2021-09

DESIGNING A MODEL FOR MEASURING THE COMPLETENESS OF NON FUNCTIONAL REQUIREMENT USING MACHINE LEARNING

DEMAMU, MELASH

BAHIR DAR UNIVERSITY

BAHIR DAR INSTITUTE OF TECHNOLOGY

SCHOOL OF GRADUATE STUDIES

FACULTY OF COMPUTING

DESIGNING A MODEL FOR MEASURING THE COMPLETENESS OF NON FUNCTIONAL REQUIREMENT USING MACHINE LEARNING

MSC FINAL THESIS

By

DEMAMU MELASH

ADVISOR: SEFFI GEBEYEHU (Ass Prof)

BAHIR DAR, ETHIOPIA

September, 2021

DESIGNING A MODEL FOR MEASURING THE COMPLETENESS OF NON FUNCTIONAL REQUIREMENT USING MACHINE LEARNING

MSC FINAL THESIS

By

DEMAMU MELASH

This Final Thesis is Submitted to the School of Graduate Studies of Bahir Dar Institute of Technology, in Partial Fulfillment for the Degree of Master of Science in Software Engineering in the Faculty of Computing

ADVISOR: SEFFI GEBEYEHU (Ass Prof)

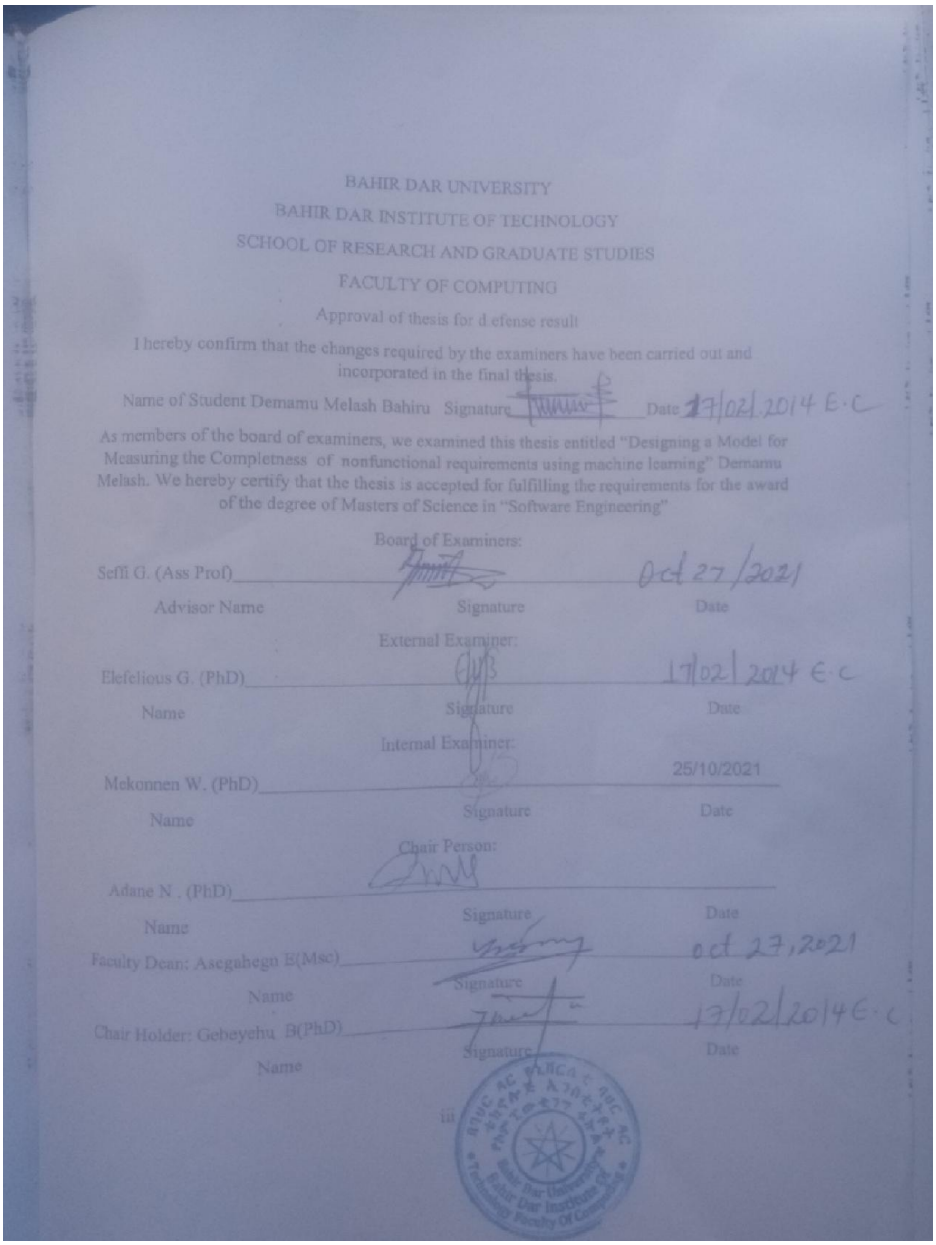
BAHIR DAR, ETHIOPIA

September, 2021

DECLARATION

This is to certify that the thesis entitled "**Designing a model for measuring the completeness of nonfunctional requirements using a machine learning approach**". Submitted in partial fulfillment of the requirements for the degree of Master of Science in **Software Engineering** under **Faculty of Computing**, Bahir Dar Institute of Technology is a record of original work carried out by me and has never been submitted to this or any other institution to get any other degree or certificates. The assistance and help I received during this investigation have been duly acknowledged.

_____	_____	_____
Name of the Candidate	Signature	Date



BAHIR DAR UNIVERSITY

BAHIR DAR INSTITUTE OF TECHNOLOGY

SCHOOL OF RESEARCH AND GRADUATE STUDIES

FACULTY OF COMPUTING

Approval of thesis for defense result

I hereby confirm that the changes required by the examiners have been carried out and incorporated in the final thesis.

Name of Student Demamu Melash Bahiru Signature _____ Date 17/02/2014 E.C

As members of the board of examiners, we examined this thesis entitled "Designing a Model for Measuring the Completness of nonfunctional requirements using machine learning" Demamu Melash. We hereby certify that the thesis is accepted for fulfilling the requirements for the award of the degree of Masters of Science in "Software Engineering"

Board of Examiners:

Seffi G. (Ass Prof) _____ Oct 27/2021

Advisor Name Signature Date

External Examiner:

Elefelious G. (PhD) _____ 17/02/2014 E.C

Name Signature Date

Internal Examiner:

Mekonnen W. (PhD) _____ 25/10/2021

Name Signature Date

Chair Person:

Adane N . (PhD) _____

Name Signature Date

Faculty Dean: Asegahegn E(Msc) _____ Oct 27,2021

Name Signature Date

Chair Holder: Gebeyehu B(PhD) _____ 17/02/2014 E.C

Name Signature Date

iii

Acknowledgments

First and formost I would to thank my God and Ever-Virigin, St Mary ,Mother of our Lord for your blessing and giving me the courage and wisdom to accomplish this thesis, I would like to thank Mr Seffi Gebeyehu(Ass Prof), my advisor for his continual support, unsights and constructive comment, encouregment and patience I greatly appreciate your support from the very beginning from proposal writing to this end. It could not be easier without your valuable comments and corrections. I would also like to thank Dr Gebeyehu forprogress approval up to detail for commenting me. I would like to say thank you for the Promise Expository a well known software engineering repository for dataset we used.

I would like to say thank you my mother and the actor in my success, Birie Mekonnen for being always with me, support and encourage me, nextly my thanks go to my friends at Bahir Dar University Institute of Technology (BiT), especially Mr Getasewu Abeba who support me by giving critical comments for the success of this thesis and those to whom Idid not mention for your direct and Indirect supports, Iwould like to thank my Families for your continual support, patience and encouagements Lots of Thanks GOD bless You all.

Table of Contents

List of Tables

List of Figures

Acronyms and Abbreviations

A	Availability
AI	Artificial Intelligence
BoW	Bag of Words
CHI^2	Chi-Squared
CNN	Convolutional Neural Network
DL	Deep Learning
FR	Functional Requirement
FT	Fault Tolerance
GPU	Graphical Processing Unit
LSTM	Long Short-Term Memory
M	Maintainability
NFR	Non-Functional Requirement
O	Operability
P	Performance
RNN	Recurrent Neural Network
S	Security/Scalability
SRC	Software Requirement Classification
SVM	Support Vector Machine
TF-IDF	Term Frequency–Inverse Document Frequency
U	Usability
KNN	K nearest neighbour

Abstract

In a software development project, the usefulness of a system specification depends on the completeness of the requirements. The software requirement is partitioned into two functional and non-functional requirements. The functional requirement is a requirement software must perform or do, whereas the non-functional requirements are the quality attributes which software must-have. Often a software engineer gives a low priority to non-functional requirements which could lead to a failure of the software or it costs a huge amount of money to fill the incomplete non-functional requirements. Although identifying all requirements is difficult, especially when requirements interact with an unpredictable environment. The main cause for the failure of the system is incomplete requirements. Even if identifying the requirement completeness is a challenging task, researchers try to measure the completeness of the system requirement. It is because of not considering whether or not to fulfill all relevant requirements? The non-functional requirements are selected in the domain of health information systems, the attributes which the study used are availability, privacy, performance, security, reliability and usability. Finally, measuring the completeness of requirements is a researchable area due to its difficulty to know the complete requirements. Therefore, in this research, we proposed machine learning techniques for measuring the completeness of non-functional requirements from SRS documents. After a comparative experimental evaluation of - machine learning classification algorithms (SVM, DTand KNN) ,out of which SVM perform best with F1 score of 92% to deterimine wheter the given health requirement documents is complete or not complete .As we recommend that for future work we compare and contrast our result with that of recent deep learning based classification.

CHAPTER ONE: INTRODUCTION

1.1 Background of the study

In a software development project, requirements completeness plays an important role to ensure all requirements are captured as required. The benefits of a system specification depend on the completeness of the requirements. Identifying all requirements is difficult, especially when requirements interact with an unpredictable environment. The main cause for the failure of the system is incomplete requirements(Alencar et al., 2019). Even if identifying the requirement completeness is a challenging task, researchers try to measure the completeness of the system requirement. The completeness of nonfunctional requirements is different from system to system. In this research, the completeness of nonfunctional requirements was identified in the health. In addition to our work, there is research that describes the complete nonfunctional requirements in health systems (DeVries & Cheng, 2016).

Text classification is a method used to identify a category of a given document (corpus) based on the probability suggested by a trained corpus that is already labeled (Categorized) document. It is widely used under supervised machine learning categories because it requires labeling of the document before classification, and it has several applications like spam email identification and news categorization(Canedo & Mendes, 2020). Furthermore, it can be used for other problems like music, images, video, and other multimedia files.

Text classification is easy with the small number of the document because it is possible to analyze the document manually, and categorized it into which it belongs. Finally, based on the information extracted it is possible to similar documents into their specific classes. However, the method becomes more challenging as the number of document increase to several thousand or millions (Sarkar, 2016).

Software Requirement Classification (SRC) is responsible for specifying the category of Software Requirement (SR) to which category belongs. SR is classified into two categories as a functional requirement (FR) and a non-functional requirement (NFR). Correct classification of requirements is a critical task in the field of software engineering (SE) (Canedo & Mendes, 2020).

Before extraction of the requirements, several documents are reviewed by a requirement engineer. These documents are legacy system documentation, reference standards, and transcripts of meeting with customers to preliminary specifications. The contents of the above documents will assist while writing the requirement of the system (Pohl, Böckle, & Linden, 2005). Since it settles the background on which the future system can start to take its form. This input document is written using a natural language and it's very helpful to extract the functional and non-functional requirements of the future system.

The term 'complete' is usually used when something is not missed from an intended purpose or when it lacks nothing. The Merriam-Webster dictionary defines *complete* as "having all necessary parts, elements, or steps," the definition of the term help for the understanding of the term and the direction of the research. In software requirement, the term used to show, the missing of the requirements which must have been included in a system to be complete. Those requirements could be either FRs which are requirements that specifies a function that a system or system component must be able to perform(IEEE, 1990), or NFRs which are desired qualities of the system to be developed and often influence the system architecture more than functional requirements do (Pohl & Rupp, 2011).

The formal approach to requirement completeness was proposed by (Carson et al., 2004) as a 3 step process namely. The approach depends on capturing the stakeholder's context and quantifying all developmental, operational, maintenance interfaces. The formal approach for requirement completeness is depicted in Figure 1.

Figure 1 The formal approach for requirement completeness(Carson et al,.2004)

Currently, machine learning algorithms are used in many real-life problems from image detection, classification, facial recognition, data science, weather predictions, stock price prediction, and many others. The application of the machine learning algorithms is not limited to this it can also be used in software requirement classification. In this study, machine learning

algorithms were used to measure the completeness of non-functional requirements in the domain of healthcare information system.

In traditional programming, programmers feed data to the device with a rule applied to it, and the computer then generates an output that is a response to a particular problem based on this rule. However, some researchers started to wonder, "Can a machine go beyond what was ordered to do?" and learn how to perform a particular task on its own? Is it possible that a machine could surprise us?(Tyagi, n.d.).

Those questions brought an opportunity to discover another programming paradigm called *Machine Learning*. In this paradigm, programmers input data and answer, expecting *rules* as for output. These rules can be applied to the new data to provide the answer. Unlike classical programming, in machine learning the machine is trained rather than explicitly programmed. For example, to build an automating photo tagging system for vacation photos, a machine learning algorithm can be used. The machine learns a statistical rule by associating previous vacation photographs. Therefore, the vacation photograph is easily tagged using the machine learning algorithm automatically.

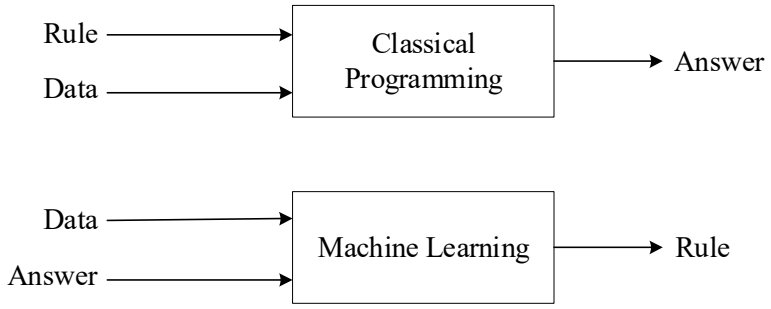


*Figure 2*Machine learning vs classical programming (Tyagi, n.d.)

1.2 Statement of the Problem

Due to ignoring the non-functional requirements of the system advancement of the stage of the London ambulance system, the whole system is deactivated after its deployment. Some of the non-functional requirement which missed from the system are reliability (vehicle location), cost (stress of the best value), usability (poor control of the data on the screen), and performance (the system works but it has less performance)(Kaur et al., 2015). Furthermore, missing non-functional requirements like interoperability and testability NASA's Mars Climate Orbiter and Polar Lander were lost in the Mars environment late in 1999. These missing of the two non-functional requirements cost United States citizens $319m.

Non-functional (NFRs) criteria, like functional requirements, have a large impact on a system's performance. When NFRs are skipped or ignored, serious and costly problems may occur. A $2.7 billion U.S. Army intelligence sharing application that was recently deployed has been deemed useless due to capacity, performance, and usability issues. Electronic health record (EHR) systems have been heavily criticized for their lack of accessibility, which has been one of the prime reasons why some EHRs adoption has failed (Slankas et al., 2013).

The proper functioning of the health care system is critical for the development and advancement of the organizations, society, and individuals. The advancement of information systems helps to enhance the functionality and accessibility to society. The NFRs are additional functional requirements that must be fulfilled by functional requirements. The dissatisfaction of NFRs is one of the main factors for the failure of software projects.. Generally, giving attention to NFRS as early as possible is important before proceed to a detailed design of the system.

Often non-functional requirements were not considered as important as functional requirements. However, the NFRs ensure the quality of the software which more critical than individual components of functional requirements. Some of the major NFRs in Health Information systems are security and privacy, usability, reliability, and availability(Slankas et al., 2013).

. Detail defining of the non-functional requirements helps to improve the performance of the system and increase the satisfaction of users of the system. Furthermore, it decreases the cost needed to fix it after it is deployed.

To build a system gathering software requirements is fundamental whether it is functional or non-functional. The core problem is, all software requirements might not explicitly be stated in the formal requirement specification document and some of the documents miss the critical NFRs guideline(Slankas et al., 2013). Finally, the study answers the following two research questions

RQ 1: How to design a machine learning model for measuringthe completeness of non-functional requirements for software projects?

RQ 2: Which machine learning algorithm suits best to build the model ?

1.3 Objectiveofthe study

1.3.1 General Objective

The general objective of the study is to design a machine learning model for measuring the completeness of non-functional requirements for software projects.

Specific Objectives

The specific objective of the study is to accomplish the following

- To conduct a systematic literature review on measuring the completeness of non-functional requirement
- To prepare a labeled dataset of NFR with health care system
- To bulid the proposed model for measuring the completeness of non functional requirements
- To train and validate the model using a prepared corpus
- To evaluate the performance of the proposed model.

1.4 Scope of the Study

Even though SRC is classified into two categories the study focuses on the non-functional requirements. Specifically, attributes of non-functional requirements like availability, performance, security, privacy , reliability and usability. Furthermore, the ultimate goal of the study is to propose an approach to measure the completeness of NFR via machine learning in the domain of health information systems.Therefore, the study will not consider the functional requirements and other non-functional requirements attributes that are not mentioned above.

1.5 Significance of the Study

Requirement completeness is a core task in software engineering, to deliver a fully functional system it must be measured whether it is complete or not. Missing requirements are hard to spot because they are invisible. Some of the significances of the study are

- It assists the analyst to suggest the missing non-functional requirement
- It can be as a tool for checking the non-functional requirements before it process to the next step of the software development life cycle
- It protects from additional cost the projects pay due to missing the NFRs
- Reduce the risk of missing non-functional requirements

1.6 The Structure of the Thesis

The remaining of the thesis is structured as follows. The next chapter deals with the literature review and summary of related works. The literature review explains a summarized report about measuring requirement completeness, how machine learning was used for functional requirement classification concepts in general. Works are done by different authors that are significantly essential and more related to measuring requirement completeness using machine learning were included in the same chapter, under the review of the related works. The third chapter then goes into the research methodologieswhich have the subsection research flow that explains the steps the research follows to accomplish the research work, data collection that explains the source of data used in the experimentation, data preparation discusses the technique and methods used for making the data suitable for machine learning algorithms. Then, the implementation tools and evaluation metrics are some of the sub-sections of the chapter. Following that, Chapter four describes the experiment and its findings. Finally, in Chapter Five, the study is summarized using the conclusion, contribution, and recommendations.

CHAPTER TWO :LITERATURE REVIEW

2.1 Introduction

In this chapter previously done and highly related paper were reviewed which helps to explain and identify a research gap in the area. In Section 2.2 software requirement completeness, Section 2.3 software requirement specification, Section 2.4 explainstext classification, Section 2.5 describes rule-based, Section 2.6 explains different types of machine learning algorithms related to requirement completeness.

2.2 Software Requirement Completeness

Since requirements guide system design and operation (at least, we hope), it is obvious that the requirements should be complete. Among others(Kar & Bailey, n.d.)emphasize the importance of providing full specifications. However, as (Wiegers & Beatty, 2013)points out, “missing specifications are difficult to identify since they are invisible.”.

The authors present an NLP-based methodology for measuring and improving the completeness of a requirements specification about the requirements specification process's input documents. A requirements document is complete with the input documents if it addresses all of the applicable concepts and connections between concepts expressed in the input documents. Backward functional completeness is the term we use to characterize this form of completeness. To evaluate such completeness, the authors give two metrics that take into account the relevant terms and relevant relationships between terms in the input documents. Furthermore, the authors present a natural language processing (NLP) method for automatically extracting such terms and relations. Completeness Assistant for Requirements (CAR) is a prototype tool that suggests relevant information during the requirements description process and automatically computes the degree of completeness of the generated requirements specification(Ferrari et al., 2014).

The author proposed a knowledge-based approach to the method of requirements engineering. He proposes an ontology as a method for verifying and validating requirements specifications. The classes of the ontology are requirements forms. The examples are criteria statements. A special class structure, the Protégé tool "DL Query," and relationships implemented across a set

of slots are intended to verify the completeness and consistency of the requirements specification(Avdeenko & Pustovalova, 2015).

Since the 1980s, developers have mentioned the difficulty of specification formalization as one of the primary causes of software development project failures. Despite the availability of automation tools for requirements engineering, testing of the requirements specification requires a considerable amount of time on the part of an analyst. The success of this process is determined by the analyst's technical expertise and experience. When developing a broad framework, additional challenges arise due to the cooperation of various analysts and the exchange of objects produced by them. When the number of device specifications approaches tens of thousands, the magnitude of the problem becomes truly immense. When developing a broad framework, additional challenges arise due to the cooperation of various analysts and the exchange of artifacts created by them. When the number of device specifications approaches tens of thousands, the size of the problem becomes truly immense. For these purposes, the tool for supporting requirements analysis verification can be useful, as it assists in reducing the time required for requirements analysis and identifying potentially contradictory instances of requirements(Avdeenko & Pustovalova, 2015).

Non-functional requirements (NFRs) exist in all systems, but they may not be specifically mentioned in a formal requirements specification. Furthermore, NFRs may be enforced externally by government legislation or industry standards. Since certain NFRs are emergent system properties, they necessitate adequate research and design efforts to ensure they are met. When the specified NFRs are not met, projects must incur expensive rework to resolve the issues. The paper aims to help analysts extract relevant non-functional requirements from accessible unconstrained natural language documents more effectively using automated natural language processing(Matsumoto et al., 2017).

The ultimate goal of the research is to assist analysts in extracting specific non-functional specifications from accessible unconstrained natural language documents using automated natural language processing. Data use agreements, installation manuals, regulations, requests for proposals, requirements specifications, and user manuals are among the documents. The paper assesses how efficiently these documents measure and classify non-functional requirements. The

paper discovered NFRs in each of the documents evaluated using a word vector representation of the NFRs, and a support vector machine algorithm performed twice as well as the same input to a multinomial naive Bayes algorithm(Matsumoto et al., 2017).

The authors created NFR Locator, a tool-based method for classifying and extracting sentences from existing natural language texts into their relevant NFR categories. The classification is required to decide the function of a sentence. The paper then extracted essential details unique to each NFR category from sentences marked as non-functional. Access control NFRs, for example, would require the extraction of topics, resources, and activities to establish relevant access control policies. The paper divided sentences into 14 NFR categories: (1) access control, (2) audit, (3) availability, (4) capability and efficiency, (5) legal, (6) look and feel, (7) maintainability, (8) operational, (9) privacy, (10) recoverability, (11) reliability, (12) security, (13) usability, and (14) other(Matsumoto et al., 2017).

The accurate classification of software specifications has developed into a critical task in software engineering. The authors aimed to compare text feature extraction techniques and machine learning algorithms to the problem of requirements engineer classification. Bag of Words (BoW), Term Frequency–Inverse Document Frequency (TF-IDF), and Chi-Squared (CHI2) are the feature extraction techniques used for classifying software requirements into functional and non-functional requirements, and the machine learning algorithms that are used for classification tasks are as follows: Logist Regression (LR), Support Vector Machine (SVM), and Multinomial Naive Bayes (MNN) were the classification algorithms used. Furthermore, the entire feature extraction technique and machine learning algorithm were evaluated on a publicly available dataset named 'PROMISE-EXP,' which contains a labeled software requirements document(Canedo & Mendes, 2020).

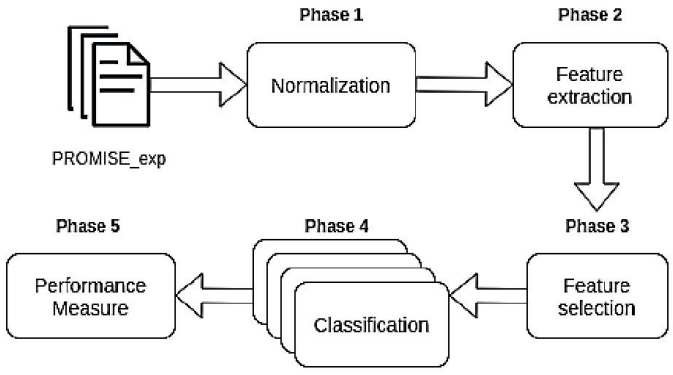


*Figure 3*Software Requirement Classification (Canedo & Mendes, 2020)

The above block diagram represents the phase of software requirement specification classification using the machine learning approach. The phases of machine learning-based approach for classification of software requirements. As depicted in Figure 3 the phases are partitioned into 5 as phase 1(Normalization), phase 2 (Feature Extraction or vectorization), phase 3 (Feature Selection), phase 4(Classification), and phase 5(Performance Measurement).

Data Source:The research used publicly available datasets and resources. The repository was created to facilitate predictive software engineering models that are repeatable, verifiable, refutable, and/or improvable. Several research discipline relies on the dataset for growth and development. Furthermore, the repository aims to broaden its scope to include other areas of software engineering study. The UCI machine learning repository, which is commonly used by researchers in the field, served as inspiration for the dataset repository (Tera-Promise, 2021).

Phase I: Normalization: Text data is pre-processed to a suitable format before it is used to train a given model. Text normalization is often an important step in text pre-processing because it simplifies the modeling process and can boost model efficiency(Binkhonain & Zhao, 2019).

Phase II: Feature Extraction: At this stage, the (normalized) software requirements corpus is converted into numerical vectors that best represent the information found in those requirements.The feature extraction techniques used in the paper are TF-IDF and BoW(Sarkar, Text Analytics with Python, 2016).

Phase III: Feature Selection: To explain this stage, feature selection is the method of removing attributes that are no longer useful for a given task, furthermore this stage is important in data pre-processing because it makes the computational less costly, requires less storage, and requires fewer resources during training(Binkhonain & Zhao, 2019)..

Phase IV: Classification: In this stage, the vectors obtained in the previous steps are used to train and test the model. Some of the machine learning algorithms used in the paper (Monkey Learn, Text Classification, 2021) are SVM, MNB, kNN, and LR.

Phase V: Performance Measurement: This is the final phase of the which classification where the results of the requirement's labels predictions and the true labels of these requirements are used to calculate the performance measures.

2.3 Software Requirement Specification

Software requirement specification contains all the requirements for a system to be and functions.These are usually divided into functional requirements (FR), which define the features of the system under development, and non-functional requirements (NFR), which include quality attributes, design constraints, and other considerations. NFRs are well known for having a significant effect on the total cost and time of the device development process because they often describe cross-cutting concerns (Avdeenko & Pustovalova, 2015).

Requirements Engineering (RE) is one of the most crucial steps of a software project: we know that successful requirements engineering is key to project success or failure, with industry statistics pointing to inadequate RE as the root cause in more than 50% of all failed software projects (Wiegers, 2006).

Table 1: List of non-functional requrements

Lists of Non-functional Requirements(Horkoff, n.d.)
Availability (A)It shows how likely the system is accessible for a user at a given point in time.
Fault Tolerance (FT) describes the degree to which a system, product, or component operates as intended despite the presence of hardware or software faults.
Maintainability (MN) It describes the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers.
Operability (O) describes the degree to which a product or system has attributes that make it easy to operate and control.
Performance (PE) It describes the performance relative to the number of resources used under understated conditions.
Portability (PO) describes the degree of effectiveness and efficiency with which a system, product, or component can be transferred from one hardware, software, or other operational or usage environment to another.
Scalability (SC)describes the degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software, or other operational or usage environments.
Security (SE) It describes the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.

2.4 Text Classification

Text classification is easy with the small number of the document because it is possible to analyze the document manually, and categorized it into which it belongs. Finally, based on the information extracted it is possible to similar documents into their specific classes. However, the method becomes more challenging as the number of document increase to several thousand or millions(Sarkar, 2016)

Text-based unstructured data can be found in a variety of places, including social media, emails, chats, websites, support tickets, survey responses, and more. Text can be a very rich source of information, but due to its unstructured nature, extracting insights from it can be difficult and time-consuming. Fortunately, there is a way to make easy this task easy and efficient using the method called text classification. Text classification (also known as text categorization or text tagging) is the process of categorizing open-ended text into a collection of predefined categories. Text classifiers can be used to organize, structure, and categorize practically every type of text, including documents, medical studies, and files, as well as text found on the internet(Sarkar, 2016).

About 80% of the information in the world has an unstructured data format. This data type is messy by its nature and understanding, analyzing, interpreting, sorting, and organizing the data is hard and time-consuming. Due to this, the potential advantages of unstructured data were not used in many applications.This is where machine learning for text classification comes in. Several real-life problems may use text classifiers to automatically structure all types of related text, including emails, legal documents, social media, chatbots, surveys, and more, in a fast and cost-effective manner. Therefore, using several machine algorithms assists in saving time analyzing text data, automate business processes, and make data-driven business decisions as a result of this(Monkey Learn, Text Classification, 2021).

One of the fundamental tasks of natural language processing is text classification which has a wide variety of applications such as sentiment analysis, topic labeling, spam detection, intent detection, software requirement classification. Text classification using machine learning is often used for software requirement classification whether it’s a functional requirement or non-functional requirement. The technique can be used in three types of the system i. A rule-based system, ii, Machine learning, iii. Hybrid system(Monkey Learn, Text Classification, 2021).

2.5 Rule-Based

Rule-based approaches use a collection of handcrafted linguistic rules to classify text into specific categories. These rules instruct the system to define appropriate categories based on the content of a text by using semantically relevant elements of the text. Each rule has an antecedent or trend as well as a projected category.For Instance: You want to categorize news stories into two categories: sports and politics. First, construct two lists of terms that describe each category (e.g., words related to sports such as football, basketball, LeBron James, etc., and words related to politics, such as Donald Trump, Hillary Clinton, Putin, etc.). Then, when you're ready to identify a new incoming text, count the number of sports-related words in the text and do the same for politics-related words. When the number of sports-related word appearances exceeds the number of politics-related word appearances, the text is listed as Sports, and vice versa(Monkey Learn, Text Classification, 2021).

2.6 Machine Learning

Machine learning is a subfield of Artificial Intelligence that is used to train or teach machines how to handle and manage data more effectively by automatically analyzing patterns and relationships between data features. ML collects information and learns from data by using statistical techniques and computations. It assists in the extraction of valuable and previously unseen patterns and knowledge from data that is difficult to interpret using traditional methods (Kühl et al., n.d.). There are various ML algorithms available depending on the problem domain. These algorithms are categorized into three types: supervised, unsupervised, and reinforcement learning.

Machine learning processes and learns effectively by using huge amounts of data. Most of the time, ML is used for two primary tasks: training/testing and prediction. In the first step, the machine collects training data, which is a collection of examples that helps the machine in learning and detecting patterns among data features. The training data contains information about the domain knowledge from which the examples and patterns are derived. This is known as learning from example. If the size of the dataset increases, the computer can learn more efficiently from the examples and more efficient models can be created.During model training or

learning, independent features are mapped into the dependent feature that must be classified or predicted based on the example. Thesetasks are known as a prediction. The main aim of machine learning is to create a model that capable of predicting or classifying the target outputs based on research, previously unseen, or unknown data (Vemuri, 2020).

2.6.1 Supervised Machine Learning

Supervised machine learning algorithms are ML algorithms that require external supervision. Ituses labeled data and is composed of descriptions, labels, and target outputs. The datasets are divided into two sections by supervising ML. These are the training and testing sets that will be used as data. The training set contains the features that will be predicted and used to train the model. The model's output will then be measured by the test set, which consists of previously unknown or untrained data to the model, following training or learning. When the model is trained using the training dataset, it discovers some patterns that will assist it in identifying new data (Vemuri, 2020; Cheng et al., 2002).

The dataset contains features, which are classified as dependent or target features that must be predicted by the model, and independent features that assist the model in predicting the target output. By mapping and determining the relationship between the dependent and independent features, the algorithm generates some specific functions. During model training, the mapping is performed. The training process continues until either the model is well trained or the performance is optimized(Tesfa, 2020).

Classification is a type of supervised machine learning algorithm with a discrete value as its target outcome. It is mainly used to categorize dependent variables into possible target groups. It is majorly divided into binary and multi-class classification problems. In binary classification, the dependent variables contain only two classes to be predicted whereas in multi-class classification problems the dependent variable contains more than two classes to be classified or predicted(Vemuri, 2020). The research aims to use supervised machine learning algorithms to measure the completeness of non-functional requirements in the domain of health care information system. Here some supervised machine learning algorithms widely used by several scholars are listed.

2.6.1.1 SVM (Support Vector Machine)

Is a type of supervised machine learning algorithm which can be used for classification and regression. The algorithm is based upon statisticallearning theory and Vaprik-Chevonenkis (VC) dimensions and the algorithm looks for all data points that are close to the opposite class. The data points are known as support vectors used for classification tasks, the other data points are ignorable (Binkhonain & Zhao, 2019) SVM is a non-linear Classifier. This is a new trend in machine learning algorithms which is used in many pattern recognition problems, including texture classification. The input data in SVM is non-linearly mapped to linearly separated data in some high dimensional space, resulting in good classification results. SVM aims to minimize the marginal difference between groups. The division of classes is carried out with different kernels. SVM is intended to work with only two classes by determining the hyperplane to divide them. This is accomplished by maximizing the margin between the hyperplane and the two classes. The samples chosen for review were those nearest to the margin of error.Multiclass classification is also valid and is built up by different two-class SVMs to solve the problem, either using one-versus-all or one-versus-one. The winning class is then determined by the highest performance function or the most votes, whichever is greater. The main advantage of SVM is its prediction accuracy is high. Even if the training example includes mistakes, it is less sensitive and versatile. SVMs, like neural networks, have a computational complexity that is independent of the dimensionality of the input space. However, this classifier includes a long learning time. It is also challenging to comprehend the learned function (weights)(Kahsay, 2019).

2.6.1.2 Decision Tree(DT)

A decision tree is another supervised machine learning algorithm that uses hierarchical and statistical models to categorize data. DT's regression and classification models have a tree structure (Russell, n.d.). Statistical methods and experimentation on the data were used to choose the root node attributes. Then, recursively go through the other attributes until no attributes remain at the end node or leaf node. While constructing the algorithm's tree structure, entropy and information gain are used to obtain the statistical measure for building the tree. The entropy of the attribute is calculated as Eq 1.

$$H(s) = -\sum P(x) log_2 P(x)$$ ……. Eq 1

Where,

- H(S): Entropy of the dataset
- S: The current dataset for which entropy is being calculated
- x: Set of classification in S
- P(x): The probability of x

Information benefit is a statistical metric that assists in the selection of a decision and root node. Attributes with the highest information gain are chosen as the root node. The tree's hierarchy can then be constructed by measuring the information gain of the other attributes recursively. The information gain of the attribute is calculated using Eq1.

$$IG(A,S) = H(S) - H_A(S)$$ ……. Eq 2

Where,

- IG (A, S): Information gain of a specific attribute 'A'
- H(S): Entropy of the dataset
- H_A (S): Entropy of the specific attribute in the given dataset.

2.6.1.3 K-Nearest Neighbors (KNN)

KNN is a supervised learning algorithm that stores all available cases during training and makes an inference and classifies a new data point group by a majority vote of its k neighbors (Swamynathan, 2019). Since it makes no assumptions on the original results, KNN is a non-parametric and lazy learner algorithm. The Euclidean, Manhattan, Minkowski, or Weighted distance functions are used to calculate the K nearest neighbors. If K is equal to one, the example is simply given to the class of its nearest neighbor(Binkhonain & Zhao, 2019). The steps of the KNN are as follows:

Step 1: Choose k number of neighbors

Step 2: Calculate the distance;

Step 3: Gets the k nearest neighbors;

Step 4: Count the amount of class appears with the shortest distance;

Step 5: Takes the class that appeared many times;

Step 6: Classifies the new data with the class that took in step 5;

2.6.2 Unsupervised Machine Learning

Unsupervised learning is a machine learning technique that discovers patterns in data without the use of any guidance. There is no target output or outcome to predict in this algorithm. It is typically used to group or cluster features without prior information based on similarities and patterns (Bhattacharyya & Lal, 2020). In this technique, there is no need to divide the dataset into training and testing to train and test the model. So the machine should learn and find patterns among the data features and cluster or associate them into groups of similarities on its own. It discovers hidden patterns in unlabeled data. As a consequence, the unsupervised learning algorithm aims to efficiently group the given dataset into a certain number of clusters, and they are extremely effective techniques for analyzing and identifying hidden patterns and trends in given datasets (S. Kaur et al., 2019).

2.6.2.1 K-Means

K-Means is clustering algorithm is an example of an unsupervised machine learning algorithm (Swamynathan, 2019). Using unlabeled data, this approach is used to solve the clustering problem, which divides data into k number of clusters. Without training the algorithms, it iteratively assigns each data point to one of the k clusters based on the given features. The similarity of the features is used to identify each cluster. It employs a centroid and minimizes the distances between data points as well as their clustering of the data.The k-means algorithm is an iterative process that continues until the algorithm fails to find the best clusters to produce the final result. It accepts k clusters as input, each of which contains a set of variables for each instance in the dataset.

The K-means clustering steps are

Step 1: Decide the number of k clusters

Step 2: Estimate k centroid

Step 2: Assign each sample to the closest cluster centroid based on the squared Euclidian distance

Step 3: Iterate the steps until all samples are clustered (Cutler & Dickenson, 2020).

2.7 Hybrid System for Text Classification

The hybrid text classification methods use the combination of machine learning algorithms and rule-based systems which improves the results. These hybrid systems can be easily fine-tuned by adding specific rules for conflicting tags that the base classifier did not correctly model using a base classifier (Monkey Learn, Text Classification, 2021).

2.8 Measuring the Completeness of Non-functional Requirement

The Merriam-Webster dictionary defines complete as “having all necessary parts, elements, or steps,” which is consistent with our intuitive understanding. Theterm completeness must be described in terms of some metric. A given set of specifications, for example, could well define the requirements for one system while being completely insufficient (incorrect and/or incomplete) for another (Carson et al., 2004).

2.9 Summary of Related Works

It was quite challenging to find a related paper measuring the requirement completeness via machine learning or other approaches. Most of the work is focused on requirement classification or validation via several machine learning algorithms. Table 2 depicts the summary of related works highly related to the area of research.

Table 2: summery of related works

SN.	Author(s) and Year	Techniques used	Main findings	Limitation
1.	(Canedo & Mendes, 2020)	Logistic Regression (LR), Support Vector Machine (SVM), Multinomial Naive Bayes (MNB), and k-Nearest Neighbors (KNN)	Software requirement classification	- Cannot identify the completeness of nonfunctional requirements
2.	(Matsumoto et al., 2017)	A prototype system based on Java programming language	Verification of non-functional requirements	- It is not scalable as well as to measure the completeness of nonfunctional requirements
3.	(DeVries & Cheng, 2016)	Symbolic analysis of hierarchical requirement model	Automatic detection of incomplete requirements via symbolic analysis	- Cannot learn from the data. - It is not scalable
4.	(Avdeenko & Pustovalova, 2015)	An ontology-based approach to support the completeness and consistency of the requirement specification	Verification and validation of requirement specification	- The techniques used are not machining learning. - It does not identify whether the nonfunctional requirement is complete and incomplete
5.	(Slankas et al., 2013)	Machine learning algorithms	Automatic extraction of non-functional requirements in available documents	- Cannot learn from the data and also cannot measure it is complete or incomplete nonfunctional requirements
6.	(Binkhonain, M., & Zhao, L. 2019).	Machine learning algorithms	Identification and classification of non-functional requirements	- Only identify requirement - Not identify the SRS whether it is complete or incompletein nonfunctional requirements. Coverage
7	(Slankas et al., 2013)	Non-functional requirements in health information systems: A systematic mapping research	Performance , Security, privacy, usability, reliability and availability are identified as significant non functional requirements for health information system .	- Cannot learn from the data. - It is not scalable - It is not identify whether the nonfunctional requirement is complete and incomplete

CHAPTER THREE: RESEARCH METHODOLOGY

3.1 Introduction

To achieve the research goals, the researcher develops a technique or experimental procedure that involves all machine learning steps. Details of how the data is collected, understanding the collected data, feature selection, preprocessing such as data cleaning, data formatting, and methodologies used for the model building will be discussed in the subsequent sections.

3.2 Research Flow

This study's phase is divided into three major steps. The problem is described in step one by reviewing various literature and identifying the problem, which leads to the objective formulation. The second phase consists of data preparation, which involves feature selection, data preprocessing, and data transformation, after which the prepared data set is divided into training and test sets. In the final step, a model is created and compared using the evaluation metric to determine the best model. Finally, a report will be written based on the study's results and conclusions. Figure 4depicts the study's overall research flow of the study.

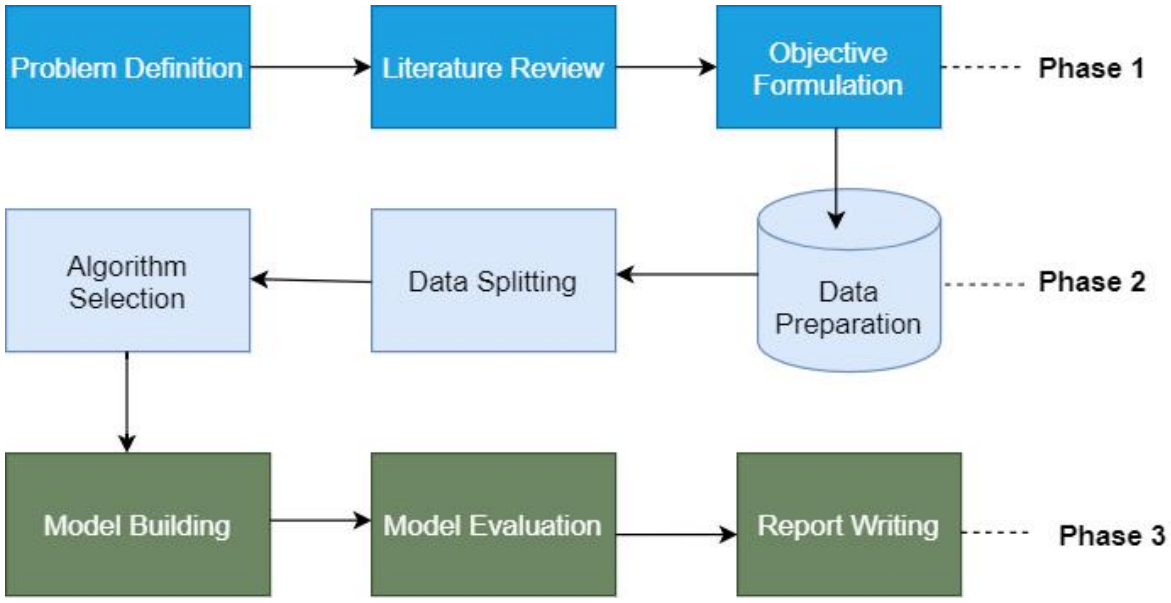


Figure 4: Research Flow

3.3 Data Preparation

Data preparation is the process of making data suitable for the machine learning algorithms and algorithms that will be used to build models. To move on to the next task, every machine learning tool and software package requires some kind of data format. As a result, the data must be converted into a format that is compatible with the relevant tool and software.

The first step in every machine learning task is to understand the problem domain and the data that will be used in the process. Data preparation tasks such as data collection, data and business understanding, dataset description, and data formatting have been undertaken in the following sections. Finally, the feature selection and data pre-processing phases are then followed by data cleaning.

3.4 Data Collection

A promise is software engineering publicly accessible datasets and resources to assist researchers in developing predictive software models (PSMs) and the software engineering community as a whole. The repository was designed to promote repeatable, verifiable, refutable, and/or improvable predictive software engineering models. This is important for the growth and development of every research discipline. The repository aims to expand into other areas of software engineering research. The PROMISE repository was inspired by the UCI Machine Learning Repository, which is widely used by researchers in the field. The source of data is found in the Github repository (Tera- promise, 2021).

3.5 Data Pre-processing

Most real-life data is dirty meaning might have features not required by the model, therefore before applying a different machine learning algorithm the data pre-processing is required. The dataset pre-processing methods vary as the nature of the data. In conclusion, carefully preparing the dataset increases increase the accuracy of the proposed work. In our case, the benchmark dataset has both functional and non-functional requirements. Since the scope of the study only focuses on the non-functional requirement in the domain of health care system, the NFR row and columns were removed.

In [410]:
```
#
RANDOM_STATE = 42
#path = 'E:/MSC DATA/thesis/condlem.csv'
df = pd.read_csv("E:/MSC DATA/thesis/de/demdata.csv",encoding='UTF-8')
dff = pd.read_csv("E:/MSC DATA/thesis/de/demtest.csv",encoding='UTF-8')
df = df[['RequirementText','class']]
X = df['RequirementText']
T = dff['RequirementText']
y = df['class']
ty = dff['class']

# Split train & test
#text_train, text_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
text_train=X
text_test=T
y_train=y
y_test=ty
# Tokenize and transform to integer index
tokenizer = Tokenizer()
tokenizer.fit_on_texts(text_train)
X_train = tokenizer.texts_to_sequences(text_train)

X_test = tokenizer.texts_to_sequences(text_test)
word_index=tokenizer.word_index
#print(X_train)
```

```
vocab_size = len(tokenizer.word_index) + 1  # Adding 1 because of reserved 0 ind
maxlen = max(len(x) for x in X_train) # longest text in train set

maxlen2 = max(len(x) for x in X_test) # longest text in train set
for x in X_test:
    if len(x)==maxlen2:
        print(x)
# Add pading to ensure all vectors have same dimensionality
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)

X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

```
[269, 9, 10, 95, 114, 45, 1627, 27, 15, 4, 16, 8, 5, 3, 435, 114, 66, 17, 44, 1
14, 142, 235, 222, 114, 8, 5, 3, 142, 11, 235, 222, 9, 10, 15, 4, 114, 8, 5, 3,
15, 16, 3, 5, 54, 11, 11, 222, 17]
```

```
In [411]: df
```

Out[411]:

	RequirementText	class
0	system shall refresh display everi s...	1
1	product shall respond fast keep data...	1
2	product shall produc search result a...	1
3	search result shall return later thi...	1
4	product shall gener cma report accep...	1
...	...	...
1164	iii person subject jurisdict food dr...	6
1165	iii individu access protect health i...	6
1166	iii disclos may permit health insur ...	6
1167	iii group health plan health insur ...	6
1168	iii name specif identif person class...	6

1169 rows × 2 columns

Figure 5: Sample code snippet for preprocessing

3.6 Understanding the NFR Data

Before taking any measures, the main activity in any machine learning task is data understanding. It begins with obtaining initial insights into the data and the problem domain. Later, it moves on to other tasks such as determining the consistency of the data, missing values, and outliers that may affect the machine learning final result. Understanding the data assists in finding valuable and interesting insights into the data that can be used to establish a hypothesis for the hidden or unknown information.

In this study, the data collected has a description of non-functional requirements in the domain of health care. The non-functional requirements that exist in the data sets are availability, performance, security, usability, reliability and privacy . First, the description of each non-functional requirement is classified into the correct categories of NFR mentioned earlier.

Finally, the completeness of the NFR measured if it has all the 6 NFR mentioned earlier. If it satisfied all the 6 NFR to be complete NFR' if not 'Incomplete NFR'.

3.7Tokenization

Is a process of splitting a given text into a set of meaningful pieces. The pieces of text are called tokens, and the data pre-processing task is named after it(Online, 2020)Tokens are the building blocks of Natural Language and the most common way to process the raw text applied at the token level. Different tokenization can be used based on the type of problem at hand, but the state of art deep learning architecture for NLP like RNN, GRU, and LSTM also process raw text to the token level (Pai, 2020).

3.8Normalization

Is a method that consists of a series of steps that should follow to wrangle, clean, and standardize the text in the document. In a simple explanation, text normalization is the process of converting lists of words into a more uniform sequence. This stage of data pre-processing can improve the text matching in the document(Sarkar, 2016). In this study, the raw data will be normalized using different methods, so that the words in the document will have a standard form.

3.9Feature Extraction (Vectorization)

To apply a machine learning algorithm on text transformation of instance, documents, in vector representation is required so that it possible to apply numeric machine learning. The process of encoding documents into numeric feature space is called feature extraction or vectorization. The encoded document has numeric feature space which is a two-dimensional array where rows are instances and columns are features. In this study, using vectorization the document will be encoded into numeric feature space so that we can apply it differently (Canedo & Mendes, 2020).

3.10Feature Selection

In some cases, text vectorization generates a weak informative feature for text classification, which may not have increase the performance of a machine learning algorithm. Simply, feature selection is the process of dropping the attributes that provide no useful information for a given task. Feature selection is an important step in data pre-processing because it makes the problem computationally less costly, required less storage, and takes fewer resources while training. In

the proposed work a feature selection will be applied to remove weak informative attributes (Canedo & Mendes, 2020).

3.11Algorithm Selection

According to a thorough analysis of related work, several machine learning algorithms were used to classify non-functional and functional data, and the results were promising. The figure below depicts the chosen classification algorithms that were used for model building to measure the completeness of non-functional requirements.

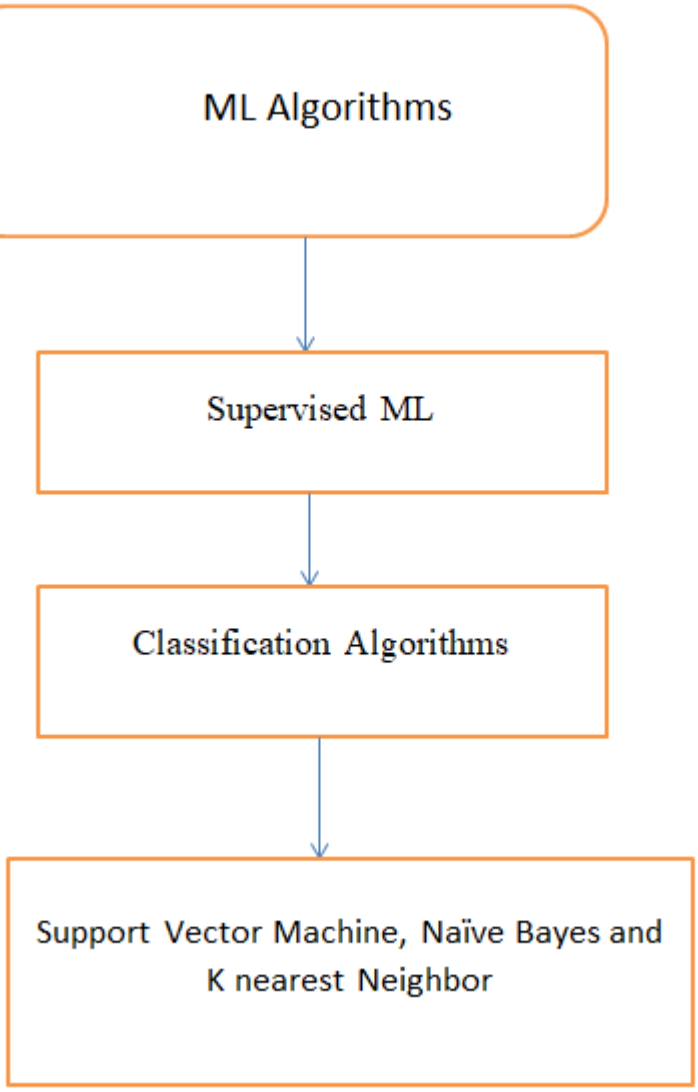


Figure 6: Algorithm selection

3.12Handling Class Imbalance

Imbalance databases are defined by an unequal distribution of data samples. This data disparity happens because certain groups have a limited number of samples or representatives, referred to as minority classes in the dataset, whereas other classes have a larger number of samples, referred to as majority classes. A misclassification problem occurs when a projection model is learned from an imbalanced dataset, and the system is normally biased against the dominant groups by a large number of members (Calude et al., 2005).

3.13Model Training and Testing

Following the completion of data preparation and algorithm selection, classification model building and training for the selected algorithm was carried out. In Chapter 2, we go through machine learning classification strategies in depth. The models are built using the three supervised machine learning classification algorithms,SVM,DTand KNN. To train and test the proposed machine learning classification algorithm, two separate model- building techniques are used. Specifically, a train/test split.

3.13.1 Train/Test Split

The train/test split is used to evaluate the ML algorithm's results. It separates the datasets into training and testing sets. The training set is used to train the algorithm and assist the model in learning on this data, which will later be mapped to other previously unknown data. The test dataset is useful for evaluating the model's prediction performance on this subset.Since the dataset preprocessed 1169rows are not very large, the dataset is divided into an 80% training and a 20% testing collection, when you have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced the recommenede rule is 80% by 20%and also this rule to remove or minimize noisy of data , if the Computational cost in training the model, evaluating the model, Training set representativeness and Test set representativeness we used this rule.As a result, **941** was used as a training set and **228** was used as a testing set from the complete instances of the dataset.

3.14Model Architecture

The model architecture of the proposed work using several machine learning algorithms was depicted in figure 7. As the figure depicts the model architecture begins with the dataset collected from the source and passes to the data preparation stage which has the feature selection and data processing steps.There is no optimal split percentage for train, and test data, but we use the recommended percentage of 80% for train, and 20% for testing the model(*Train-Test Split for Evaluating Machine Learning Algorithms*, 2020).This methodwas applied for all machine learning algorithms used in this work, and comparing all the three machine learning algorithms the best model selected based on evaluation metrics explained in chapter three. Finally, using the best performing ML model the completeness of non-functional requirements was measured. The model gives us the 'incomplete NFR' result if one of the NFR missed after the classification result was generated.After a classification model development the nextis to whether a given requirement is complete or incomplete for health- related system. So first the researcher identified the type of NFRs and then checked whether the SRS classification result includes all significant NFRs for health systems. The significant NFRs are identified in the literature for health system domains.

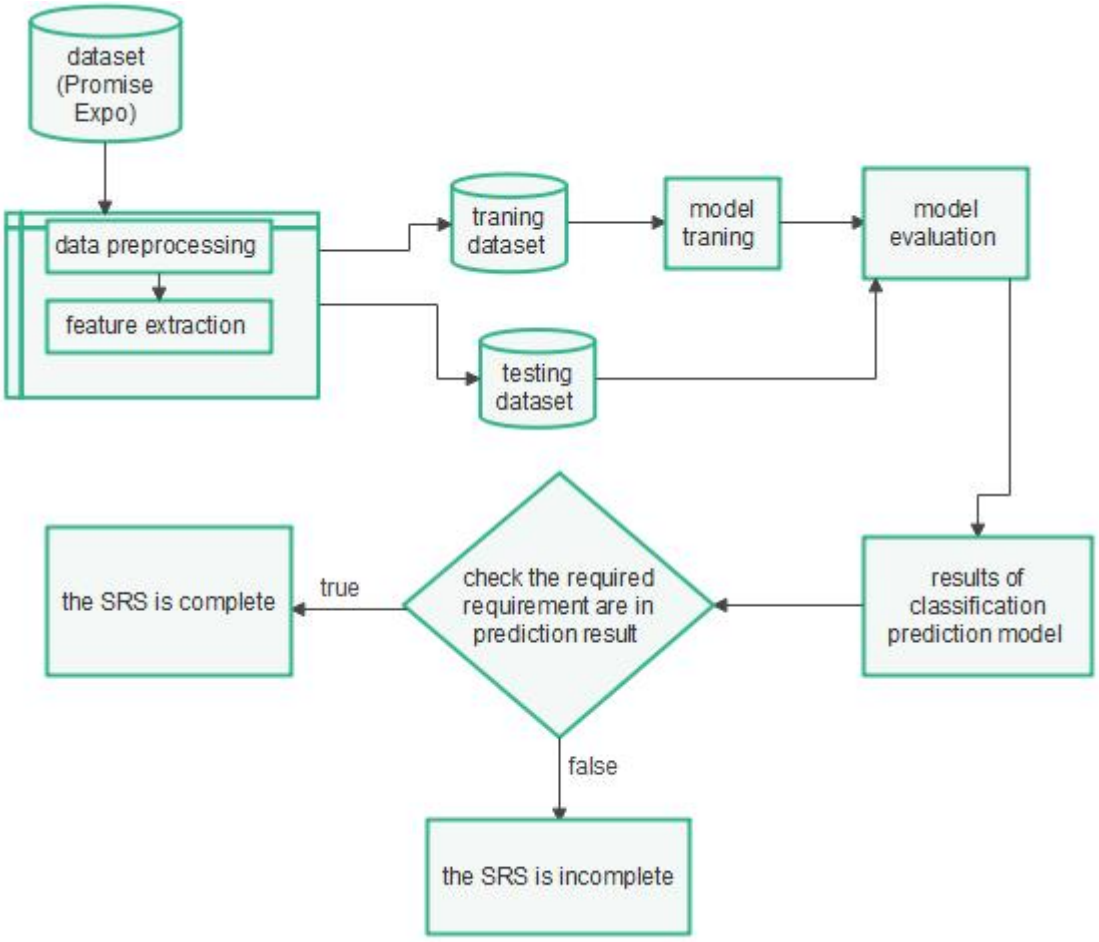


Figure 7: Model Architecture

3.15Implementation Tools

The implementation uses a python programming language because the programming language has many libraries that support to solve the problem like text classification. Below, the tools and programing language that will be used mentioned. Based on the literature review and ease of usingcriteria the following tools are selected.

- **Jupyter Notebook**: used as a code editor because it allows creating and document source code, visualization, and plain text.
- **Python**: A general-purpose programming language highly recommended for machine learning researches and experimentations.

- **Notepad++**: This is a text and source code editor and will be used to manipulate raw text during dataset preparation.
- **Matplotlib**: A python library that generates publication-quality figures. This study will use for data visualization.
- **NLTK**: Build a python program to work with human language data. The tool will be used during the data pre-processing stage.
- **Seaborn**:This is a tool to help with statistical data visualization.
- **Numpy**: This is a library for mathematical functionality.

Figure 8: Implementation tools

3.15.1 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in the Anaconda® distribution, which is a free Python distribution designed for scientific computing and developed by Continuum Analytics. It is free to use and for commercial purposes. It is free to use and for commercial purposes. The best part, however, is that it comes with a range of pre-installed packages that are needed for data science, math, and engineering. Anaconda Navigator enables us to launch applications and control Conda packages, environments, and channels without the need for command-line tools. Navigator may look for packages in the Anaconda Cloud or a local Anaconda Repository. It is available for Windows, macOS, and Linux, with version 1.9 being used for this study.

3.16Evaluation Metrics

Model evaluation task for determining how effective and efficient a model is. Accuracy, precision, recall, and F1-score are all evaluation metrics used in many machine learning problems. These metrics assist in calculating the model's performance. In the case of a classification ML problem, metrics provide insight into the output of a model as well as which classes instances belong to; based on the metrics value, it is possible to determine whether instances were predicted correctly or incorrectly. All of the metrics mentioned above are calculated based on the confusion matrix value, namely TP (True Positive), TN (True Negative), FP (False Positive), and FN (False Negative) (False Negative). The confusion matrix is shown below(Binkhonain & Zhao, 2019).

Table 4: Confusion matrix

Actual values

Predicted value

	Negative	*Positive*
Negative	TN	FN
Positive	FP	TP

Where:

TN: Observation is positive and predicted to be positive.

TP: Observation is negative, and predicted to be negative.

FP: Observation negative, but predicted positive.

FN: Observation is positive, but predicted negative.

- $\boldsymbol{Accuracy} = \frac{\boldsymbol{TP+TN}}{\boldsymbol{TP+FP+TN+FN}} \ldots\ldots\ldots\ldots$ *Eq3: Accuracy (Classification metrics)*
- $\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FP}} \ldots\ldots\ldots\ldots\ldots\ldots\ldots$ *Eq4: Precision (Classification metrics)*
- $\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FN}} \ldots\ldots\ldots\ldots\ldots\ldots\ldots$ *Eq5: Recall (Classification metrics)*
- $\boldsymbol{F1-Score} = \frac{\boldsymbol{2}}{\frac{\boldsymbol{1}}{\boldsymbol{Recall}}+\frac{\boldsymbol{1}}{\boldsymbol{Precison}}} \ldots\ldots\ldots$ *Eq6: F1-Score (Classification metrics)*

i. **Accuracy**: Answers the question about how often the model predicts the classes correctly i.e. complete and incomplete.

ii. **Precision**: Is gives insight into how often a positive value prediction is correct. Example: Predicting a requirement uncompleted, how often the prediction precisely predicts.

iii. **Recall**: Also known as sensitivity it describes how sensitive the classifier is while detecting positive instances.

iv. **F1-Score**: This is the harmonic mean of the precision and recall, and the lowest value of the F1-score is 0 which means one of the metrics has a value of 0. It indicates perfect precision or recall.

Chapter Four :Result and Discussion

4.1 Introduction

This chapter explains and explores the research's practical application and experimental results. It includes extensive details about the machine that was used for the implementation, as well as the programming language, software tools, and libraries that were used. The chapter describes all of the experiments used to develop the model and concludes by evaluating the performance of classification models and reporting their results.

4.2 Development Environment

This research work is carried out on a computer that follows the specifications mentioned below. HP Folio PC with an Intel® Core i7-6100U CPU running at 2.30GHz, 8GB RAM, a 2TB hard drive, and Windows 10 (Pro) installed. To train and develop a model, an updated version of Python 3.8 and Anaconda Navigator are installed.4.3.

4.3 Model Building with Under Sampled Data

All of the models in this experiment are trained using SMOTE oversampling, which is another class imbalance handling technique. In this technique, synthetic samples from the dataset's minority classes are oversampled into the number of instances of the majority class, which in this dataset is a small injury of any instances. Resulting from the application of SMOTE through the‘imblearn' library, all classes are oversampled into any instances.

4.4 Experimentation on the Machine Learning Algorithms

The supervised machine learning algorithms experimented within this study, Specifically called support vector machine, Decision Tree and K nearest neighbour is the algorithm experimented within this work.The reason behind selecting these algorithms was the amount of data available, the simplicity to understand and train. Below is section 2.12.1 the experimentation result of-support vector machine is explained, following Decision Treeand K- nearest neighbour algorithms.

4.4.1 Support vector machine

The dataset prepared was used to train a model with support vector machine, the training and testing were partitioned as 80% training and 20% testing. the evaluation metrics mentioned all the classification reports for all classes (6 Classes). Since for unbalanced data, the metrics recommended is F1-score, referring to this evaluation metrics for support vector machine - it achieved an F1-score of 92%. The result found is good but also have a sufficient amount of data could improve their performance - The sample code snippet and classification report of the support vector machine is depicted below the figures.

Figure 9 Snipshoot code of SVM classifier

```
In [414]:  # Training Support Vectore Machine (SVM) classifier


svclassifier =SVC()
svclassifier.fit(X_train, y_train)
svm_predictions = svclassifier.predict(X_test)
```

```
In [417]:
print(confusion_matrix(y_test,svm_predictions))
print(classification_report(y_test,svm_predictions))
print(accuracy_score(y_test, svm_predictions))
```

```
[[ 32   0   0   0   0   0]
 [  0 103   0   0   0   1]
 [  0   3  29   0   0   0]
 [  0   3   0   8   0   0]
 [  1   4   0   0  24   0]
 [  0   6   0   0   0  14]]
              precision    recall  f1-score   support

           1       0.97      1.00      0.98        32
           2       0.87      0.99      0.92       104
           3       1.00      0.91      0.95        32
           4       1.00      0.73      0.84        11
           5       1.00      0.83      0.91        29
           6       0.93      0.70      0.80        20

    accuracy                           0.92       228
   macro avg       0.96      0.86      0.90       228
weighted avg       0.93      0.92      0.92       228

0.9210526315789473
```

4.4.2 Decision Tree

The prepared dataset was used to train a model ofDecision Tree, with the training and testing divided as 80 percent training and 20% testing. The assessment metrics listed all classification results for all grades (6 Classes). Since the F1-score is the suggested metric for unbalanced data, referring to this measurement metric for Decision Tree, it achieved an F1-score of 91%. The found result is good, but having a sufficient amount of data could improve things. Below the figures is a sample code snippet and classification report for the Decision Tree.

Figure 10 Snipshoot of DT classifier

```
In [492]: from sklearn.tree import DecisionTreeClassifier
          dtree=DecisionTreeClassifier()
          dtree.fit(X_train,y_train)
          dtree_pred = dtree.predict(X_test)

In [493]: print(confusion_matrix(y_test,dtree_pred))
          print(classification_report(y_test,dtree_pred))
          print(accuracy_score(y_test, dtree_pred))

[[32  0  0  0  0  0]
 [ 1 99  3  0  0  1]
 [ 0  2 30  0  0  0]
 [ 1  1  1  8  0  0]
 [ 2  0  2  1 24  0]
 [ 0  2  0  0  2 16]]
              precision    recall  f1-score   support

           1       0.89      1.00      0.94        32
           2       0.95      0.95      0.95       104
           3       0.83      0.94      0.88        32
           4       0.89      0.73      0.80        11
           5       0.92      0.83      0.87        29
           6       0.94      0.80      0.86        20

    accuracy                           0.92       228
   macro avg       0.90      0.87      0.89       228
weighted avg       0.92      0.92      0.92       228

0.9166666666666666
```

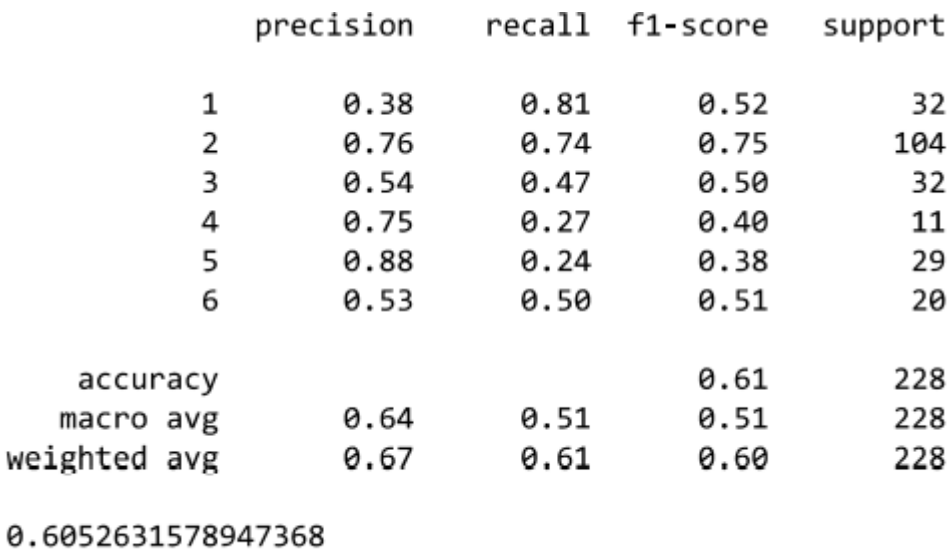
4.4.3 K nearest neighbor

The prepared dataset was used to train a model ofKNN, with the training and testing divided as 80 percent training and 20% testing. The assessment metrics listed all classification results for all grades (6 Classes). Since the F1-score is the suggested metric for unbalanced data, referring to this measurement metric for KNN, it achieved an F1-score of 61%. The found result is low, but having a sufficient amount of data could improve things. Below the figures is a sample code snippet and classification report for the KNN.

```
In [415]: # Training KNN classifier


          knn_classifier = KNeighborsClassifier(n_neighbors=3)

          # Train the model using the training sets
          knn_classifier.fit(X_train, y_train)
          knn_predictions = knn_classifier.predict(X_test)
```

```
In [416]:

          print(confusion_matrix(y_test,knn_predictions))
          print(classification_report(y_test,knn_predictions))
          print(accuracy_score(y_test, knn_predictions))
```

```
[[26  4  0  0  0  2]
 [18 77  3  0  0  6]
 [ 6  9 15  0  1  1]
 [ 4  2  2  3  0  0]
 [ 9  5  7  1  7  0]
 [ 5  4  1  0  0 10]]
```



	precision	recall	f1-score	support
1	0.38	0.81	0.52	32
2	0.76	0.74	0.75	104
3	0.54	0.47	0.50	32
4	0.75	0.27	0.40	11
5	0.88	0.24	0.38	29
6	0.53	0.50	0.51	20
accuracy			0.61	228
macro avg	0.64	0.51	0.51	228
weighted avg	0.67	0.61	0.60	228

0.6052631578947368

Figure 11 Snipshoot of KNN classifier

The study's ultimate objective is to use a various machine-learning algorithms to determine the completeness of non-functional requirements. The results depicted in the preceding sections, as well as the study's final findings, help in answering the research questions raised at the start of the study.designing a model for Measuring the completeness of non functional requirements using machine learning techniques with tfidf vectorization comparison results

Classification techniques	Precession	Recall	F1 score
SVM with TFIDF	96%	86%	92%
DTwith TFIDF	90%	87%	91%
KNN with TFIDF	64%	51%	61%

Table 5 Comparison Results of classifiers

We can show the result in charts

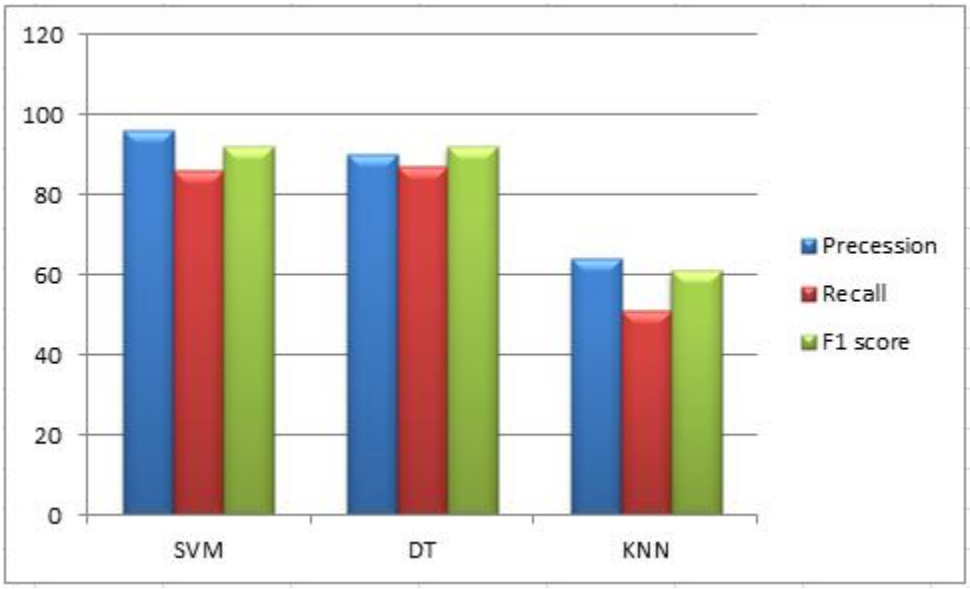


Figure 12 Comparison results

RQ 1: How to design a machine learning model for measuring the completeness of non-functional requirements for software projects?

Machine leraning algorithms learn from the data without explicitly programmed. In this study several machine learning algorithms were studied and experimentedto measure the completeness of the non-functional reqirments. The algorithms applied using the available data with 80-20% train and test split. Finally, the trained model capable of providing an answer whther or not the non-functional requirent in the domain of health care system are complete or not complete. All the NFR quality attributes in our experiment is equal value or it should divide the 100% in to six eqal parts. Sample code is looks like the following.

```
In [160]: uniquevalue = np.unique(yhat)
          uniquevalue

Out[160]: array([1, 2, 3, 5, 6], dtype=int64)
```

```
In [163]: total=len(uniquevalue)
          if(total==6):
              print("complete")
          elif(total==5):
              print("83 % complete")
          elif(total==4):
              print("66 % complete")
          elif(total==3):
              print("50 % complete")
          else:
              print("in complete")

          83 % complete
```

Figure 13 Snipshoot of Compelete and incomplete NFR

RQ 2: Which machine learning algorithm suits best to build the model ?

Three supervised machine learning algorithms were experimented and among them the support vector machine has achieved with 92% F1-score. The performance results of each classifier are depicted and explained on each experimental part. So, after doing different experiments by considering the problem domain we can conclude that the performace of the model would increase having more dataset prepared.

4.5 Discussion of the results

Firstly SVM's are very good with F1 score of 92%, when we have no idea about the data (Taneja et al., 2014). even if the dataset is smaller because the algorithm does not rely on the entire data (Rao, 2016).

Secondly KNN Performs wellwith F1 score of 61%, on applications which has a sample with many class labels mostly it is effective if the training data is not small but KNN classifiers give each attribute equal weight. When there are many irrelevant attributes in the data, this can lead to uncertainty and poor accuracy(J. Wang & Li, 2010). Thirdly, Decision tree works good with F1 score of 91% because Missing values in the data also do not affect the process of building a

decision tree to any considerable extent and does not require scaling of data as well. But if it is not good when calculaton is more complex compared to other algorithms, an it is often involves higher time to train the model (Dihiraj K,2019), after selecting the best algorithm (SVM) for classification model.

In this research we can answer a NFR attributes are checked from any SRS document that is complete especially the thesis work scope includeavailability, privacy, performance, security, reliability and usability. This quality attribute is selected based on the necessity of helath related systems. After identifiying this quality attributes in our experiment all the attributes have equal value or it should divide the 100% into six equl parts, based on using rule based experiment if the random value is accepted from the user then based on the testing result from the classification model and the user level testing is on the test result of the classification model the experiment shows it is complete SRS, if it misses one quality attribute from six attributes, the experiment shows 83% comeplete, if it misses two from the six, the experiment shows 66% complete,if it misses Three from the six then the experiment shows 50% complete and if it misses four, five and six the experiment shows Incompelet SRS.

.

Chapter Five :Conclusion and Recommendation

5.1 Conclusion

Non-functional requirement completeness is a critical part of software development, nonetheless,the attention given is less. As we mentioned in previous chapters missing non-functional requirements caused a huge problem, and caused billions of money loss.In this study,benchmark datasets were used to train several different machine learning algorithms. The dataset contains 1169 rows of non-functional requirement text descriptions with their label. Based on the 80-20% train/test partitioning the datasets were partitioned.

To train the collected dataset, three machine learning classification algorithms were used and compared to determine the best one. To do this, several experiments were carried out, and the most recent version of the Python programming language was used to implement the support vector machine,Decision tree , and k nearest neighbour Algorithms.

The data processing phases have been completed after the collection of the dataset, and 1169 non-functional documents have been used for research and model development. To deal with the dataset's class imbalance, SMOTE oversampling techniques were used which is explained in chapter three.

Finally, based on the experimentation using several types of supervised machine learning the built models were compared and the promising result is achieved to measure the completeness of non-functional requirement is 92% via support vector machine(SVM). Since the data used for the experimentation is small the performance of the model could increase by providing enough amount of dataset. After selecting best algorithm this research concludes that based on accepting the NFR requirement from the user then the experiment showed that if the testing result from the classification model and the users testing result should aligned then it is complete NFR unless incomepete NFR.

5.2 Contribution of the study

Various machine learning algorithms are applied and compared in this study to determine the best one to predict the completeness of non-functional requirements. This study's conclusions and results assist software engineers in determining whether or not the non-functional requirement is complete. The following are the key contributions of this study:

i. Pre-processing the benchmark dataset and prepare for experimentation.
ii. Building a machine learning model for measuring the completeness of non-functional requirements.

5.3 Recommendation

Based on the result and finding of this study the following recommendation was made

i. The research can be further improved by developing software tools to easily measure the completeness of non-functional requirements.Therefore, the machine learning model could be integrated with applications, web-site, or other platforms which ease the work required.
ii. Other than machine learning algorithms, applying and experimenting with other approaches with sufficient data could help measure the completeness of non-functional requirements.

6. **References**

Alencar, G. A., De Oliveira, F. V. S., Da Silva Correia-Neto, J., & Teixeira, M. M. (2019). Non-functional requirements in health information systems: A systematic mapping research. *Iberian Conference on Information Systems and Technologies, CISTI*, *2019-June*(June), 1–5. https://doi.org/10.23919/CISTI.2019.8760720

Avdeenko, T., & Pustovalova, N. (2015). The ontology-based approach supports the completeness and consistency of the requirements specification. *2015 International Siberian Conference on Control and Communications, SIBCON 2015 - Proceedings*. https://doi.org/10.1109/SIBCON.2015.7147184

Bhattacharyya, K., & Lal, R. (2020). Classification of Leaf Disease using Image Processing and Machine Learning. *International Research Journal of Innovations in Engineering and Technology*, *04*(12), 06–12. https://doi.org/10.47001/irjiet/2020.412002

Binkhonain, M., & Zhao, L. (2019a). A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X*, *1*. https://doi.org/10.1016/j.eswax.2019.100001

Cheng, J., Greiner, R., Kelly, J., Bell, D., & Liu, W. (2002). Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, *137*(1–2), 43–90. https://doi.org/10.1016/S0004-3702(02)00191-1

Cutler, J., & Dickenson, M. (2020). *Introduction to Machine Learning with Python*. https://doi.org/10.1007/978-3-030-36826-5_10

Canedo, Mendes, E. D., & Cordeiro, B. (2020). *Software Requirements Classification Using Machine Learning Algorithms.* Entropy, 22(9), 1057.

Carson, R. S., Aslaksen, E., Caple, G., Davies, P., Gonzales, R., Kohl, R., & Sahraoui, A.-E.-K. (2004). 5.1.3 Requirements Completeness. *INCOSE International Symposium*, *14*(1), 930–944. https://doi.org/10.1002/j.2334-5837.2004.tb00546.x

Chazette, L. &. (2020). Explainability is a non-functional requirement: challenges and recommendations. Requirements Engineering. *25(4)*, 493–514. doi:https://doi.org/10.1007/s00766-020-003331

Cleland-Huang, J., Settimi, R., & Solc, X. Z. (2007). Automated classification of non-functional requirements. *Springer*, 103–120.

DeVries, B., & Cheng, B. H. C. (2016). Automatic detection of incomplete requirements via symbolic analysis. *Proceedings - 19th ACM/IEEE International Conference on Model-Driven Engineering Languages and Systems, MODELS 2016*, 385–395. https://doi.org/10.1145/2976767.2976791

Getachew, S., & Kekeba, K. (2020). Automated Amharic Hate Speech Posts and Comments Detection Model Using Recurrent Neural Network. *Journal of Big Data*. doi: 10.21203/rs.3.rs-114533/v1

Fenner, M. E. (1392). Machine Learning with Python for Everyone. In *Addison-Wesley* (Vol. 4, Issue 3). Addison-Wesley. http://marefateadyan.nashriyat.ir/node/150

Ferrari, A., Dell'Orletta, F., Spagnolo, G. O., & Gnesi, S. (2014). Measuring and improving the

completeness of natural language requirements. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8396 LNCS*, 23–38. https://doi.org/10.1007/978-3-319-05843-6_3

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pp. 1–84.

Irfan, M. T., & Gudivada, V. N. (2016). Cognitive Computing Applications in Education and Learning. *Handbook of Statistics*, *35*, 283–300. https://doi.org/10.1016/bs.host.2016.07.008

Horkoff, J. (2019). Non-functional requirements for machine learning: Challenges and new directions. *Proceedings of the IEEE International Conference on Requirements Engineering*, *2019-Septe*, 386–391. https://doi.org/10.1109/RE.2019.00050

Kaur, H., Ahamad, S., & Verma, G. N. (2015). A Case Study upon Non-functional Requirements of Online. *International Journal of Computer Applications Technology and Research*, *4*(4), 220–225. https://doi.org/10.7753/ijcatr0404.1002

Kahsay, M. (2019). *Classification of Wheat Leaf Septoria Disease Using Image Processing and Disease Using Image Processing and* (Issue June) [Addis Ababa Science and Technology University]. https://nadre.ethernet.edu.et/record/3788/export/dcite4

Kar, P., & Bailey, M. (1996). Requirements Management Working Group: Characteristics of Good Requirements. *INCOSE International Symposium*, *6*(1), 1225–1233. https://doi.org/10.1002/j.2334-5837.1996.tb02142.x

Kaur, S., Babbar, G., & Gagandeep. (2019). Image processing and classification, a method for plant disease detection. *International Journal of Innovative Technology and Exploring Engineering*, *8*(9 Special Issue), 868–871. https://doi.org/10.35940/ijitee.I1139.0789S19

Kühl, N., Goutier, M., Hirt, R., & Satzger, G. (2020). Machine learning in artificial intelligence: Towards a common understanding. *ArXiv*. https://doi.org/10.24251/hicss.2019.630

Manolomon/tera-promise. GitHub. (2021). Retrieved 8 April 2021, from https://github.com/Manolomon/tera-promise/find/master.

Matsumoto, Y., Shirai, S., & Ohnishi, A. (2017). A Method for Verifying Non-Functional Requirements. *Procedia Computer Science*, *112*, 157–166. https://doi.org/10.1016/j.procs.2017.08.006

Online, L. O. (2020, December 20). *Python Machine Learning Cookbook*. Retrieved from https://www.oreilly.com/library/view/python-machine-learning/9781786464477/ch06s02.html

Pai, & A., W. K. (2020, December 20). *What is Tokenization | Tokenization In NLP*. Retrieved from Analytics Vidhya: Available: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization NLP/#:~:text=Tokenization%20is%20a%20way%20of,n%2Dgram%20characters)%20tokenization.

Pohl, K., & Rupp, C. (2011). *Requirements Engineering Fundamentals.* Sevastopol, CA, USA: O'Reilly Media, Inc.

Pohl, K., Böckle, G., & Linden, F. v. (2005). Software product line engineering: foundations, principles, and techniques. *Springer*.

Rayson, P. G. (1999). Recovering legacy requirements. *In: Proc. Of REFSQ 1999*, pp. 49–54.

Russell, R. (2018). *Machine learning step-by-step guide to implementing machine learning algorithms with Python*. 106. http://booksdescr.org/item/index.php?md5=D161EE832B8007A058CD006DD67E388E

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

Swamynathan, M. (2019). Mastering Machine Learning with Python in Six Steps. In *Mastering Machine Learning with Python in Six Steps*. https://doi.org/10.1007/978-1-4842-4947-5

Sarkar, & D. (2016). *Text Analytics with Python.* New York, NY, USA: Adventure Works press.

Tesfa, T. (2020). *Analysis Of Road Traffic Accidents To Identify Major Causes Of Accidents Using Machine Learning Techniques : In The Case* of Addis Ababa(Issue November). Addis Ababa Science and Technology University.

Tyagi, V. (2018). Understanding Digital Image Processing. In *Taylor & Francis Group;CRC*. Taylor & Francis Group;CRC. https://doi.org/10.1201/9781315123905

Vemuri, V. K. (2020). The Hundred-Page Machine Learning Book. *Journal of Information Technology Case and Application Research*, *22*(2), 136–138. https://doi.org/10.1080/15228053.2020.1766224

Wiegers, K. (2006). More About Software Requirements: Thorny Issues and Practical Advice. *Microsoft Press*, 224.

Train-Test Split for Evaluating Machine Learning Algorithms. (n.d.). Retrieved April 13, 2021, from https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/

Appendix A sample snapshot code

Snapshot for packages used in the experiment

```
In [54]: import numpy as np
         import pandas as pd
         import re
         import nltk

         import sklearn
         from sklearn import preprocessing
         label_encoder = preprocessing.LabelEncoder()
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import classification_report
         from tensorflow.keras import preprocessing
         from tensorflow.keras.preprocessing.text import Tokenizer
         from tensorflow.keras.preprocessing.sequence import pad_sequences



         from sklearn.naive_bayes import MultinomialNB
         from sklearn.svm import SVC
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
```

Snapshot for reading the training and testing dataset

```
In [56]: data = pd.read_csv("E:/MSC DATA/thesis/de/Nfdataset.csv")
         tdata = pd.read_csv("E:/MSC DATA/thesis/de/testfinaldx.csv")

         #new_test_data = pd.read_csv("E:/MSC DATA/thesis/newtestdata.csv",
         data
```

Out[56]:

	RequirementText	class
0	'The system shall refresh the display every 60...	1
1	'The product shall respond fast to keep up-to-...	1
2	'The product shall produce search results in a...	1
3	'The search results shall be returned no later...	1
4	'The product shall generate a CMA report in an...	1
...	...	...
1164	iii A person subject to the jurisdiction of th...	6
1165	iii An individual's access to protected health...	6
1166	iii Not disclose and may not permit a health i...	6
1167	iii The group health plan, or a health insuran...	6
1168	iii The name or other specific identification ...	6

1169 rows × 2 columns

Snapshot for stemming the testing and training dataset

In [64]:
```
# Stemming
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
STOPWORDS = set(stopwords.words('english'))
from string import punctuation
def get_stemmed_text(corpus):
    text=corpus['RequirementText']
    stemmer = PorterStemmer()

    stem_sentence=[]
    for word in text.split():
        if word not in STOPWORDS and word not in punctuation:
            stem_sentence.append(stemmer.stem(word))
            stem_sentence.append(" ") #adding a space so that we can join all
    return " ".join(stem_sentence)
```

In [65]:
```
data['RequirementText'] = data.apply(get_stemmed_text, axis=1)
tdata['RequirementText'] = tdata.apply(get_stemmed_text, axis=1)
```

```
In [66]: data['RequirementText']
```

```
Out[66]: 0       system   shall   refresh   display   everi   s...
         1       product   shall   respond   fast   keep   data...
         2       product   shall   produc   search   result   a...
         3       search   result   shall   return   later   thi...
         4       product   shall   gener   cma   report   accep...
                                      ...
         1164    iii   person   subject   jurisdict   food   dr...
         1165    iii   individu   access   protect   health   i...
         1166    iii   disclos   may   permit   health   insur ...
         1167    iii   group   health   plan   health   insur  ...
         1168    iii   name   specif   identif   person   class...
         Name: RequirementText, Length: 1169, dtype: object
```

```
In [67]: tdata['RequirementText']
```

```
Out[67]: 0       recycl   part   audit   report   shall   retur...
         1       respons   time   common   request   reach   us...
         2       respons   time   gener   student   manag   tas...
         3       respons   time   product   interfac   exceed  ...
         4       respons   time   shall   fast   enough   maint...
                                      ...
         223     iii   disclos   may   permit   health   insur ...
         224     iii   group   health   plan   health   insur  ...
         225     iii   name   specif   identif   person   class...
         226     iii   extent   disclosur   expressli   author ...
         227     iv   descript   purpos   request   use   discl...
         Name: RequirementText, Length: 228, dtype: object
```

Snapshot for checking the NFR is complete or incomplete

```
In [118]: import numpy
          arr = numpy.array(sampled)
```

```
In [119]: dataln=len(tedata)
          dataln
```

```
Out[119]: 30
```

```
In [120]: Dict = {1: 'Performance', 2: 'Security', 3: 'usability',4: 'reliability', 5: 'Av
          print("\nDictionary with the use of Integer Keys: ")
          print(Dict)

          # accessing a element using key
          print("Accessing a element using key:")


          yhat = knn_classifier.predict(tedata)
          yhat
```

```
Dictionary with the use of Integer Keys:
{1: 'Performance', 2: 'Security', 3: 'usability', 4: 'reliability', 5: 'Availib
ility', 6: 'Privacy'}
Accessing a element using key:
```

```
Out[120]: array([6, 1, 3, 2, 1, 6, 2, 6, 6, 1, 3, 2, 1, 2, 2, 2, 2, 6, 2, 6, 2, 2,
                 1, 1, 3, 1, 2, 2, 1, 5], dtype=int64)
```

In [121]:

```
for i in range(100):
    print(tedata[i],Dict[yhat[i]])
```

```
[  6   1  62  74   4   9 192 193 194 195  74   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
   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] Performance
[196   1 197 198   5  64   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   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] Performance
[199   1 200 201 202 203 204   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   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] usability
[ 23   5  75 205  19 206   1  76  77 207  14  23   5  78  75   1  76  77
 208  14   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

In [160]:

```
uniquevalue = np.unique(yhat)
uniquevalue
```

Out[160]:

```
array([1, 2, 3, 5, 6], dtype=int64)
```

In [163]:

```
total=len(uniquevalue)
if(total==6):
    print("complete")
elif(total==5):
    print("83 % complete")
elif(total==4):
    print("66 % complete")
elif(total==3):
    print("50 % complete")
else:
    print("in complete")
```

```
83 % complete
```