconds System can handle user different request at same time
4	User login time must be less than 1 minutes	User login time must be less than 1 minutes The system shall refresh the display every 60 seconds	User login time must be less than 1 minutes The system shall refresh the display every 60 seconds
.	……..	……	
An	…….	OA_a	
'O' Original set		**Augmented subset (OB_d)**	
The system should perform every task in less than 6 seconds.			The system should perform every task in less than 6 seconds User login time must be less than 1 minutes.
User login time must be less than 1 minutes			User login time must be less than 1 minutes The system should perform every task in less than 6 seconds.
System can handle user different request at same time			System can handle user different request at same time The system shall refresh the display every 60 seconds
The system shall refresh the display every 60 seconds			The system shall refresh the display every 60 seconds System can handle user different request at same time
			$OAaOB_d$
On			
'B' Sorted set (Descending order)			
1	User login time must be less than 1 minutes	The system should perform every task in less than 6 seconds User login time must be less than 1 minutes.	
2	The system should perform every task in less than 6 seconds.	User login time must be less than 1 minutes The system should perform every task in less than 6 seconds.	
3	The system shall refresh the display every 60 seconds	System can handle user different request at same time The system shall refresh the display every 60 seconds	
4	System can handle user different request at same time	The system shall refresh the display every 60 seconds System can handle user different request at same time	
Bn	…….	OB_s	

Appendix- C: The Custom Augmentation Approach

It was observed in the previous experiment that the LSTM network was performing satisfactorily when trained with CDA and pre-trained word embeddings. However, it was not showing any learning behaviour until the first ten epochs. To make fair comparison in these section three modifications have been made to check whether the LSTM network used previously would converge in the longer run.

- **Training Configuration for LSTM with CDA Approach**

To validate the performance of LSTM model. Few modifications were performed in the form of some hyperparameter tuning such 1) Extended the number of epochs, 2) Reduce the number of nodes (from 256 to 64) in the second layer of network to check if it effects the performance and, 3) Modification in the learning rate. The same data distribution was used for this experiment as was done for the one under the section 6.3. and the data augmentation was performed with the CDA approach.

- **Experimental results for LSTM**

The observation is shown in the Figure A- 1, when the number of epochs were increased the LSTM not only converged but has achieved extraordinarily good results reaching to 94% classification score at validation loss of 0.275.

Astonishingly, it was observed that by reducing the number of nodes, the network converged earlier without losing the performance. The total number of parameters decreased from 568,029 to 195,549 as can be seen from Figure A- 2.

With the change in learning rate, it was found that initial learning rate won't affect the final accuracy of the LSTM network at all.

From the Figure A- 3, it is evident that it stared improving but still when compared it did not outperform the other three models. It is important to mention here that these experiments on LSTM have been performed only for CDA approach.

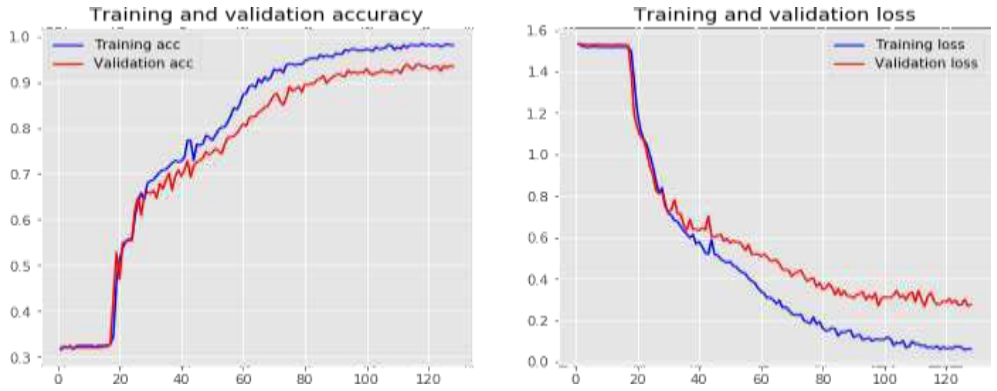

Figure A- 1: LSTM Convergence with Increased Number of Epochs

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 76, 100)	115800
lstm_1 (LSTM)	(None, 76, 64)	42240
lstm_2 (LSTM)	(None, 64)	33024
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 5)	325

Total params: 195,549
Trainable params: 195,549
Non-trainable params: 0

Figure A- 2: LSTM Convergence with Reduced Nodes

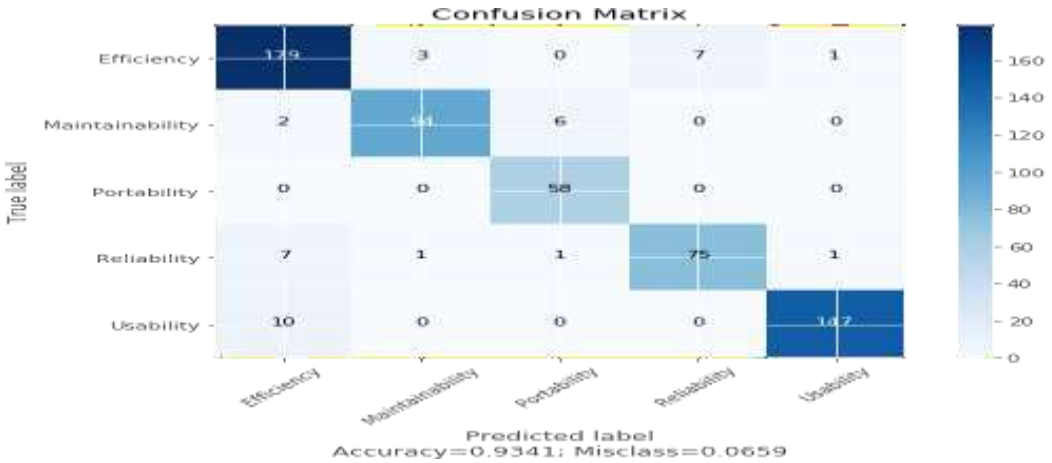

Figure A- 3: LSTM Convergence with Modified Settings

Appendix- E: Results for DNNs based on Accuracy and Loss for Convergence Graph

Table A- 4: Comparison of Various Representation Learning Approaches Based on Training/Validation Accuracy and Loss

Representation Learning	Augmentation	Skip-gram Embeddings	Training		Validation	
			Accuracy	Loss	Accuracy	Loss
CNN	NONE	From Scratch	95.44	0.1	60.27	1.74
	EDA	Pre trained on Eng-CONLL17	97.32	0.06	87.85	0.42
	CDA	Pre trained on Eng-CONLL17	100	0	96.28	0.19
GRU	NONE	From Scratch	94.93	0.11	62.23	1.66
	EDA	Pre trained on Eng-CONLL17	95.53	0.10	86.84	0.37
	CDA	Pre trained on Eng-CONLL17	99.79	0.4	94.93	0.17
ANN	None	From Scratch	93.41	0.20	56.57	1.70
	EDA	Pre trained on Eng-CONLL17	96.20	0.07	87.68	0.34
	CDA	Pre trained on Eng-CONLL17	99.96	0.001	94.76	0.17
LSTM	NONE	From scratch	97.32	0.06	87.85	0.42
	EDA	Pretrained on Eng-CONLL17	92.53	0.10	86.84	0.37
	CDA	Pretrained on Eng-CONLL17	97.51	0.09	89.02	0.35

Appendix- E: Results for DNNs based on Accuracy and Loss for Convergence Graph

Table A- 5: Comparison of Various Representation Learning Approaches Based on Training/Validation Accuracy and Loss for Revised Experiment

Representation Learning	Augmentation	Skip-gram Embeddings	Training		Validation	
			Accuracy	Loss	Accuracy	Loss
CNN	NONE	From Scratch	97.0	0.729	50.00	34.699
	EDA	Pre trained on Eng-CONLL17	86.30	40.00	48.30	34.66
	CDA	Pre trained on Eng-CONLL17	88.58	37.15	60.83	10.78
GRU	NONE	From Scratch	92.74	0.11	52.33	29.20
	EDA	Pre trained on Eng-CONLL17	95.00	11.66	51.33	29.20
	CDA	Pre trained on Eng-CONLL17	100	0.001	51.67	49.13
ANN	None	From Scratch	95.36	0.991	48.00	35.67
	EDA	Pre trained on Eng-CONLL17	96.20	0.07	47.68	34.01
	CDA	Pre trained on Eng-CONLL17	99.92	0.039	48.00	38.66
LSTM	NONE	From scratch	78.14	59.75	51.33	17.68
	EDA	Pretrained on Eng-CONLL17	92.53	0.10	56.84	37.20
	CDA	Pretrained on Eng-CONLL17	78.14	59.75	51.33	17.68

The relevant code and NFR Corpus is available at:
https://github.com/maliha212/CUSTOM-NFRs-Corpus