8-2022

SecREP : A Framework for Automating the Extraction and Prioritization of Security Requirements Using Machine Learning and NLP Techniques

Shada Khanneh

Abstract

Gathering and extracting security requirements adequately requires extensive effort, experience, and time, as large amounts of data need to be analyzed. While many manual and academic approaches have been developed to tackle the discipline of Security Requirements Engineering (SRE), a need still exists for automating the SRE process. This need stems mainly from the difficult, error-prone, and time-consuming nature of traditional and manual frameworks. Machine learning techniques have been widely used to facilitate and automate the extraction of useful information from software requirements documents and artifacts. Such approaches can be utilized to yield beneficial results in automating the process of extracting and eliciting security requirements. However, the extraction of security requirements alone leaves software engineers with yet another tedious task of prioritizing the most critical security requirements. The competitive and fast-paced nature of software development, in addition to resource constraints make the process of security requirements prioritization crucial for software engineers to make educated decisions in risk-analysis and trade-off analysis.

To that end, this thesis presents an automated framework/pipeline for extracting and prioritizing security requirements. The proposed framework, called the *Security Requirements Extraction and Prioritization Framework (SecREP)* consists of two parts:

- SecREP Part 1: Proposes a machine learning approach for identifying/extracting security requirements from natural language software requirements artifacts (e.g., the Software Requirement Specification document, known as the SRS documents)
- SecREP Part 2: Proposes a scheme for prioritizing the security requirements identified in the previous step.

For the first part of the SecREP framework, three machine learning models (SVM, Naive Bayes, and Random Forest) were trained using an enhanced dataset the “SecREP Dataset” that was created as a result of this work. Each model was validated using resampling (80% of for training and 20% for validation) and 5-folds cross validation techniques. For the second part of the SecREP framework, a prioritization scheme was established with the aid of NLP techniques. The proposed prioritization scheme analyzes each security requirement using Part-of-speech (POS) and Named Entity Recognition methods to extract assets, security attributes, and threats from the security requirement. Additionally, using a text similarity method, each security requirement is compared to a super-sentence that was defined based on the STRIDE threat model. This prioritization scheme was applied to the extracted list of security requirements obtained from the case study in part one, and the priority score for each requirement was calculated and showcased.

Keywords-machine learning, natural language processing, text classification, software security, security requirements engineering, security requirement prioritization.

MONTCLAIR STATE UNIVERSITY

SecREP: A Framework for Automating the Extraction and Prioritization of Security Requirements Using Machine Learning and NLP Techniques

by

Shada Khanneh

A Master's Thesis Submitted to the Faculty of

Montclair State University

In Partial Fulfillment of the Requirements

For the Degree of

Master of Science

August 2022

College of Science and Mathematics
Department of Computer Science

Thesis Committee:

Dr. Vaibhav Anu
Thesis Sponsor
[redacted] (08/05/22)

Dr. Aparna Varde
Committee Member
[redacted] (Aug 5, 2022)

Dr. Kazi Zakia Sultana
Committee Member
[redacted] 08/05/22

SecREP: A FRAMEWORK FOR AUTOMATING THE EXTRACTION AND PRIORITIZATION OF SECURITY REQUIREMENTS USING MACHINE LEARNING AND NLP TECHNIQUES

A THESIS

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

by

Shada Khanneh

Montclair State University

Montclair, NJ

2022

Acknowledgements

Thanks to:

- Dr. Vaibhav Anu for his continuous counseling, guidance, support, and kind encouraging words throughout the entire year working on this thesis.
- Dr. Aparna Varde for her interactive, interesting, and fulfilling classes, where my passion for machine learning and algorithms were ignited.
- Dr. Kazi Zakia Sultana for her diligence in teaching secure programming, where in her class I discovered my interest in cybersecurity.
- To my Parents Mahmoud Khanneh and Sadika Alotti who always valued education and planted the commitment to knowledge and learning in me. And for the unconditional love!
- To my sister Dr. Enas Khanneh whom without her continuous love and support I wouldn't be where I'm today.
- For my friends Peter Farid, Roy Rosario, Syed Arif Shahjalal, and Jonathan Carlin for their faith in me, positive thoughts, and for always being there for me when I needed them the most!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: Introduction

Security in the context of software systems is an ever-growing concern especially with the degree of ubiquity and availability the world is witnessing in software driven services, networking, and shared resources. With that in consideration, security presents itself as an integral part in implementing a successful and satisfactory software that incorporates the necessary measures for protecting the stakeholders' assets [37].

Security concerns must be addressed and accounted for in the early stages of the software development lifecycle (SDLC) to prevent security risks, exploits, and financial loss [2,13,28,33,36,38,51,55]. The high costs incurred by organizations due to poor security requirement engineering (SRE) show that there would be a high value to even a small improvement in this area. By the time that an application is deployed in its operational environment, it is very difficult and expensive to significantly improve its security [36]. Bearing in mind these elements, security requirements engineering and elicitation, was granted a specific area of knowledge in the information technology literature.

Security Requirements Engineering (SRE) is an activity conducted during the early stage of the software development process. SRE involves eliciting, analyzing, and documenting security requirements. However, SRE is still a new area of knowledge that requires considerable efforts and expertise for identifying complex security requirements. Additionally, for most software systems used and sold on commercial levels, the system requirements specification (SRS) document is normally of a large size and intertwined nature. This could result in neglecting or incorrectly identifying valuable security requirements [1,46].

The elicitation and identification of security requirements alone, leaves software engineers with yet another tedious task of prioritizing the most critical security requirements. Even though one might think that all security requirements are considered relevant, implementing all security mechanisms that protect the software against every possible threat is not feasible and almost impossible. The competitive and fast-paced nature of software development, in addition to resource constraints makes the process of security requirements prioritization crucial for software engineers to make educated decisions in risk-analysis and trade-off analysis and guarantees that at least the topmost security measures are accounted for and implemented, especially in the software's early releases [8,15,17,38,41,57,58].

Machine learning techniques have been widely used to facilitate and automate strenuous and complex tasks [53]. Machine learning refers to the area of knowledge that falls under the realm of artificial intelligence (AI). Where an algorithm is fed a series of data to either transform, restructure, extract information, find similarities, or predictions from that said data, using predefined sets of rules and/or mathematical equations. This can be used in a variety of applications such as, image recognition, weather prediction, fraud detection, information extraction, etc. Under the concept of machine learning falls yet another sub area of knowledge, that is Natural language processing (NLP). NLP offer in depth techniques and focus on processing natural language raw text artifacts into desired useful information.

The opportunity that presents itself here is to introduce a machine learning process to automatically identify the security requirements from a software system's requirements specifications artifact (these artifacts generally tend to be 100s of pages in length, depending on the size of the software being developed). Additionally, a prioritization process can be added that

provides a reliable scheme for the identified security requirements in a single automated pipeline. Such pipeline would offer great value, save time, and labor, and potentially improve the efficiency and the accuracy of an otherwise manual process.

1.1 Goal

The overarching goal of this work is to establish a refined process where security is adequately incorporated in the early stages of the SDLC, namely the requirements elicitation and specification stage. This will ensure that the security risks to the software and any future correction measures can be significantly minimized.

Thus, the overall goal of this thesis is stated as follows:

"To automate the identification/extraction and prioritization of security requirements from natural language software requirements artifacts"

1.2 Related Work

The literature as it stands offers abundant work in security requirements engineering, as well as prioritization. Additionally, many scholars tackled the uses of machine learning approaches to address the extraction, elicitation, and prioritization of software requirements. However, most of the previous work is mostly focused on the general concept of automating the extraction of software requirements, without addressing in depth the specificity of security requirements extraction and prioritization. This section summarizes relevant work addressing either the identification of security requirements using machine learning approaches, or the issue of prioritizing security requirements.

1.2.1 Machine learning for security requirements classification

In the context of utilizing machine learning techniques for extracting security requirements form natural language artifacts Riaz et al. [46] introduced a framework that takes as input natural language artifacts documents. The process then examines each sentence in the document to determine whether it is a security relevant sentence and then classify it according to the security objectives, that are either explicitly stated or implied by that sentence. The authors used and compared the results of a k-NN classifier, and from Weka[20], a multinomial naïve Bayes classifier, and a SMO (sequential minimal optimization classifier). As for the training data the authors selected six different documents form the health care domain, and with the aid of three experts classified 10,963 sentences and extracted corresponding security objectives. The manual analysis showed that 46% of the sentences were security relevant. Of these, 28% explicitly mention security while 72% of the sentences are functional requirements with security implications. The proposed tool predicted and classified 82% of the security objectives for all the sentences (precision) and identified 79% of all security objectives implied by the sentences within the documents (recall). Finally based on an analysis they conducted; the authors develop context-specific templates that can be instantiated into a set of functional security requirements by filling in key information from security relevant sentences.

Kobilica et al.[29] work was more focused on examining to what extent shallow machine learning classifiers, with basic pre-processing technique, can achieve in terms of accuracy. The authors conducted multiple experiments to examine twenty-two different machine learning approaches. Th training was conducted using the SecReq [59] dataset and the results showcased that ensemble techniques such as LSTM Ensemble Boosted Trees and CNN gave the best accuracy performance. requirements. Despite that fact that the Long Short-Term Memory

(LSTM) reached the highest accuracy level, the authors recommend the use of supervised machine learning classification approaches in the context of security requirement. Since deep learning approaches such as the LSTM exhibit longer training time that might not be suitable for instant and fast classification.

In another paper published by Kurtanovic et al.[32] on the use of supervised learning machine learning to classify requirements as either functional (FR) or non-functional (NFR). This classification was conducted on the "Quality attributes (NFR)" dataset. Furthermore, the non-functional requirements were identified as either usability, security, operational, or performance requirements. The process the authors adopted in this work consisted of first identifying whether a security requirement is a functional or a non-functional requirement. In doing so the most informative features in the text were also identified and the top ten were selected to conclude the classification. For the non-functional requirements further classification, the authors filtered out the functional requirements from the dataset and then assessed four binary classifiers for identifying the four most frequent NFRs in the dataset: usability, security, operational, and performance. Then a multi-class classifier was used for predicting these four classes.

Abad et al. [1] work tackles classifying requirements into functional requirements (FR) and non-functional ones (NFR), and how automated classification of requirements into FR and NFR can be improved. The proposed approach introduced a preprocessing solution that standardizes and normalizes requirements before applying the classification algorithms. By leveraging rich sentence features and latent co-occurrence relations, the authors showcased improved results in precision when the classification was applied to the processed dataset using

the proposed technique. In addition to this work the authors also conducted an analysis to showcase the performance of several existing machine learning methods that are used for the automatic classification. This study was conducted on 625 requirements provided by the OpenScience tera-PROMISE repository.

Dave et al. [10] also used the PROMISE repository dataset of functional and non-functional requirements, to conduct their experiments. The main premises of this work was to automate the classification of software requirements into FN and NF requirements, and to further classify the NF requirements into nine different categories. Using three machine learning algorithms, SVM, Stochastic Gradient Descent (SGD), and Random Forests, the authors conducted a comparison of these machine learning performances. Where the results showed that, SVM with TF-IDF produced the best results when classifying FRs, the SGD with TF-IDF produced the best results for NFRs, and in the case of both FRs and NFRs, the 3 models produced quite similar results.

Another consideration regarding the elicitation of software requirements was brought to light by Emebo et al.[11,44] work on identifying the Implicit Requirements (IMRs) of a said system. IMRs are assumed needs that a system is expected to fulfill though not elicited during requirements gathering. The authors highlight that addressing such requirements is crucial for the completeness and success of the overall system. Especially that, studies have shown that a major factor in the failure of software systems is the presence of unhandled IMRs. To address this issue the authors, propose a novel framework called the COTIR (Commonsense knowledge, Ontology and Text mining for Implicit Requirements) for identifying and managing IMRs based on combining three core technologies: common sense knowledge, text mining and ontology.

Additionally, and building on the COTIR tool, Emebo et al.[43] conducted another work which incorporates CNN-based deep learning to further enhance the COTIR detection of IMRs from complex SRS big data such as images and tables. This approach aimed to deploy a Convolutional Neural Network (CNN) for further analysis of IMRs. This enhanced COTIR framework ability to extract IMRs was evaluated using three SRS documents and compared against eight software engineering experts (SE researchers and IT professionals). Where, the eight study participants first carefully read each of the SRS documents supplied to them and identified requirements with implicit patterns. As a result of this manual effort, all participants collectively were able to identify 8 potential IMRs. The COTIR tool was then used to identify the following types of implicit patterns in the SRS documents: i) Ambiguity, ii) Incomplete Knowledge, iii) Vagueness, and iv) Miscellaneous. The experts' evaluation served as the ground truth. A major result of this study was that the COTIR approach was able to identify 6 out of 8 known instances of implicit patterns in the supplied requirements. The authors conclude that, such results provide evidence that COTIR can relieve human analysts from the tedious manual task of reading huge SRS documents to find IMRs.

What is worth mentioning here is that several approaches and studies were conducted to classify natural text as functional requirements (FR) or non-functional requirements, and/or the NFR were classified as security, performance, usability. However, there is clear shortage in addressing the classification of security requirements as a specific principle of their own. Security requirements intertwine with all categories of requirements, and many security requirements can be expressed as functional requirements, performance, availability, and usability. Thus, security requirements must be extracted from functional requirements and non-functional requirements alike to achieve the completeness of the security requirements elicitation

process and aid the shift of how to address security concerns and issue in software systems. With that, this work will retain from any pre-classification and instead will aim to extract security requirements from any sentence described as a requirement.

1.2.2 Security Requirements Prioritization

The extraction and identification of security requirements appears to be the focus of most of the current literature work when it comes to the use of automation and machine learning technologies. Which draws the attention to the lack of such automated frameworks for the purpose of prioritizing security requirements. However, there are significant work that addresses the issue and suggests feasible process to manually elicit, identify, and prioritize security requirements. Even though such frameworks require experts' skill and significant efforts, they still provide sufficient aid for software engineers. For example, Yoo et al. [58] proposed a rather easy to understand and implement technique for prioritizing security requirements, which the authors called the enhanced misuse-case. The enhanced misuse-case extends upon the well-established use-case diagram. This is perhaps what is most appealing about this approach. The proposed solution addresses the prioritization of security requirements in terms of the number of functional requirements and/or assets each security requirement is addressing. In addition to the risks value that each security requirement is trying to mitigate. The authors suggest calculating this risk value using the Common Vulnerability Scoring System (CVSS).

In another work presented by Carvalho et al. [8] that was mainly concerned with the security issues and incidents associated with smart toys that uses sensors and cloud-based services to collect data. To adequately address these security issues associated with such systems, the authors proposed an approach where they used the Microsoft SDL method to

identify a comprehensive list of security issues based on specific regulations, threats based on surface attack analysis, and security requirements that address security issues and threats. As for prioritization, the authors presented a method based on risk assessment, AHP, and generic scenarios. The suggested process to prioritize security requirements requires to first, identify the severity of all threats addressed by the security requirements using the security bug bar. Second, the severity of all security issues addressed by the security requirements must be defined using standards and regulations that must be considered in the context of a system. Finally, the risk of each security requirement is calculated based on the severity of the threats and the security issues addressed by it. This final value is then used to prioritize the security requirement. In the case of security requirements with the same risk values the authors suggest that using the AHP techniques provides the most reliable results.

Park et al. [45] approach for prioritizing security requirements facilitates the threat modeling model to create a process that allows for the prioritization of security requirements via the valuation of assets, threats, and countermeasures. Modeled in a tree-like structured graph referred to as a "valuation graph". The valuation graph requires a total of eight steps: six steps that must proceed the prioritization scheme to achieve the prioritization, that is manifested in seventh and eighth steps. The suggested prioritization scheme derives its principle from identifying the system's assets, the threats per asset, and their valuation in terms of impact and risk. Once assets and threats and their values are well established, security requirements become the countermeasures that can be used to mitigate and address these threats. The priority of these security requirements is then calculated using the total impact (TI) of threats that a countermeasure mitigates, the gain (G) of each countermeasure, and the value of assets that the countermeasure protects.

1.3 Contributions of this Study

The contributions of this work can be summarized as follows:

1. An enhanced dataset (called the SecREP dataset), which includes samples of requirements classified as either a security requirement or non-security requirement. This enhanced dataset is a combination of the SecReq [59] dataset and the NFR dataset available on "zenodo.org" [52,60], in addition to some requirements gathered from security requirements specifications documents.
2. A comparison between three machine learning models (SVM, Naïve Bayes, and Random Forest) trained using the enhanced dataset.
3. A process that employs a voting ensemble approach for extracting security requirements from software language artifacts (such as SRS documents).
4. A novel semi-automatic prioritization scheme for security requirements. This scheme uses sentence features and relation extraction to identify assets, threats, and security properties. In addition, the scheme uses a sentence similarity technique to identify threats that correspond to STRIDE [30] modeling.

1.4 Thesis Structure

This thesis presents the Security Requirements Extraction and Prioritization Framework (henceforth referred to as SecREP). The SecREP pipeline consists of two major parts: *Part One*, wherein a machine learning based approach for security requirements identification/extraction is proposed and evaluated; *Part two*, wherein a security requirements prioritization scheme is proposed.

The thesis structure is summarized in Figure 1 and a description of the two-step SecREP pipeline is provided below.

Part One: this part represents the process and the steps taken to conduct the security requirements extraction framework. The proposed extraction process was a result of the following iterations:

1. Using the SecReq [59] dataset after manual cleaning, and modifications which yielded a total of approximately 584 requirements.
2. Using the enhanced SecREP dataset, which included the SeqReq [59] dataset, a portion of the NFR dataset [52,60] and manually extracted requirements from SRS documents (total of 752 requirements)
3. For each iteration three machine learning models (SVM, Naïve Bayes, and Random Forest) were trained, after applying preprocessing techniques to the input datasets. The trained model was used to extract security requirements from a real-world system requirements specification artifact (a list of 30 requirements). Using background knowledge and judgment to evaluate the extracted security requirements list. Satisfactory results were achieved in the third iteration using the enhanced SecREP dataset.
4. The performance for each model was evaluated in terms of precision, accuracy, recall, and F1 scores. Using 80% training and 20% testing resampling approach as well as 5-folds-cross validation.
5. The final extracted security requirements list was based on an ensemble decision where all three trained models classified a requirement as a security requirement.

Part Two: the second part of this work represents the process and the steps taken to conduct a prioritization scheme for the security requirements that were extracted from the previous part (part one). The proposed prioritization process was a result of the following steps:

1. The extracted security requirements list from the ensemble extraction process, was treated with preprocessing techniques to be normalize and reshaped.
2. Using spaCy [23] pretrained natural language medium model, feature extraction techniques were applied on the processed security requirements list to identify the assets, threats, and security attributes (such as Authentication, Integrity, Confidentiality…etc.) that appear in the extracted list.
3. For each security requirement a total score was calculated based on the number of assets, threats, and security properties present in the said security requirement.
4. For additional mapping, spaCy [23] Word2Vec text similarity algorithm was utilized to calculate the similarity of each security requirement to a *super-sentence*. Each super-sentence corresponds to a STRIDE [30]threat definition.
5. A weight of 1 was added to each STRIDE [30] threat that appears in the security requirement, based on a similarity of at least 90% to each super-sentence.
6. The total count of STRIDE [30] similarity weight and the total count of assets, threats, and security properties, becomes the priority score of the security requirement.

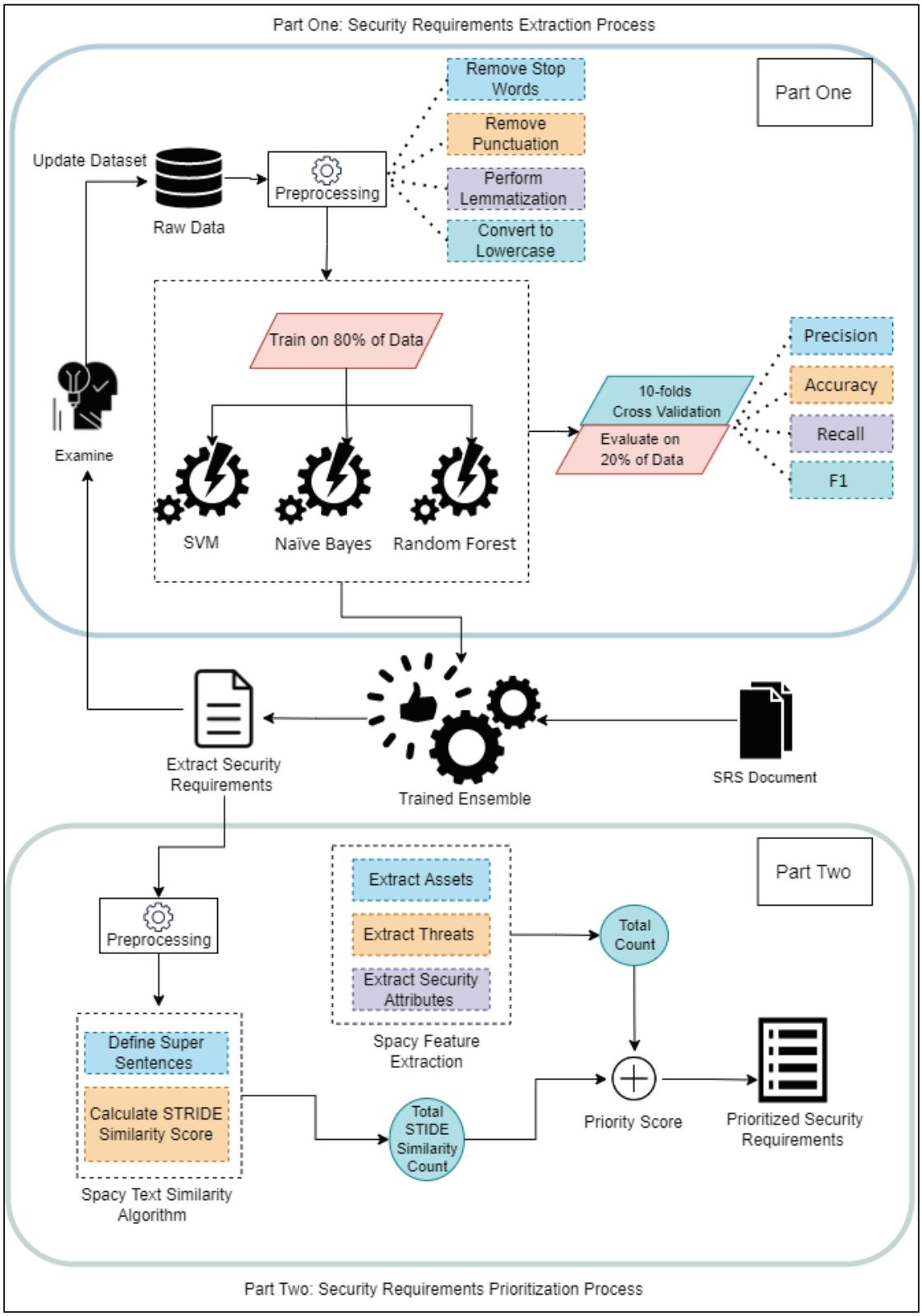

Figure 1: Security Requirements Extraction and Prioritization Pipeline (SecREP)

CHAPTER 2: Background

2.1 Requirements Engineering and Elicitation

Requirements Engineering (RE) is defined as the process of gathering, eliciting, analyzing, documenting, and maintaining requirements in the software development process [42]. Requirements engineering (RE) is an essential step that must proceed the design and development of a software's code [5]. This importance stems from the fact that addressing requirements errors, such as ambiguous, incomplete, or omitted requirements, is more expensive to fix when the software is operational. Due to that its notable that extensive work is done in the early stages of the software development life cycle (SDLC).

An important outcome of the requirements' engineering process is the elicitation of the system's requirements specification (SRS). Where the functional requirements are distinguished from the nonfunctional requirements. Functional requirements are those that can directly relate to an action the system is expected to perform as per the end-user expectations. Nonfunctional requirements on the other hand are more ambiguous to derive and normally refer to those requirements that express a system's constraint or quality, such as performance, reliability, efficiency, and security[21,34] . Table 1 Provides examples of functional and nonfunctional requirements to clarify the differences between the two concepts.

Table 1: Examples of Functional and Nonfunctional Requirements

Requirement Example	Category
The system should allow users to change the color scheme of the website.	Functional requirement: allows for changing colors.
Ability to upload large pictures and videos of up to 5Gb	Nonfunctional requirement: performance, load
Users should receive notifications of new features	Functional requirement: allows for notification action
The system should send emails to the users in less than 3 seconds of adding new content to the website	Nonfunctional requirement: performance, efficiency
The system should allow users to interact with each other.	Functional requirement: allow for communication action
Warning and instructions issued by the system should be, in English without disclosing any sensitive information. Error messages should be colored red. Instruction messages should be colored green.	Nonfunctional requirement: reliability, user experience, security.

2.1.1 Security Requirements Engineering

Security requirements are most categorized under the non-functional requirements of the system. These types of requirements describe the constraint on the system's functions, where

these constraints operationalize one or more security goals [19]. Security requirements are dynamic in nature and evolve as the software is developed. Most researchers classify them as nonfunctional requirements because they do not have a clear criterion for their specification and satisfiability [2]. However, security requirements can be also expressed as functional requirements that describe the system's behaviors to achieve these security goals of a said system.

The process of security requirements engineering provides valuable information that covers the definition of security requirements and the concepts that correlate with it like, assets, objects, threats, and vulnerabilities. Which in turn provokes important questions, such as what assets should be protected? Against whom? How should these assets be protected? And to what extent? It is important to recognize that security requirements do not represent security measures and policies, nor should they describe the underlying mechanisms that implement these security measures. With that it is notable that security requirements are complex, and they integrate with the general concept of requirements. Making the process of eliciting security requirements a challenging task, that must be addressed early in the software's development stages [8,31,54] .

When addressing security requirements, it is difficult to derive conclusive rules that can be used to determine if a requirement is a security requirement or a non-security requirement. The system's needs and nature, the standards and regulations of the operational country and environment, and the software engineers' expatriates, are some of the important variables that can drastically influence the categorization of a software requirement. However, guidelines and standards can still be utilized to tackle the issue of deciding if a requirement is a security requirement. For example, in information security *confidentiality*, *integrity, and availability* are

broadly known as the security “Tried” and they are considered the three most important concepts within information security. Considering these three principles and what they present, could alone provide a sustainable start to define a security requirement, where a requirement could be considered a security requirement if it addresses the protection of these concepts and/or the prevention of the inverse of them. Table 2 Provides some examples of security requirements expressed as functional and nonfunctional requirements.

Table 2: Security Requirements as Functional and Non-Functional Requirements.

Security Requirement as Functional requirements	**Security Requirements as Nonfunctional requirements**
The system must issue emails to users if their data was compromised	The system must protect users’ information and prevent any illegal access to users’ sensitive data.
All secrete data must be encrypted/hashed and must not be stored or transmitted in plain text format.	All critical functionalities must be available and accessible to authorized users.
The system must perform regular scanning to any data entering the operational environment as well as any user input.	The system must guarantee the integrity of users’ assets stored and retrieved within the system environment.

2.1.2 Prioritization of Security Requirements

Despite the significance of security in software applications. Implementing all security measures is expensive in time, cost, and effort, in addition to the performance and availability restrains that it might reflect on the system. Absolute software security is an unrealistic

expectation. Thus, it is important to identify the topmost critical security requirements that must be addressed and accounted for. Which in turn, guarantees that at least the most essential protection measures are incorporated in the software's early release [8].

Ideally security requirements prioritization must be considered as an essential task in the process of requirements engineering. Additionally, the lack of a prioritization process could leave the process of security requirements elicitation ambiguous and incomplete. Leaving the decision of which security measures to be implemented to the expertise and skills of the software engineers performing the elicitation. One might easily argue that without prioritization the establishment of security requirements alone does not utilize them into tangible use. The optimal purpose of security requirements is to support the objectives of a project and its overall quality. Prioritization offers great value for risk-analysis and trade-of-analysis aiding stakeholders and software engineers in making educated decisions. Many factors are at play when attempting to prioritize software requirements more so for security requirements. Unlike clear requirements that present the end-users and stakeholders needs, security requirements tend to be difficult to evaluate and valuate[2,48,54]. However, a common criterion on how to address and guide the process of security requirements prioritization can still be derived and established using the most common factors found in a security requirement. Examples of these factors include, the system and user assets, the security attribute being addressed according to the security tried (confidentiality, integrity, and availability), the threats being addressed, and the risks associated with these threats [8,16,22,26,45,51,58].

2.2 Machine Learning and Natural Language Processing

Machine learning (ML) approaches have become in recent years, a necessary part in numerous commercial and industrial applications and appliances. With the explosion of data gathering and information mining, a new paradigm emerged where the use of ML is the automatic determination for otherwise impossible, complex, and/or manual tasks. Such tasks and applications include robotics, text and image recognition, fraud and anomalies detection, intelligent chat bots, data classifications and predictions in whether, medical diagnosis, and navigation...etc. [7,40].

Machine learning (ML): is the scientific study of algorithms and statistical methods that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. There are five types of machine learning algorithms: supervised, semi-supervised, active learning, reinforcement, and unsupervised learning [7,27,40].

Natural language processing (NLP): is a subset of artificial intelligence and a significant filed machine learning. NLP refers to the process where a computer system is used to understand, parse, and extract human language that is in the form of raw text in often times. NLP is a broader term where several problem areas fall under it, such as text categorization\classification, syntactic parsing, part-of-speech tagging (POS), named entity recognition (NER), coreference resolution, machine translation, and the sub area of natural language understanding (NLU). Natural language understanding (NRU) describes the area of

NLP where the computer systems are trained to extract context, intent, and what is inclined by the text. NLU is commonly used in areas such as, relation extraction, paraphrasing, semantic parsing, sentiment analysis, question and answering, and summarization [4,27].

Figure 2 demonstrate the relation between these three fields (AI, ML and NLP), where is Figure 3 demonstrate the problem areas where NLP and NLU are used in relation to each other.

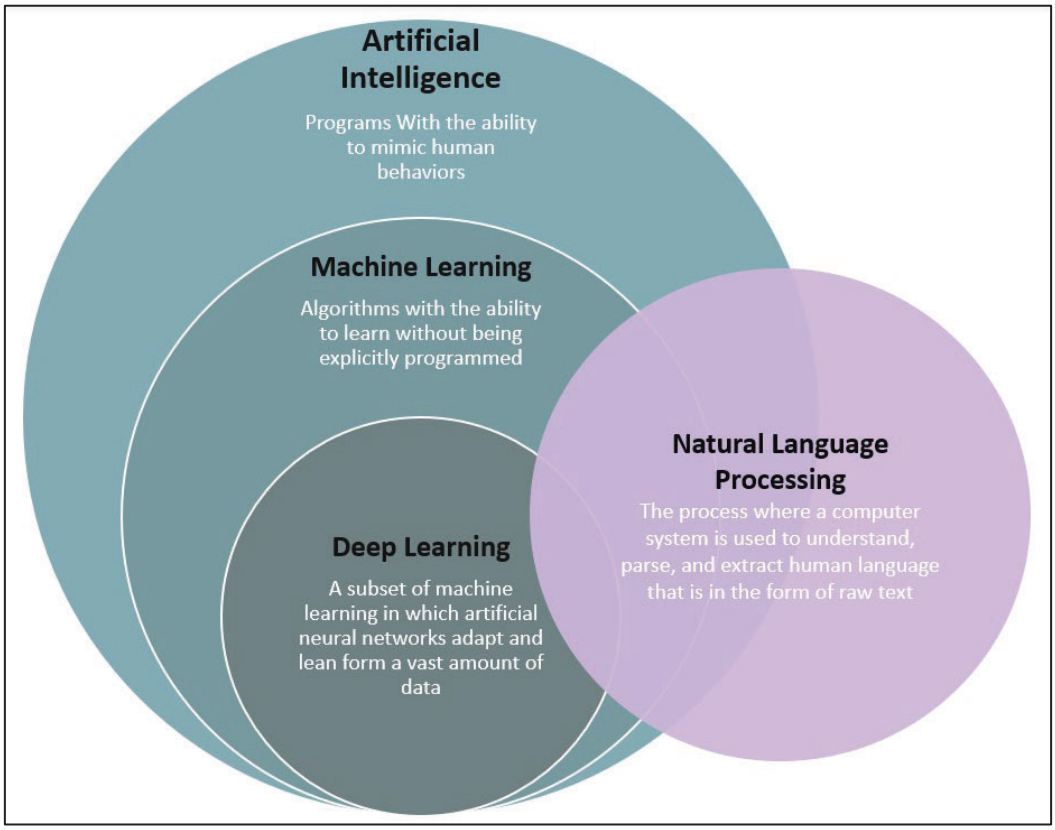

Figure 2: NLP in Relation to AI and ML

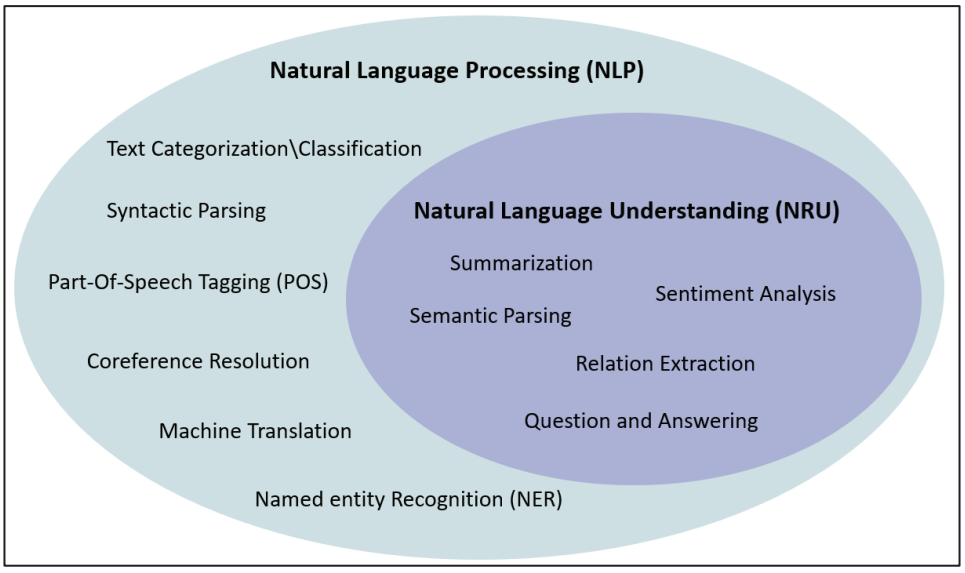

Figure 3: Natural Language Processing (NLP) Knowledge Areas

2.2.1 Information Extraction and Text Classification

Information extraction and classification are both popular applications of NLP. Information extraction focuses on transforming raw text into structured and relevant information to the problem at hand. Whilst text classification is more concerned with labeling and grouping the unstructured data with relation to its content into two or more classes. Text classification can be used on either raw data or on restructured data that is the result of an information extraction process [27,47]. Such pipeline has the potential of producing more accurate and powerful results for many cases, as shown in Figure 4.

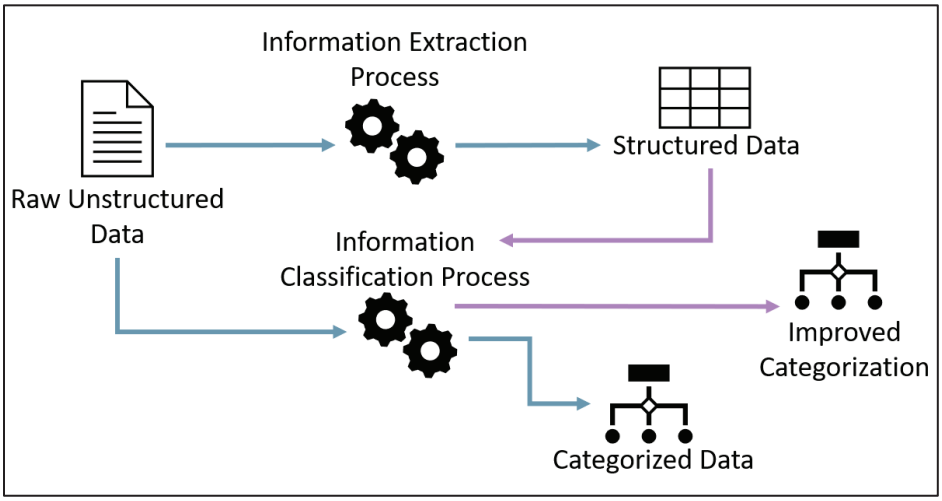

Figure 4: Information Extraction and Classification

2.2.2 Text Preprocessing

In machine learning approaches the most important factor that determines the quality of the process outcome is the condition, completeness, and plausibility of the data that is being fed to the machine for training. Incorrect or poor-quality input will always produce faulty output. More so for NLP techniques, since making a numeric based machines such as computers understand and derive meaning from human spoken languages is a far more difficult task than

processing numeric data. Text preprocessing, describe the cleaning and preparing operations that transforms the raw text, into a well-defined sequence of linguistically meaningful units. Raw text datasets normally contain noisy, duplicated, lengthy, high dimensional, corrupted, and empty data that is not valuable for the NLP task. Which makes preprocessing a vital step that must proceed every NLP task to produce more accurate and reliable results [27,56]. Below is a description for essential text preprocessing techniques that were also used in later chapters in this work.

Manual cleaning: while many automated libraries exist that provide reliable text preprocessing techniques, in some cases manual cleaning and validation of the dataset is still necessary. For example, in some cases where many samples are unlabeled, experts' judgment will be required to provide an accurate classification and prevent losing the sample. In other cases, reducing the sentence length would also be a manual task to ensure that this reduction did not jeopardize the integrity and usefulness of that data sample [27].

General Cleaning: it is important to understand the dataset that will be used for the NLP task. Therefore, it is advised to scan the text, understand its structure, and content. This helps identify what cleaning processes might be needed. Additionally, as a rules of thumb, a general cleaning step that includes removing extra white spaces and line breaks is always a good start to a better dataset. This can be easily done with a simple automated codes and regular expressions.

Fixing Null and NaN values: Data needs to be checked for NaN and Null values. Such values do not only compromise the training process but for many cases they can cause the machine learning algorithm to fail its execution. Null and NaN values can be handled either

manually or automatically after importing the data. In many cases it is recommended to fix the sample where there is a null/Nan value instead of removing it [27]. This fix can be done manually using expert judgment, or automatically using mean values or estimated values that can fill the missing data.

Tokenization: or text segmentation is the process of converting a sentence into a series of words. Each token carries a semantic meaning that is associated with it. The importance of this process is that it breaks down larger pieces of text into smaller more meaningful ones. The tokenization process can be achieved by defining the boundaries for a word and separating them by whitespace and punctuation as well as splitting contractions [4,27].

Stop Words Removal: stop words are very abundant and common in natural language artifacts, and they provide little to no value in terms of analyzing the special meanings of a given text. Hence removing stop words reduces the noise in the given data and promotes better results. Example of such strop words include auxiliary verbs (be, do and have), conjunctions (and, or) and articles (the, a, and an) [4,25,27].

Stemming and Lemmatization: *Stemming* refers to reducing inflected (or sometimes derived) words to their word stem base or root. For example, the words ‘goes’, ’gone’ and ‘going’ will map to ‘go’. *Lemmatization* on the other hand determines the lemma (the infinitive form of verbs and the singular form of nouns and adjectives) of each word. For example, the verbs ("see", "saw", "seeing", "seen"}) will all map to “see”. Both or either techniques can be used depending on the task at hand, to improve the accuracy of the result, and to reduce the overhead of finding text similarity problems. However, lemmatization is often recommended and

is broadly regarded as more useful than stemming, since it adds a morphological analysis to the words [4,12,27].

Punctuation Removal: similar to stop words, punctuations also add extra noise to the data that might affect the tokenization, classification, and extraction processes. However, for some scenarios it might be necessary to keep some punctuations, for example to detect the end of a sentence, extract questions, extract quoted phrases…etc. [12,27].

Convert to Lowercase: converting the text to lowercase is vital for text parsing, similarity extraction, and words vector presentations. For example, the words "Apple" and "apple" will be given different numerical and weight values. While in essence these words serve the same purpose and meaning.

Part of Speech (POS) Tagging: in linguistic features, text is analyzed to extract features related to the interested objective. Part of Speech (POS) tagging involves tagging a word with a part of speech label (such as noun, verb, adjective, etc.) based on the definition and its context within the sentence in which it is found [12,23].

2.2.3 Word vectors

Word vectors or word embeddings are numerical representations of words in multidimensional space through metrices. It is well established that, the working language of computers is numbers. Which represents a challenge for natural language processing and words analysis from the perspective of the computer architecture. With that said, many methods were established to address this limitation where words are converted into corresponding unique numbers the computers can understand and process quickly. This process of encoding documents in a numeric feature space is called feature extraction or, vectorization.

One of those initial methods, is the *bag of words,* where words are stored in a dictionary, each word is that dictionary key and its numerical representation is the value. For example, {“the”: 1, “she”: 3} etc. However, in such method and while the computer can recognize the word, it is unable to identify or derive meaning of it, in terms of how that word functions within a sentence, how it works within a language, and how it relates to other words. Word vectors on the other hand, add dimensionality to the word, where a word is represented by an array of decimal numbers. These dimensions are honed via machine learning models that consider the frequency of that word alongside words across a body of text, in addition to, the appearance of other words in similar contexts. This allows for the computer to determine the syntactical similarity of words numerically. Hence why, vectorization is often referred to as Feature selection. Word vectors approaches use *Matrices* to represent these relationships numerically. To represent these matrices more concisely, models flatten a matrix to a float (decimal number) where the number of dimensions represent the number of floats in the matrix [4,6,27].

Below is a description of two vectorization models that were used in later chapters of this work.

2.2.3.1 Term Frequency—Inverse Document Frequency (TF-IDF)

The TF-IDF approach is very commonly used for vectorizing terms and extracting features based on occurrence. It is used in many search engines, information retrieval, and text mining systems. TF-IDF combines two metrics, the raw frequency value of a term in a particular document (TF), and the inverse of the document frequency for each term (IDF).

The term frequency *TF* of a word *w* in a document can be calculated as follows:

$$TF(w) = \frac{\text{Number of times the word w occurs in a document}}{Total\ number\ of\ words\ in\ the\ document}$$

The inverse of the document frequency *IDF* for each term *w* can be calculated as follows:

$$IDF(w) = \log \frac{\text{Total number of documents}}{\text{Number of documents containing word w}}$$

Finally, the weight of word *w* in document *d* can be calculated as follows:

$$weight(w,d) = TF(w,d)xIDF(w)$$

Hence and according to the above equations, the weight of word *w* in document *d* is the product of the *TF* of word *w* in document *d* and the *IDF* of word *w* across the text corpus. [6,27,35].

2.2.3.2 Word2vec

Word2vec [39] was introduced by Google in 2013 and it was developed by Mikolov et a l. At Google. Later the model was made available as an open source for the community to use and build on [27]. In essence the Word2vec is a model that enables the building of word vectors using contextual information from the neighborhood of a word. For every word whose embedding is developed, it's based on the words around it. Word2Vec is a pre-trained two-layer neural network, that takes as input a text corpus and outputs a set of feature vectors that represent words in that corpus [27,39]. Word2vec models can be trained by two approaches, as follows:

- **Continuous Bag Of Words:** this approach takes as input the context of each and tries to predict the word corresponding to the context. Where the context in this case is the surrounding words.

- **Skip-gram:** the skip-gram predicts the context word using the target word as input. In this approach the target word, (i.e., the word to generate a representation for) is used to predict the context.

Figure 5 showcases a simple representation of the two approaches that can be used to train a Word2Vec model.

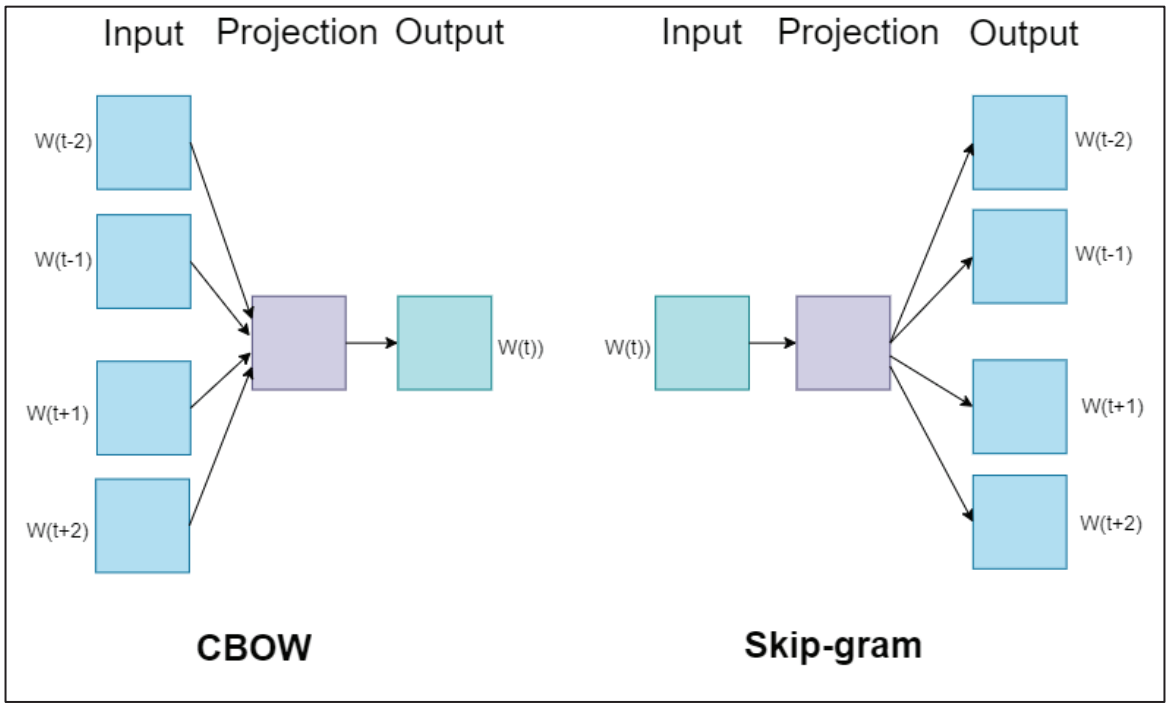

Figure 5: Word2vec Continuous Bag Of Words and Skip-gram Training Methods

2.3 Text Similarity

Text similarity from the perspective of the machine is the distance between two vectors where the vector dimensions represent the features of two objects. Similarity provides insights into the distance between two vectors and measures how different or alike two data objects are. If the distance is small, the objects are said to have a high degree of similarity and vice versa. Once the vector representation of the text, or the text embedding is established, text similarity

becomes a simple distance task that can be calculated using simple well-known mathematical equations, such as the Euclidean Distance and the Cosine similarity [23,27].

2.4 Naïve Bayes

The Naïve Bayes (NB) is a classification technique based on the Bayes' theorem: the basic assumption is that the predictor variables are independent of each other. It is a statistical classifier that performs probabilistic prediction. A simple naïve Bayesian classifier has comparable performance with decision tree and selected neural network classifier.

The Bayes' theorem is mathematically expressed as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where A and B are events and P(B) ≠ 0.

- P(A) and P(B) are the probabilities of observing A and B without regard to each other.
- P(A|B), a conditional probability, is the probability of observing event A give that B is true.
- P(B|A) is the probability of observing event B given that A is true

For a text classification problem, A can be set to the probability that a specific word vector/ words vectors belongs to the targeted class, and B as the entire vectors in the vocabulary set. If P(A|B) > P(¬A|B), then a sentence can be classified accordingly [4,6,12,27,40].

2.5 Support Vector Machines

Support Vector Machines (SVM) is a supervised classification method that works for both linear and nonlinear data. The SVM algorithm aims to find the optimal hyperplane (i.e.,

Decision Boundary) that separates the data points that need to be classified. Naturally, there are many possible hyperplanes that could be chosen. However, the objective is to find a plane that has the maximum distance between data points of both classes (i.e., the margin between the two classes data points).

Hyperplanes: the decision boundaries that separate the data points. Data points falling on either side of the hyperplane can be attributed to different classes. The dimensionality of a hyperplane correlates to the number of features each date point presents. If the number of features is 2, the hyperplane is just a line. If the number of features is 3, then the hyperplane becomes a two-dimensional plane. Figure 6 demonstrates the SVM hyperplanes.

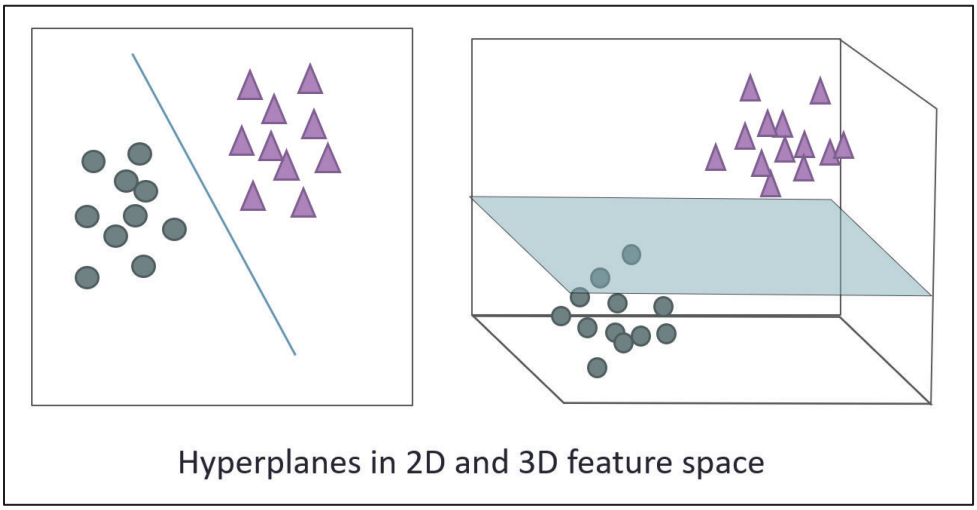

Figure 6: SVM Hyperplanes

Support vectors: are considered as the critical elements of a dataset. These represent the data points closest to the decision boundary. Consequently, if these points are removed, altered, or the data is changed, the position of the dividing hyperplane will be changed and recalculated.

The margin: the distance between the hyperplane and the nearest data point from either class is known as the margin.

Maximum marginal hyperplane (MMH): the hyperplane with the greatest possible margin which yields a greater chance of new data being classified correctly.

Figure 7 demonstrates the SVM support vectors and margins.

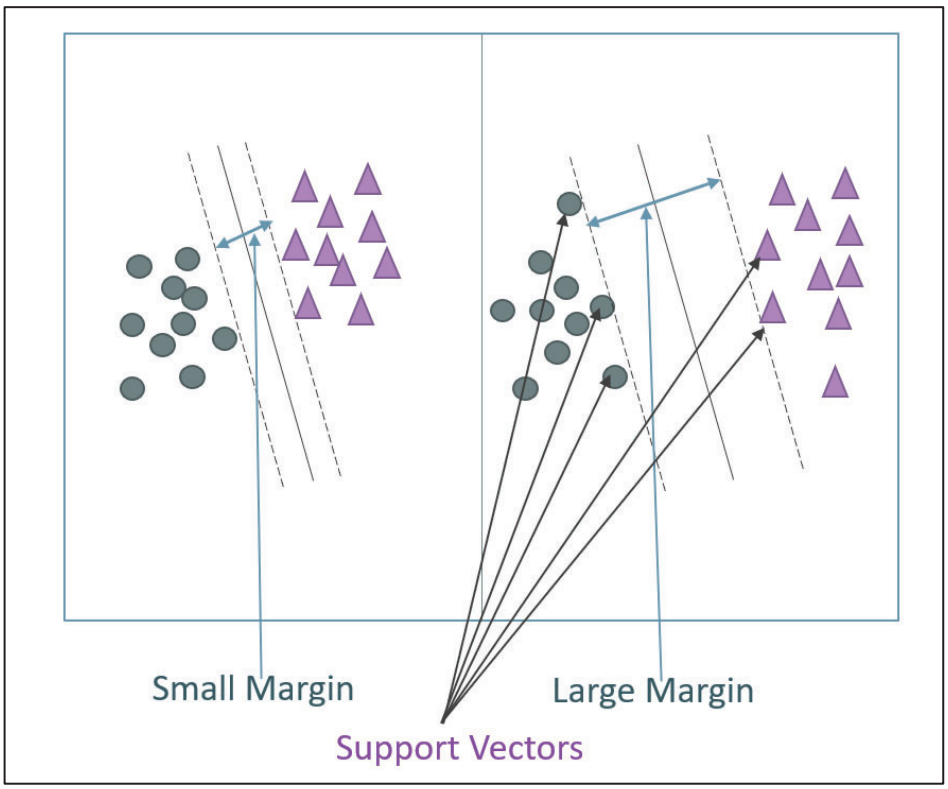

Figure 7: SVM Support Vectors and Margins

Calculating the hyperplane and the margin:

SVM algorithm takes the output of the well-known linear function ($y=mx+b$) and if that output is greater than 1, the point is placed in one class and if the output is -1, it's places with another class. Since the threshold values are 1 and -1 in SVM, this reinforcement range of values ([-1,1]) can be obtained, and act as margin.

The separating hyperplane function is defined as follows:

$$h(x) = w.x + b = \left(\sum_{i=1}^{d} w_i.a_i\right) + b = 0$$

where *W= {w1, w2, ..., wn}* is a weight vector, and b is a scalar called the bias. The margin can then be calculated as:

$$m = \frac{2}{||w||^2}$$

The goal of the SVM as mentioned before, is to determine the weight vector ***w*** and bias ***b*** that maximize the margin *m* to create a distinct hyperplane that satisfies the following constrains:

$$f(x) = \begin{cases} 1 \; if \; w^t x_i + b \geq 1 \\ -1 \; if \; w^t x_i + b \leq -1 \end{cases}$$

Where ***f(x)*** is the decision function used to create a distinct separating hyperplane that classify input data in either positive or negative class [4,6,14,18,27,56].

2.6 Random Forest

Random forest (RF) classifiers fall under the broad umbrella of decision trees and ensemble-based learning methods. Random forests are proven to be a very powerful and successful techniques in pattern recognition and machine learning for high-dimensional classification. As the name suggests, random forests consist of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest produces a class prediction and the class with the majority vote (mode) across them becomes the model's prediction. Among other benefits, this voting approach has the effect of correcting for the undesirable property of decision trees to overfit training data. The reason for this effect is the low

correlation between the models. Where the trees protect each other from their individual errors (assuming the low probability of all trees constantly producing similar errors).

In the training stage, random forests apply a technique known as *bagging* to individual trees in the ensemble. Bagging allows each individual tree to repeatedly select a random sample from the dataset with replacement, resulting in different trees. Each tree is grown without any pruning. The number of trees in the ensemble is a free parameter learned automatically using the out-of-bag error (OOB). The OOB error is the average error for each bootstrap sample calculated using predictions from the trees that do not contain in their respective that bootstrap sample. This allows the random forest classifier to be fit and validated during its training. Hence, the random forest, generates trees that are not only trained on different sets of data (du to bagging) but also use different features to make decisions. Figure 8 demonstrate how a random forest model is split into different decision trees using different features for each sub tree. [3,12,14].

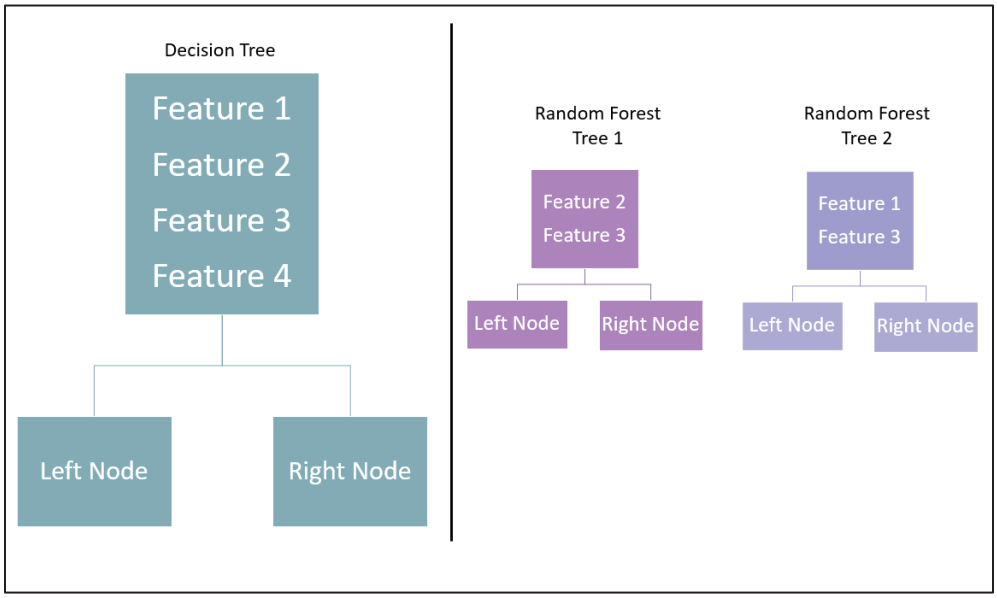

Figure 8: Node splitting in a Random Forest Model

2.7 Performance Measures

Performance measures in machine learning are used to evaluate the correctness and completeness of the trained model performance in a specific classification problem. These performance metrics include accuracy, precision, recall, and F1-score. Such metrics provide tangible insight regarding the strengths and limitations of these trained models, especially when making predictions on new samples.

True Positive (TP): true positive measures how well the model can correctly predict the positive class. Where the model predicts that a data sample is positive, and it is actually positive.

False Positive (FP): false positive is the case when the model wrongly predicts the class of a given data sample. That is, a data ample classified under the negative class, but the model retrieves it as a positive class sample.

True Negative (TN): true negative measures how well the model can correctly predict the negative class. Where the model predicts that a data sample is negative, and it is actually negative.

False Negative (FN): false negative is the case when the model wrongly predicts the class of a given data sample. That is, a data ample classified under the positive class, but the model retrieves it as a negative class sample.

The Precision Score: measures the proportion of positively predicted labels that are actually correct. Precision is also known as the positive predictive value. Precision is used in conjunction with the recall to trade-off false positives and false negatives. Precision is affected by the class distribution. If there are more samples in the minority class, then precision will be lower. The precision score of a model can be calculated using the following equation:

$$Precision\ Score = \frac{\mathrm{TP}}{(\mathrm{FP} + \mathrm{TP})}$$

Where *TP* is the machine learning model true positive measure, and *FP* is its false positive measure.

The Recall Score: represents the model's ability to correctly predict the positives out of actual positives. Unlike precision which measures how many predictions made by models are actually positive out of all positive predictions made. Thus, when calculating the recall score the value of the model's false-negative measure would impact the recall score. The recall score for a model can be calculated using the following equation

$$Recall\ Score = \frac{\mathrm{TP}}{(\mathrm{FN} + \mathrm{TP})}$$

Where TP is the machine learning model true positive measure, and *FN* is its false negative measure.

The Accuracy Score: is described as the ratio of true positives and true negatives to all positive and negative observations. Accuracy provides insight into how often a machine learning model will correctly predict an outcome out of the total number of times it made predictions.

Mathematically, it represents the ratio of the sum of true positive and true negatives out of all the predictions. The accuracy of a machine learning model can be calculated using the following equation:

$$Accuracy\ Score = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{FN} + \mathrm{TN} + \mathrm{FP})}$$

Where *TP* is the machine learning model true positive measure, *FP* is its false positive, *FN* is its false negative measure, and TN is its true negative measure.

The F1-Score: represents the model's score as a function of its precision and recall. The F1-score gives equal weights to the model's Precision and Recall scores to measure its performance in terms of accuracy. Making it an alternative to Accuracy metrics (it doesn't require us to know the total number of observations). It's often used as a single value that provides high-level information about the model's output quality. The F1-Score for a machine learning model can be calculated using the following equation:

$$F1 - Score = \frac{2 * \text{ Precision Score} * \text{Recall Score}}{\text{Precision Score} + \text{Recall Score}}$$

CHAPTER 3: Isolating Security Requirements from Requirements Datasets

The objective of this thesis work is to 1) produce an automatic framework that extracts security related requirements from a software system's natural language artifacts, for example, a Software Requirements Specification document (an SRS). And 2) to produce a prioritization scheme where the extracted list of security requirements can be further refined into a prioritized list. This section represents the design and approach adopted to tackle the first step in the SecREP pipeline showed before in Section 1.4.

3.1 The Datasets

The datasets used in this work's experiments, were mainly the SecReq [59] dataset, and the NFR [52,60] dataset. Both datasets were obtained online from "zenodo.org".

3.1.1 SecReq Dataset

The SecRecq dataset introduced by Houmb et al. [24] Is composed of three industrial SRS documents: Customer Premises Network (CPN), Common Electronic Purse (ePurse), and Global Platform Specification (GPS). In total the dataset contains 510 samples of non-functional requirements labeled as either a security requirement or non-security requirement.

Table 3 provides examples form the SecReq dataset.

Table 3: Requirements Samples from the SecReq Dataset

SRS Document	Requirement Description	Label
CPN	The CNG may support mechanisms supporting nomadism of the users and their subscribed services from one physical customer environment to another.	Non-security
	The diagnostic operations on the CPN by an operator shall be performed in accordance with rules protecting the users' privacy.	Security
ePurse	A single currency cannot occupy more than one slot. The CEP card must not permit a slot to be assigned a currency if another slot in the CEP card has already been assigned to that currency.	Non-security
	Load and unload functions must be authenticated using end-to-end security between the card and the card issuer.	Security
GPS	The Card Issuer is responsible for Working with Application Providers to create and initialize Security Domains other than the Issuer Security Domain.	Non-security
	"The Card Issuer is responsible for Enforcing standards and policies for Application Providers governing all aspects of Applications to be provided to the Card Issuer or operated on the Card Issuer's cards.	Security

3.1.2 The NFR Dataset

The NFR [52,60] dataset that can be accessed on “zenodo.org” is also known as the PROMISE dataset. This dataset can be attributed to Jane Cleland-Huang and was provided for the RE'17 Data Challenge [9]. It was first made available on the PROMISE Software Engineering Repository [52]

The NFR [60] software requirements dataset contains a total of 625 software requirements sentences. All requirements not labeled with “F” are non-functional with the following types: A=Availability, L = Legal, LF = Look and feel, MN = Maintainability, O = Operational, PE = Performance, SC = Scalability, SE = Security, US = Usability, FT = Fault tolerance, and PO = Portability. Table 4 provides examples form the NFR dataset.

Table 4: Requirements Samples from the NFR Dataset

Requirement Description	Label
The system shall display the Events in a graph by time.	F
The product shall be available for use 24 hours per day 365 days per year.	A
The product shall retain user preferences in the event of a failure	FT
The System shall meet all applicable accounting standards. The final version of the System must successfully pass independent audit performed by a certified auditor.	L
The website shall be attractive to all audiences. The website shall appear to be fun, and the colors should be bright and vibrant.	LF
The product must be highly configurable for use with various database management systems for the end users. 80% of end users are able to integrate new database management systems with the product without changing the product's software code.	MN
The product is expected to integrate with multiple database management systems. The product will operate with Oracle SQL Server DB2 MySQL HSQL and MS Access.	O
The search results shall be returned no later 30 seconds after the user has entered the search criteria.	PE
The product is expected to run on Windows CE and Palm operating systems.	PO
system shall be able to handle all the user requests/usage during business hours.	SC
Only authorized users shall have access to clinical site information.	SE
The system shall be used by realtors with no training.	US

3.1.3 The Enhanced Dataset (SecREP)

The SecREP enhanced dataset was the result of multiple iterations experimenting with three different combinations of the SecReq dataset [59] and the NFR dataset [60], in addition to requirements that were handpicked from general SRS documents. In each iteration the dataset was revisited for further enhancement and expansion until satisfactory results were reached.

These alterations made to the datasets were a result of a number of issues that were encountered working with the above datasets.

The issues with the SecReq [59]**dataset are summarized as follows:**

1. The dataset's security requirements samples are all expressed as non-functional requirements. Which would jeopardize the completeness of extracting all security related requirements, since and as explained before, security requirements intertwine with functional and non-functional requirements.
2. The dataset contained a significant number of corrupted data, large sentences, null values, unlabeled samples, and samples labeled as security requirements in a subset of the dataset and non-security in another.
3. Removing these corrupted, unclear, and unlabeled samples will significantly reduce the size of the dataset (from 510 to 375). This, consequently, made the machine learning models predictions considerably poor and unreliable.

The issues with the NFR dataset [60] **are summarized as follows:**

1. Similar to the case with the SecReq dataset [59], all security requirements are classified as non-functional requirements. Additionally, the 11 categories of NFR

could also be expressing a security concern, and thus a considerable amount of these requirements can also be extracted as security requirements.

2. Manually adjusting the classification of the 625 records, to transform the dataset into the shape that servers the classification problem at hand requires multiple experts ruling, taxing time, and considerable effort to ensure the correctness and completeness of such a large-scale problem.

To address these issues, multiple experiments were conducted with different combinations of the available datasets. Each experiment/iteration helped improve the prediction of the trained machine learning models.

First Iteration: using a cleaned version of the SecReq dataset [59] where all corrupted, unlabeled, unclear requirements where removed. Which yielded a total of 375 requirements classified as either security or non-security. After using preprocessing techniques to reshape and vectorize the text. The processed list was fed to the machine learning models namely: SVM, Naive Bayes and RF. Despite the good performance measures scores, all models exhibited poor predictions when subjected to a list of requirements from an SRS document.

Second Iteration: in this iteration using educated judgments, the raw SecReq [59] was manually cleaned, fixed, and reshaped as follows:

1. Corrupted records were rewritten.
2. Unclassified entries were labeled.
3. Some large and unclear requirements samples were split into 2 or more requirements and labeled accordingly.

4. Finally, requirements that were classified as “unknown” were addressed and added to the list. Which yielded a total of 584 requirements classified as either security or non-security.

After using preprocessing techniques to reshape and vectorize the text. Theis processed list was fed to the machine learning models namely: SVM, Naive Bayes and RF. Despite the good performance measures scores, all models exhibited poor predictions when subjected to a list of requirements from an SRS document.

Third Iteration: in this iteration the cleaned 584 requirements extracted from the SecReq dataset [59] were combined with some samples obtained from the NFR [60] dataset.

The sample selection criteria applied on the NFR [60]dataset was as follows:

1. All requirements marked as “LF” (look and feel), and “US” (usability) were added to the list and labeled as non-security requirements. Since it is safe to assume that these types of requirements do not express a direct security concern on the system’s functionality and/or operations in its environment.
2. All requirements marked as “L” (legal) where classified as non-security requirements. Examining the provided samples that are labeled as “legal” requirements. It was determined that even though these may contain security related terms and vocabulary, they do not represent security concerns that effect the system functionality and ability to operate in its environment. And hence this classification would yield more reliable results identifying the security requirements that address operational threats.

3. All requirements marked as “SE” (security) were added to the list and labeled as security requirements.
4. All requirements marked as “A” (availability) were added to the list and labeled as security requirements. Since availability has a direct correlation to the security CIA Triad (Confidentiality, Integrity, and Availability).
5. Functional requirements labeled with “F” and the remaining non-functional requirements (SC, PO, PE, O, MN, and FT) were excluded from the final dataset.
6. Finally, several handpicked samples of security requirements and non-security requirements from general SRS documents were added. In order to further expand the diversity and coverage of the dataset.

The combined and enhanced dataset was then subjected to preprocessing techniques and transformed into a machine digestible form. The processed list consisting of a total of 752 requirements, was then fed to the machine learning models namely: SVM, Naive Bayes and RF. Despite the slight drop in performance measures scores, all models exhibited improved predictions when subjected to a list of requirements from an SRS document. The validation on this iteration concluded that the predictions results were satisfactory.

Table 5 shows the difference in the predictions results between the SecReq dataset [59] used in “Iteration Two” and the SecREP dataset used in “Iteration Three”.

The enhanced SecREP dataset is available on the following link, and can be made available for interested researchers upon request:

https://github.com/shadakhanneh/SecREP-Dataset

Table 5: Machine Learning Predictions Results Comparison

Requirement description	**Iteration Two (**Clean and fixed SecReq dataset)		**Iteration Three (**SecREP)	
The system should support secure virtual private network connections	Machine Learning Model	Prediction	Machine Learning Model	Prediction
	SVM	Non-security	SVM	Security
	Naive Bayes	Non-security	Naive Bayes	Security
	Random Forest	Non-security	Random Forest	Security
The system should be extensible to provide access to the interfaces through PDA's and mobile data terminals	SVM	Security	SVM	Security
	Naive Bayes	Non-security	Naive Bayes	Security
	Random Forest	Security	Random Forest	Security
The system should be designed for access through browser-based systems and must impose minimal requirements on the client device	SVM	Non-security	SVM	Security
	Naive Bayes	Non-security	Naive Bayes	Security
	Random Forest	Non-security	Random Forest	Security
Use of AJAX based technology to improve user experience.	SVM	Non-security	SVM	Non-security
	Naive Bayes	Non-security	Naive Bayes	Non-security

Aggressive page loading to be considered based on the screen and estimate usage pattern	Random Forest	Non-security	Random Forest	Non-security

3.2 Data Preprocessing

In machine learning approaches the most important factor that determines the quality of the process outcome is the condition, completeness, and plausibility of the data that is being fed to the machine for training. Incorrect or poor-quality input will always produce faulty output. for NLP techniques the raw text requires considerable preprocessing in order to transform it into a digestible form that the machine can process and understand. Raw text datasets normally contain noisy, duplicated, lengthy, high dimensional, corrupted, and empty data that is not valuable for the NLP task. This makes preprocessing a vital step that must proceed every NLP task to produce more accurate and reliable results. Detailed descriptions for each text preprocessing technique used in this work's experiments is provided in Section 2.2.2

The below listed text preprocessing techniques were performed before feeding the SecREP to the machine learning models.

1. Manual cleaning and Null/NaN values substitution, which were performed inherently while producing the enhanced dataset. As described before in Section 3.1.
2. Stop words, non-numeric and punctuation removal
3. Stemming and lemmatization
4. Lower case text conversion

5. Part-of-speech (POS) tagging to understand if the word is noun or verb or adjective…etc.
6. Vectorization using Term Frequency—Inverse Document Frequency (TF-IDF).

Additionally in the context of text preprocessing, negation handling represents a common issue that must be taken into account especially in sentiment analysis problems. However, for the problem at hand, negations were considered irrelevant and hence removed. This decision was made based on the observation, that a security requirement is classified as such depending on what it offers to the system or what it prevents or both. For example, a requirement could be considered a security requirement if it is addressing the act of "access" or the act of "no access", where both cases will provide similar outcomes to the classification prediction.

Table 6 shows examples of the dataset before and after applying the preprocessing techniques.

Table 6: Dataset Text Examples Before and After Preprocessing

Original Requirement Text	Requirement Text after Preprocessing	Label
Successful authentication is a prerequisite for the processing of any on-line or offline CEPS transaction	Successful authentication prerequisite processing offline ceps transaction.	Security
In the case of an unlinked load, no presumption may be made as to the business relationship between the cardholder, the funds issuer, the card issuer, or the load acquirer	case unlinked load presumption may make business relationship cardholder fund issuer card issuer load acquirer	Non security

3.3 Machine Learning Techniques

This section describes the training and validation processes adopted to train the selected machine learning models, which are the SVM, Naive Bayes and Random Forest (These models are explained in detail in Chapter 2 Background). These three models are highly regarded and widely praised to exhibit accurate results in classification tasks. Especially for small-size datasets. Where despite the enhanced SecREP dataset of 752 records, is still the case. For example, machine learning models such as the Convolution Neural Networks (CNN), Recurrent Neural Networks (RNN) are known to require thousands of samples to produce satisfactory result.

3.3.1 Training

The three selected machine learning models: SVM, Naive Bayes and Random Forest, were trained using the Scikit-Learn library [50] implementation. After applying the selected preprocessing steps described earlier in Section 3.2. These pre-built models were trained using the cleaned 584-SecReq dataset, and the enhanced SecREP dataset of 752 requirements. Two validation methods were applied in each training experiment: Resampling and N-Folds Cross Validation. The performance of each model was assessed using the performance measure: Precision, Recall, Accuracy, and the F1-Score. These performance measures or metrics are described in detail in Section 2.7.

3.3.2 Experiment 1

Using the 584-SecReq dataset, the three machine learning models were trained and validated using a resampling of 80% training and 20% testing and a 5-folds cross validation. Additionally, the performance metrics for each validation approach were calculated.

The resulting training curve for each model is exhibited in Figure 9.

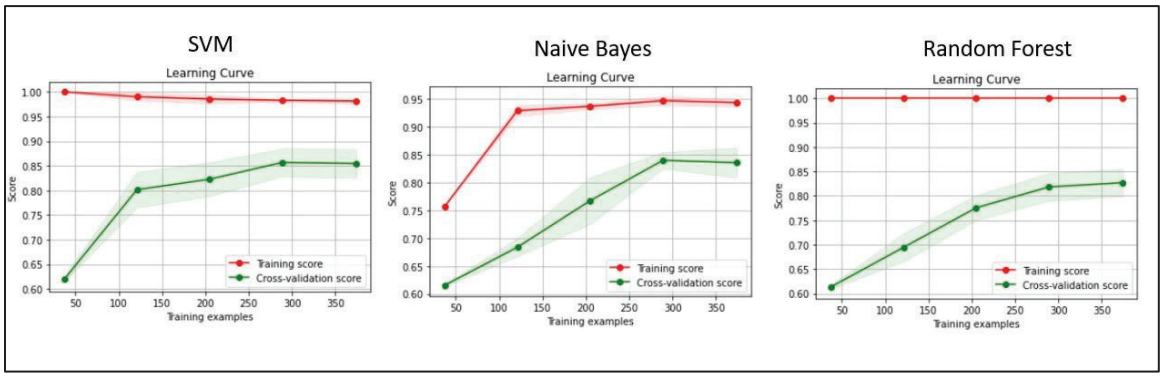

Figure 9: Machine Learning Models Learning Curve for the SecReq Dataset Training

The performance metrics results for each machine learning model using resampling of 80% for training and 20% for validating are showcased in Table 7 and Figure 10. While the 5-folds cross validation performance metrics results are showed in Table 8 and Figure 12.

Table 7: Performance Measures Results Using the SecReq Dataset and 20% Resampling Validation

Model	Accuracy	correctly classified samples	Precision	Recall Score	F1 Score
Naive Bayes (NB)	85.47	100	0.81	0.71	0.76
Support Vector Machine (SVM)	92.30	108	0.89	0.86	0.88
Random Forest (RF)	90.59	106	0.90	0.78	0.84

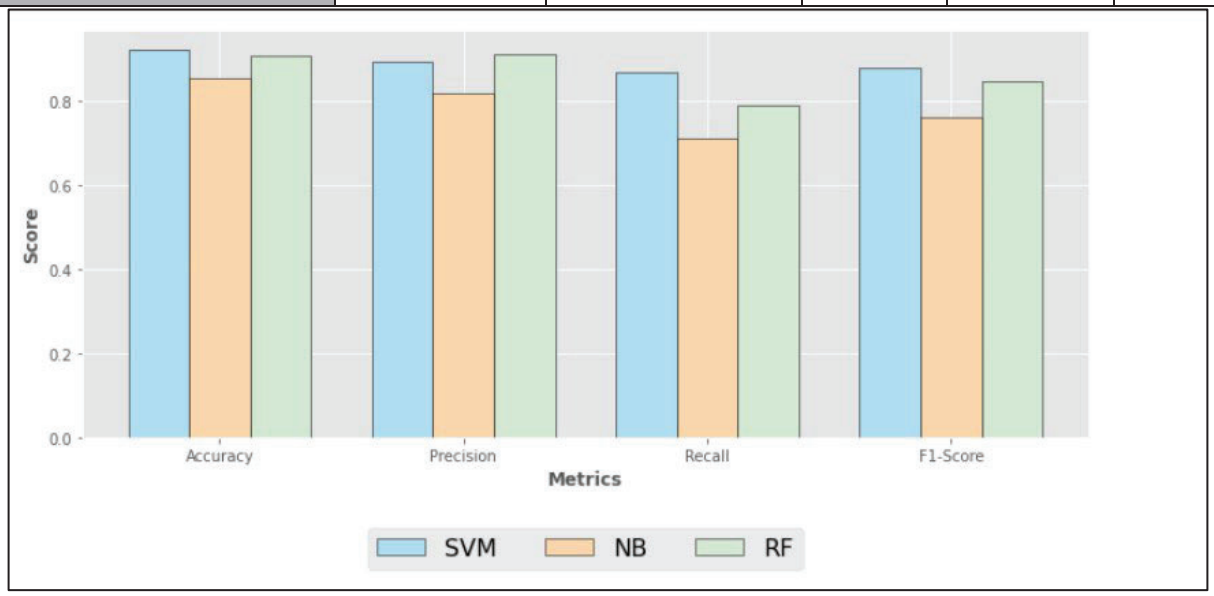

Figure 10: Performance Measures Results Graph Using the SecReq Dataset and 20% Resampling Validation

Figure 11 below illustrates the confusion metrics used in calculating the performance scores for each model.

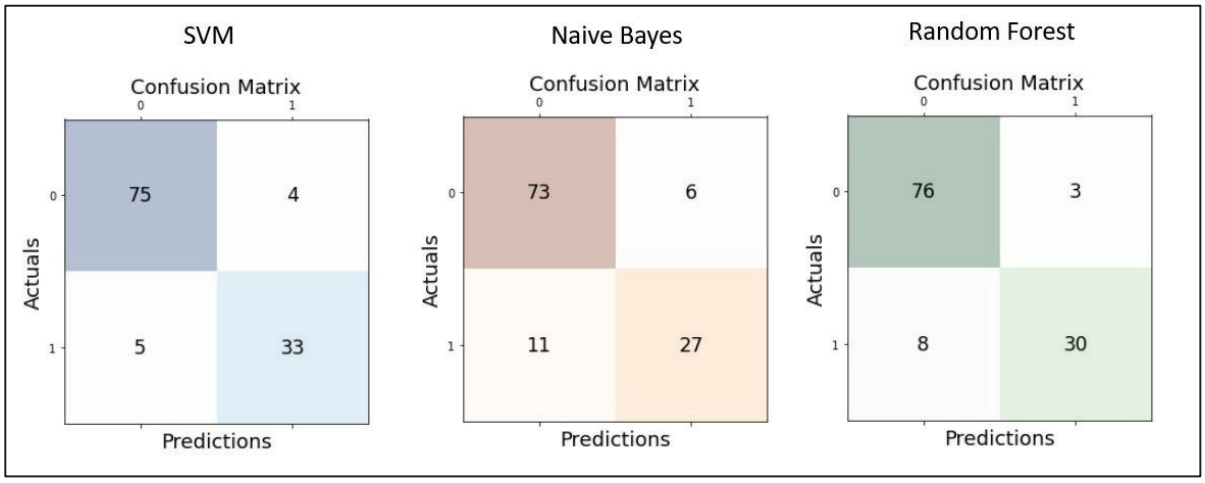

Figure 11: Machine Learning Models Confusion Metrics for the SecReq Dataset Training

Table 8: Performance Measures Results Using the SecReq Dataset and 5-Folds Cross Validation

Model	Mean Accuracy	Mean Precision	Mean Recall Score	Mean F1 Score
Naive Bayes (NB)	0.70	0.77	0.67	0.64
Support Vector Machine (SVM)	0.70	0.76	0.68	0.66
Random Forest (RF)	0.67	0.75	0.64	0.60

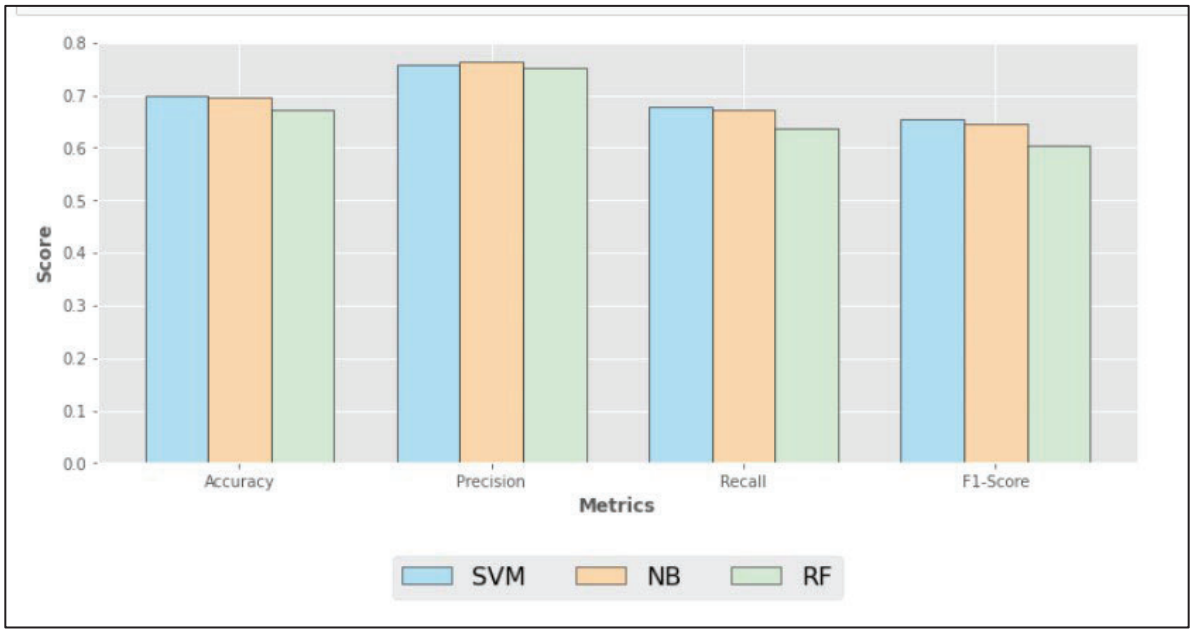

Figure 12: Performance Measures Results Graph Using the SecReq Dataset and 5-Folds Cross Validation

3.3.3 Experiment 2

Using the SecREP dataset, the three machine learning models were trained and validated using a resampling of 80% training and 20% testing and a 5-folds cross validation. Additionally,

the performance metrics for each validation approach were calculated. The resulting training curve for each model is exhibited in Figure 13.

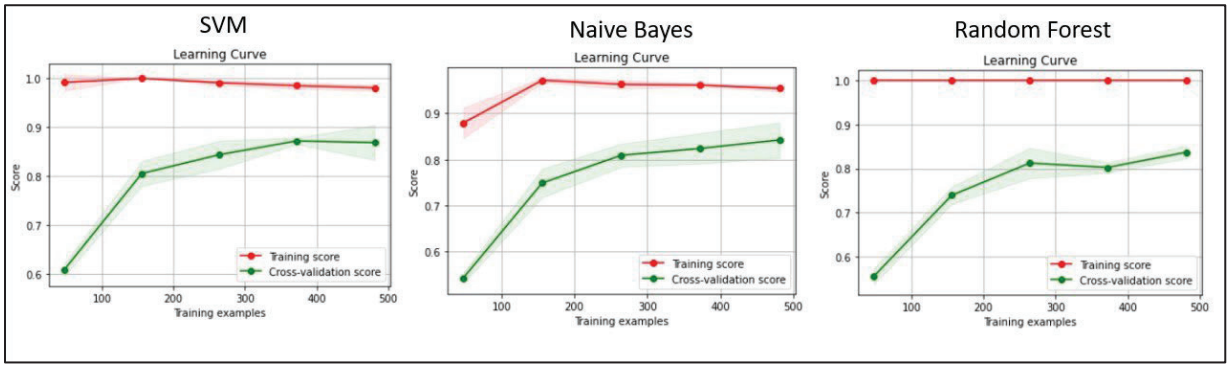

Figure 13: Machine Learning Models Learning Curve for the SecREP Dataset Training

The performance metrics results for each machine learning model using resampling of 80% for training and 20% for validating are showcased in Table 9 and Figure 14. While the 5-folds cross validation performance metrics results are showed in Table 10 and Figure 16. These

Table 9: Performance Measures Results Using the SecREP Dataset and 20% Resampling Validation

Model	Accuracy	correctly classified samples	Precision	Recall Score	F1 Score
Naive Bayes (NB)	92.71	140	89	0.93	0.91
Support Vector Machine (SVM)	93.37	141	0.90	0.93	0.92
Random Forest (RF)	90.59	136	0.89	0.85	0.87

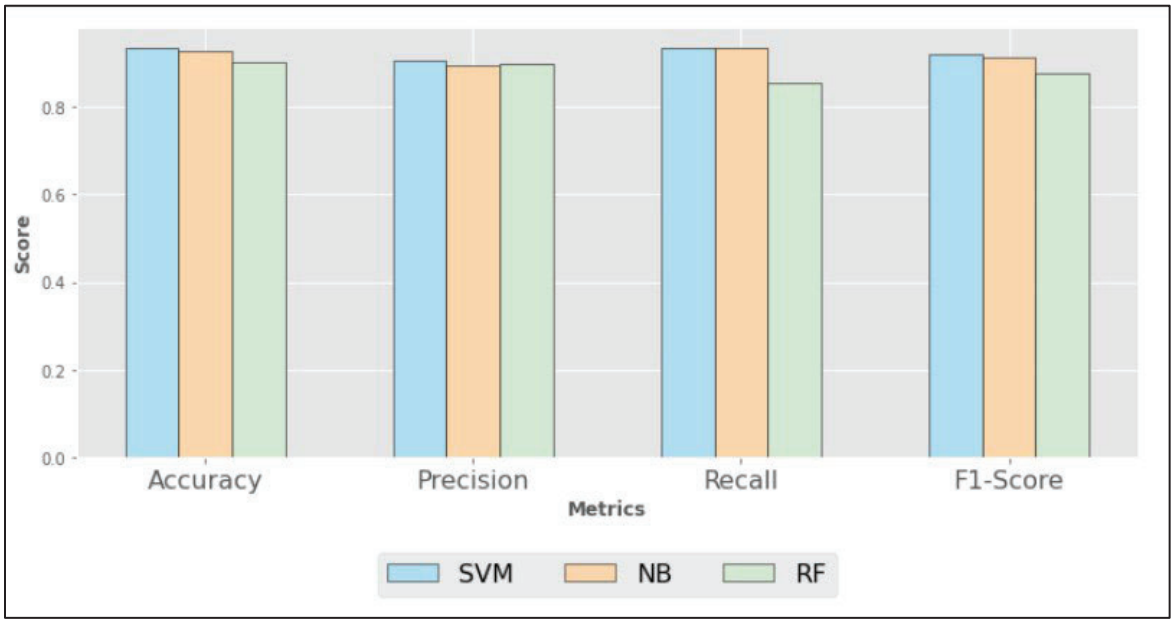

Figure 14: Performance Measures Results Graph Using the SecREP Dataset and 20% Resampling Validation

Figure 15 below illustrates the confusion metrics used in calculating the performance scores for each model.

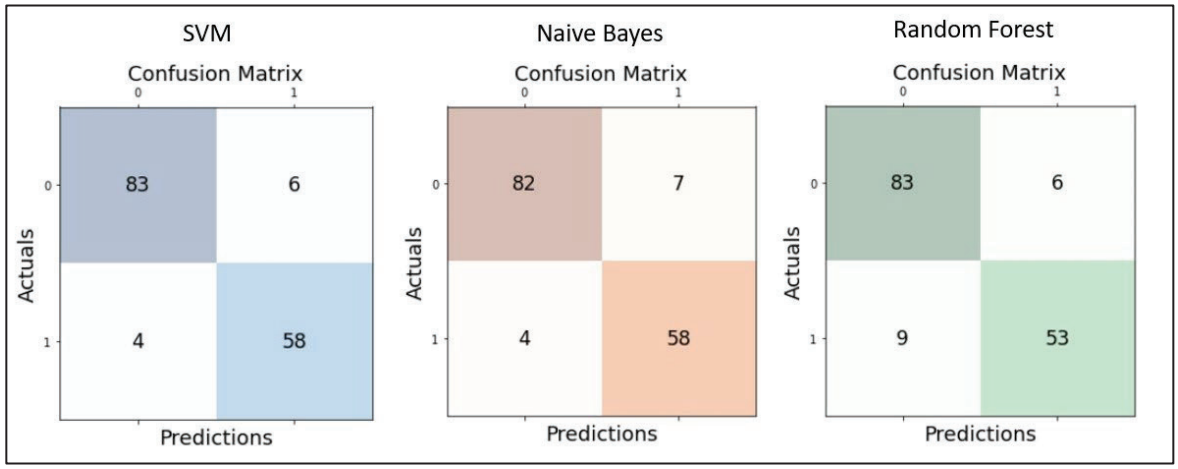

Figure 15: Machine Learning Models Confusion Metrics for the SecREP Dataset Training

Table 10: Performance Measures Results Using the SecREP Dataset and 5-Folds Cross Validation

Model	Mean Accuracy	Mean Precision	Mean Recall Score	Mean F1 Score
Naive Bayes (NB)	0.68	0.76	0.68	0.65
Support Vector Machine (SVM)	0.71	0.77	0.71	0.68
Random Forest (RF)	0.65	0.71	0.65	0.62

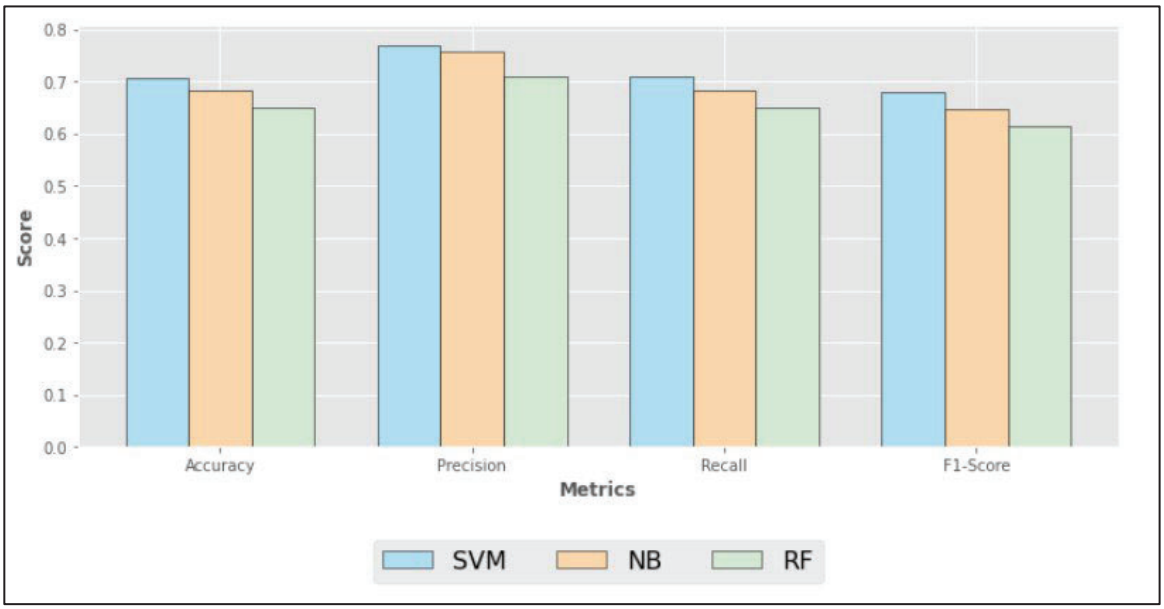

Figure 16: Performance Measures Results Graph Using the SecREP Dataset and 5-Folds Cross Validation

3.3.4 Case Study

For both experiments once the training process was completed. Each trained model was fed a list of a software specifications that was acquired from an SRS document. The specifications list contained 30 requirements in total. Table 11 shows examples taken from that requirements list.

To obtain a robust list that can be determined as a security requirement list of a said system. First the list of 30 requirements were subjected to the preprocessing techniques addressed earlier in Section 2.2.2 to transform it into a machine understandable form. Second, a majority vote ensemble was adopted, where a requirement is added to the list if the three-trained models (the SVM, the Naïve, and the Random Forest) label that requirement as security related.

Table 11: Examples from the Software Requirements List Used for The Case Study

Requirement Description
The system should support multilingual interface
The system should work even in an offline mode with the critical functionality
The search should fetch only the fields that need to be displayed to the user. Only when the user clicks on a particular record to view its further details should a query be fired to fetch the additional details for this particular record only
Database Indexes should be applied on the key columns used for searching
The system must support multiple types of communication services for remote access
The system should be built on a common User Access and Authentication Service to ensure Single-Sign on for the end-user
The system should have capability to support public access to a subset of data and functionality

For the *first experiment* where the ensemble models were trained using the SecReq dataset [59], the ensemble majority vote extracted only 3 requirements as security requirements. Examining the ensemble extracted list in comparison to the original 30 requirements list. It was clear that there was a significant loss in valuable requirements that clearly address security threats or protect the system assets.

Table 12 shows the requirements that were extracted as security related from the first experiment using the SecReq dataset [59].

For the *second experiment* where the ensemble models were trained using the enhanced SecREP dataset, the ensemble majority vote extracted 17 requirements as security requirements. Examining the ensemble extracted list in comparison to the original 30 requirements list. The list does exhibit satisfactory results were all the extracted requirements do address a security issue in terms of threats that need to be accounted for, system assets that needs protecting, or both. Additionally, and in comparison, to the original 30 requirements list that was fed to the ensemble, and using educated judgments the extracted list was comprehensive, and included all the requirements that are indeed security related.

Table 13 shows the requirements that were extracted as security related from the second experiment using the SecREP dataset.

Table 12: Security Requirements List Extracted from First Experiment Using the SecReq dataset

No.	Requirement Description
1:	The system should be built on a common User Access and Authentication Service to ensure Single-Sign on for the end-user.
2:	The system should ensure secure transmission of data over the network and utilize SSL and 2-way digital signatures.
3:	The system should ensure high standards of security and access control through: a) Prevent cross-site scripting b) Validate the incoming data / user request c) Encode the incoming data / user request d) Prevent SQL Injection e) Utilize parameterized queries f) Sanitize the user-inputs g) Validate the data both at the client and server h) Do not allow hard delete and perform only soft tagging the row for deletion.

Table 13: Security Requirements List Extracted from Second Experiment Using the SecREP dataset

No.	Requirement Description
1:	The system should work even in an offline mode with the critical functionality.
2:	The system should be designed to have satisfactory performance even in Police Stations connected on low-bandwidth.
3:	The system should be built on a common User Access and Authentication Service to ensure Single-Sign on for the end-user.
4:	The system should be developed to be deployed in a 3-tier datacenter architecture.

5:	The system should be designed to have a n-tier architecture with the presentation logic separated from the business logic that is again separated from the data-access logic.
6:	The system should be extensible to provide access to the interfaces through PDA's and mobile data terminals.
7:	The system should adopt standardized formats and common metadata elements.
8:	The system should be designed for access through browser-based systems and must impose minimal requirements on the client device.
9:	The system should support SSL encrypted connections.
10:	The system should support secure virtual private network connections.
11:	The system should run on multiple browsers.
12:	The system should support selective encryption of the stored data.
13:	The system should ensure secure transmission of data over the network and utilize SSL and 2-way digital signatures.
14:	The system should ensure high standards of security and access control through: a) Prevent cross-site scripting b) Validate the incoming data / user request c) Encode the incoming data / user request d) Prevent SQL Injection e) Utilize parameterized queries f) Sanitize the user-inputs g) Validate the data both at the client and server h) Do not allow hard delete and perform only soft tagging the row for deletion.
15:	Use of cache for storing frequent data.
16:	The search should fetch only the fields that need to be displayed to the user. Only when the user clicks on a particular record to view its further details should a query be fired to

	fetch the additional details for this particular record only.
17:	Database Indexes should be applied on the key columns used for searching.

3.4 Discussion

To build part one of the SecREP pipeline where security related requirements can be identified in an automated manner using a machine learning classification approach. First, and as for all machine learning classification problems, a reliable dataset must be obtained. That is, a dataset with sufficient, clear, balanced, and assorted samples, that can be used to achieve a reliable prediction model. For the classification problem of identifying/extracting security requirements from software requirements natural language artifacts. Two datasets were identified that are publicly available and can serve this purpose, the SecReq dataset [59] and the NFR dataset [60]. However, both datasets present a major challenge to the problem at hand. The issues with the SecReq dataset [59] besides the intensive cleaning and fixing needed, the security requirements samples are all expressed as non-functional requirements. Which would jeopardize the completeness of extracting all security related requirements, since and as explained before, security requirements intertwine with functional and non-functional requirements. Similar to the SecReq dataset [59] the NFR dataset [60] also classifies the security requirements under the non-functional requirements. Additionally, the 11 categories of NFR can also be expressing a security concern, and thus a considerable amount of these requirements can also be extracted as a security requirement. To address these issues, different combinations of the available datasets were conducted and put under the test. Each experiment/iteration helped improve the predictions of the trained machine learning models. The resulting enhanced dataset “SecREP Dataset” is a

combination of a cleaned version of the SecReq dataset [59] where all corrupted records were rewritten, unclassified entries were labeled using educated judgments, and some large and unclear requirements samples were split into two or more requirements and labeled accordingly. This cleaned SecReq dataset [59] was then combined with some samples obtained from the NFR dataset [60]. The sample selection criteria from the NFR dataset [60] were based on educated judgments and experimentation. Finally, some samples were handpicked form SRS document to further increase the diversity of the dataset.

To test the validity of the SecREP dataset, two experiments were conducted where in one the machine learning models were trained using the SecRq dataset and using the SecREP in the other. The training results, in terms of performance measures, shows similar results for both variants of trained models. However, in the case study where each trained ensemble was fed a list of requirements obtained form an SRS document. The trained model using the SecREP dataset exhibited significantly improved results, and the ensemble majority vote was able to extract 17 requirements that clearly address a security concern. Whereas the trained ensemble model using the SecReq [59] dataset was able to extract only 3 requirements as security related. These results highlight a notable loss of many requirements that should have been extracted when the training was conducted using the SecReq dataset, and a notable increase in the prediction performance using the SecREP enhanced dataset.

CHAPTER 4: Security Requirements Prioritization

4.1 Prioritization Scheme

The objective of this thesis work is to 1) produce an automatic framework that extracts security related requirements from a software system's natural language artifacts, for example, a Software Requirements Specification document (an SRS). And 2) to produce a prioritization scheme where the extracted list of security requirements can be further refined into a prioritized list. This section represents the prioritization scheme adopted to in second step of the SecREP pipeline showed before in Section 1.4

Security requirements are complicated in nature. They express many aspects of a software system and intertwine with all its properties, as well as with other types of requirements. Examining security prioritization techniques introduced in literature that were also summarized in Section 1.2.2. Many factors need to be identified in a security requirement to assess it is quality and importance. These factors can vary depending on the nature and functionality of the software system under question, the stakeholders need, and the system's operational environment. However, and in terms of security, a few constant factors can be used to express the quality of a security requirement. These are, the assets a security requirement is protecting, the threats it is attempting to mitigate, and the security properties its expressing (e.g., Confidentiality, Integrity, Availability…).

4.2 Proposed Prioritization Scheme

The prioritization scheme in this work, considers the quality and valuation of a security requirements based on the three constant factors that were identified based on this work's literature study. These factors are assets, threats, and security attributes. With the aid of NLP

approaches that can be used to extract information form a software system's requirements artifacts, such factors can be more adequately identified in the context of a specified system.

To achieve the second part of SecREP framework, the extracted security requirements list from the first part was subjected to a part of speech tagging method and a named entity recognition method. To help recognize from that list, assets that need protecting, threats that need to be addressed, and security properties the system must incorporate. Once these factors are identified, a list of each factor can be established, where each security requirement can then be tested against each list and weighed according to the number of assets, threats and security properties being addressed in that specific security requirement. Additionally, to derive more meaning to the security requirement and ensure that all threats are accounted for, each security requirement was granted a STRIDE score. The STRIDE score was calculated based on text similarity scoring approach. For each STRIDE threat a super-sentence that describes how the system should prevent such threat was constructed. Using these super-sentences each security requirement similarity score to each STRIDE sentence can thus be calculated. The security requirement STRIDE score can then be expressed based on the number of STRIDE threats that sentence is addressing. Finally, the priority score of each requirement can then be the sum of it is corresponding STRIDE score, and the count score for each asset, threat, and security property present in the security requirement.

4.3 Case Sturdy and Prioritization Results

Continuing with the case study presented in the first part of the SecREP pipeline under Section 3.3.4. The list of 17 security requirements was subjected to the preprocessing techniques discussed in Section 3.2. To transform the text into a machine digestible form.

4.3.1 Constructing the Comparison Lists

Following the proposed prioritization pipeline. First, three lists were constructed: a list of assets, a list of threats, and a list of security properties or attributes. The values in each of these lists were chosen using generic values of known assets, threats, and security properties. Additionally, the security requirements list was loaded into a spaCy natural language possessing medium model, to utilize the models Word2vec vectorization method discussed in Section 2.2.3.2. In addition to the Word2vec method, the spaCy library offers several built in NLP techniques.

To further enhance the comparison lists and make them specific to the context of the given system. The spaCy Part-of-speech (POS) tagging method and the spaCy matcher were used to extract, subjects, objects, nouns, subjects followed by verbs, and objects followed by a verb. As expected, the resulting list from this extraction, contained a considerable number of items. However, using educated judgments, the list was reduced to cover what was considered either an asset, a threat, or a security property.

With this new insight into the system at hand, the complete comparison lists were constructed, where each list contains general items, and items extracted from the specific nature of the security requirement list.

Table 14 shows the final comparison lists that were constructed as a result of processing the security requirements list. While Figure 17 demonstrates the identification of these items in the security requirements list.

Table 14: Comparison lists

Type	**Items**
Assets	Data, network, information, database, client, datacenter, design, user inputs, record, architecture, access interface, client device, connection, business, certificate, user.
Threats	access, spoof, alter, change, delete, modify, update, edit, tamper, repudiation, denial, unauthorized, injection, deny, phishing, phish.
Security Properties	Authentication, authenticity, authentic, authorization, confidential, available, availability, confidentiality

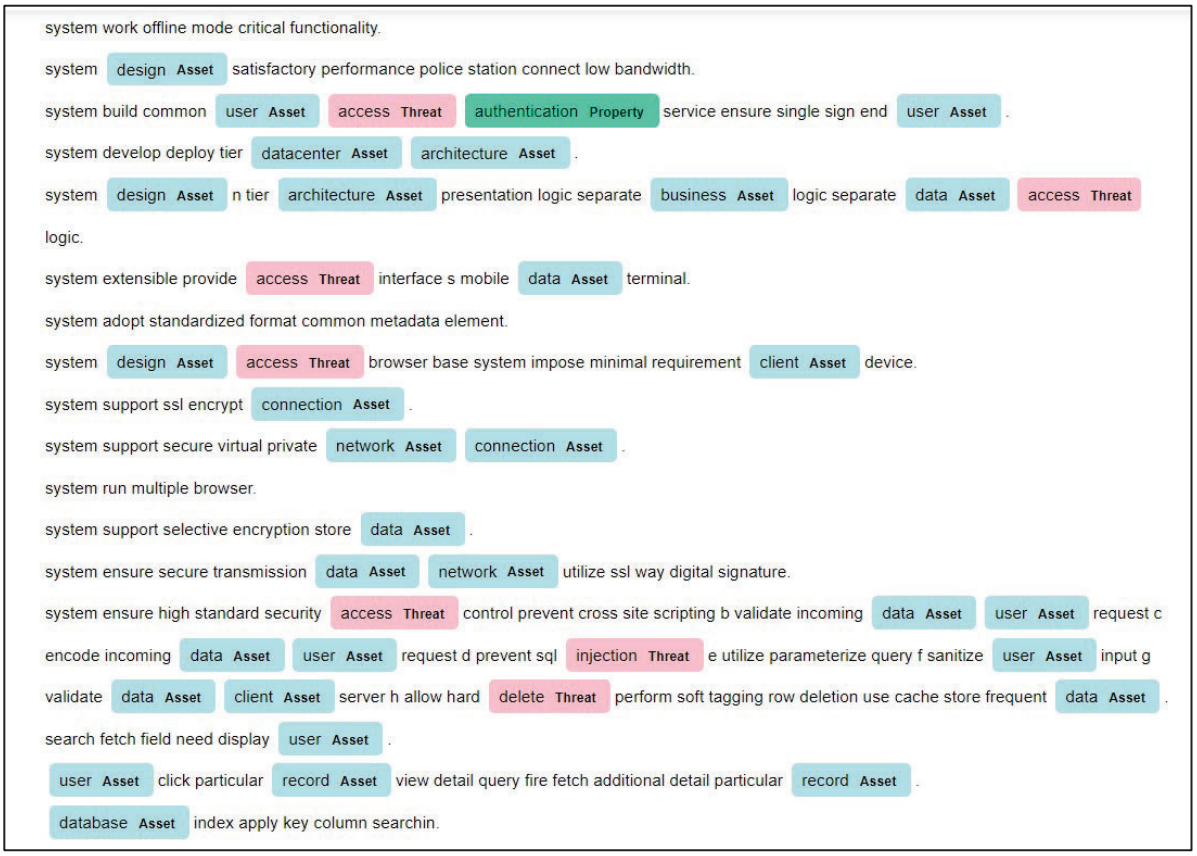

Figure 17: Identifying Assets, Threats, and Security Properties

4.3.2 Constructing the Super-Sentences

To add more robustness to the SecREP framework prioritization scheme. An approach was adopted to identify if a security requirement protects against or is addressing a threat that corresponds to the STRIDE threat modeling. The STRIDE model developed by Microsoft [30] is a popular method used to identify security threats and groups them in six different categories: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.

To incorporate the STRIDE model into the prioritization scheme. First, each security requirement must be examined to see not only if a direct mention of a threat is present, but also if

that requirement derived meaning correlates to the definitions of the STRIDE threats. To tackle this task a text similarity technique was chosen. To use the idea of text similarity, first there must be another sentence that can be compared to the text under question, to see if that text contains the desired information. To do so, a list of super-sentences was constructed where each sentence in the list corresponds to a behavior or quality, that would protect against the STRIDE threats.

With the establishment of the super-sentences, the problem becomes a simple similarity problem to see how similar each security requirement is to that defined super-sentence. For measuring a similarity score between each security requirement in the case study and the super-sentences, the spaCy similarity function was used. The resulting score is represented as a percentage of similar. To simplify the results, a security requirement was given a weight of 1 for each STRIDE super-sentence with a similarity of 90% and above. Hence, the total STRIDE score of a security requirement is the sum of all STRIDE similarity scores. Thus, in this case the minimum STRIDE score a security requirement can have is 0 and the maximum is 6.

Table 15 describes the list of super-sentences.

Table 15: STIDE Threat Model and The Super-Sentences

Threat type	Property	Super-Sentence Description
Spoofing	Authentication	Prevent illegal access and using another user's authentication information, such as username and password
Tampering	Integrity	Prevent the malicious modification of data. Examples include unauthorized changes made to persistent data, such as that held in a database, and the alteration of data as it flows between two computers over an open network, such as the Internet
Repudiation	Non-repudiation services	Prevent users from performing an illegal action without other parties having any way to prove otherwise, when a user performs an illegal operation, the system should provide the ability to trace that prohibited operations. The system should have the ability to counter repudiation threats.
Information disclosure	Confidentiality	Prevent the exposure of information to individuals who are not supposed to have access to it. prevent the ability of users to read a file that they were not granted access to, or the ability of an intruder to read data in transit between two computers
Denial of service	Availability	Prevent denial of service (DoS) to valid users. Prevent making a Web server temporarily unavailable or unusable. A system should be available and reliable
Elevation of privilege	Authorization	Prevent an unprivileged user form gaining privileged access that is sufficient to compromise or destroy the entire system. A system should protect against situations in which an attacker has effectively

		penetrated all system defenses and become part of the trusted system itself

4.3.3 The Prioritized List

Each security requirement in the list was compared with each item in the three comparisons list that were constructed earlier. The weight of each security requirement is increased by 1 if a matching item in these lists appears in the security requirement. Similarly, after calculating the total STRIDE score for each sentence, the weight of each requirement was incremented by its STRIDE score. Thus, the priority score for a given security requirement is the sum of the count of all the assets, threats, and security properties it is addressing, in addition to its STRIDE score that was defined earlier.

Figure 18 demonstrates the final extracted and prioritized security requirements list as a result of the SecREP pipeline.

Requirement	Asset	Threat	Sec.P	Total	S	T	R	I	D	E	STRIDE score	Priority
The system should ensure high standards of security and access control thro	3	3	0	6	0	1	0	1	1	0	3	9
The system should ensure secure transmission of data over the network and	2	0	0	2	1	1	1	1	1	0	5	7
The system should be built on a common User Access and Authentication Su	1	1	1	3	1	0	1	0	1	0	3	6
The system should be designed for access through browser-based systems a	2	1	0	3	0	0	1	0	1	1	3	6
The system should be designed to have a n-tier architecture with the preser	4	1	0	5	0	0	0	0	0	0	0	5
The system should support secure virtual private network connections.	2	0	0	2	0	0	0	0	0	1	1	3
The system should support selective encryption of the stored data.	1	0	0	1	0	1	0	0	1	0	2	3
The system should be developed to be deployed in a 3-tier datacenter archi	2	0	0	2	0	0	0	0	0	0	0	2
The system should be extensible to provide access to the interfaces through	1	1	0	2	0	0	0	0	0	0	0	2
Only when the user clicks on a particular record to view its further details sh	2	0	0	2	0	0	0	0	0	0	0	2
The system should support SSL encrypted connections.	1	0	0	1	0	0	0	0	0	0	0	1
The search should fetch only the fields that need to be displayed to the use	1	0	0	1	0	0	0	0	0	0	0	1
Database Indexes should be applied on the key columns used for searchin.	1	0	0	1	0	0	0	0	0	0	0	1
The system should be designed to have satisfactory performance even in Pe	1	0	0	1	0	0	0	0	0	0	0	1
The system should run on multiple browsers.	0	0	0	0	0	0	0	0	0	0	0	0
The system should adopt standardized formats and common metadata elen	0	0	0	0	0	0	0	0	0	0	0	0
The system should work even in an offline mode with the critical functional	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18: final extracted and prioritized security requirements list

4.4 Discussion

To establish part two of the SecREP pipeline where extracted security related requirements can be further analyzed and prioritized. First, a prioritization criterion was established. The focus of this prioritization scheme was to incorporate NLP automated techniques, such that minimal human interaction is needed. The proposed prioritization assesses each requirement based on the number of assets, threats, and security properties that are directly being addressed by the requirement. Additionally, and to add robustness to the prioritization, the underlying meaning of a security requirement was also accounted for, using text similarity approach to compare the security requirement against six super-sentences each corresponding to the STRIDE threat model.

The case study showed how this prioritization scheme is conducted and how the priority score was calculated. The final list of prioritized security requirements showcased the difference in priority score between security requirements. Additionally, inspecting the list, the topmost security requirements (with highest priority score) show a high level of correlation to critical security aspects that must be addressed in every software system. What is worth mentioning here, is that this prioritization scheme could be a reliable start to eliciting and prioritizing security requirements. However, to ensure the completeness of a security requirement and its priority, other factors and calculations must also be considered. For example, this proposed prioritization scheme could be adopted to help identify assets, threats, security properties, and to map the threats discussed by the security requirement to the STRIDE threat model. After doing so, software engineers can use this information to further assess the security requirement in terms of the risk value, the impact, the system's vulnerabilities that can cause a threat to occur, the underlying cost of implementing the security mechanisms...etc. Such values can improve the

completeness and correctness to the security requirement and its priority. Additionally, further valuation of the security requirement can help address those with competing priority scores, that is two or more security requirement have the same priority weight. Additionally, techniques such as, the AHP[49,57] the CVSS (Common Vulnerability Scoring System), and the bug bars, can help in calculating these values and solve the issue with competing security requirements.

CHAPTER 5: Conclusion

This chapter provides the major conclusions and the planned future work based on the results of this research effort.

5.1 Conclusion

This thesis has described the work and steps taken to achieve the Security Requirements Extraction and Prioritization (SecREP) framework/pipeline. The SecREP pipeline presented in this thesis consists of two major parts. Part one, demonstrates the efforts and experiments conducted to establish a machine learning ensemble that extracts/identifies security requirements from natural language software requirements artifacts. The majority vote ensemble was created using three machine learning models, SVM, Naïve Bayes, and Random Forests. Two different experiments were conducted to train and evaluation the machine learning models. In the first experiment the machine was fed the SecReq dataset [59]. In the second experiment the machine was fed the enhanced SecREP dataset that was conducted as a result of this work. Both trained ensembles were applied to a case study. Where each ensemble model was fed a list of 30 requirements obtained form an SRS document. The results showed significantly improved predictions using the majority vote of the ensemble trained on the SecREP dataset. The proposed approach was able to extract 17 requirements as security requirements. In comparison, the ensemble trained using the SecReq dataset [59] was able to extract only 3 requirements.

In the second part of the SecREP pipeline, a prioritization scheme was proposed to evaluate the importance and analyze the extracted security requirements. The SecREP prioritization process, valuates the priority of each security requirements in terms of the assets it

is protecting, the threats it is addressing, and its security properties. In addition to its similarity to a super-sentence that corresponds to the STRIDE threat model.

The proposed prioritization scheme was applied to the case study, to prioritize the extracted list of 17 security requirements. Using NLP techniques such as, Part-of-speech (POS) tagging and named entity recognition, three lists were conducted that represent the system assets, threats, and security properties. Each security requirement was then compared against each item in those lists, to extract the number of assets, threats, and security properties that are present in the requirement. Finally using the Word2Vec and text similarity methods provided in spaCy library [23], each security requirement was compared against 6 super-sentence that correspond to the STRIDE threats definitions. In order to extract its STRIDE similarity score. The priority score for each requirement was calculated that is, the sum of its assets, threats, and security properties, and its STRIDE score. The final list of prioritized security requirements was evaluated using expert judgment, where the list exhibited a variance in the priority scores and the security requirements with highest priority scores do indeed show a high level of correlation to critical security aspects that must be addressed in every software system.

5.2 Limitation of Study

Obtaining a dataset that can be utilized in a machine learning approach for classifying security requirements was perhaps the most challenging aspect of this study. Two datasets that can be used for such problem were publicly available. The SecReq dataset [59] and the NFR dataset [60]. The concerns with these datasets are the low count of samples and the domain specific examples. Additionally, and what is perhaps the biggest limitation is the fact that all security labeled requirments are expressed as non-functional requirements. This classification

under non-functional requirements can jeopardize the task at hand. That is to extract a comprehensive list of security requirements for a software system form natural language artifacts. In order for that list to be complete and correct, security requirements must be extracted from functional and non-functional requirements alike. Due to the fact that security concerns and the security aspects of a system relate directly to many of its functoriality, environment and performance.

Despite the construction of the SecREP dataset in this work. Training a machine learning model to extract security requirements from software systems natural language artifacts. Can be significantly improved by further improving the dataset to contain more versatile and correct samples that express the security requirements as functional and non-functional. Additionally, increasing the size of the dataset will produce better learnt machine models, and can also allow for training other machine learning models than the ones addressed in this study (SVM, Naïve Bayes, and Random Forest). For example, models such as the CNN, RNN and deep learning that require large datasets, can be used to obtain remarkable results.

In terms of this work validity, the SecREP framework/pipeline was tested on one case study and its results evaluation was based on its performance with that example. Further testing can help fortify the validity of the SeREP framework/pipeline and can help recognize weak points that needs addressing.

Finally, and what is worth mentioning here, is that this prioritization scheme provided by the SecREP is only a reliable start to eliciting and prioritizing security requirements. However, to ensure the completeness of a security requirement and its priority, other factors and calculations must also be considered. For example, the risk value, the impact, the system’s vulnerabilities that

can cause a threat to occur, the underlying cost of implementing the security mechanisms...etc. Such values can improve the completeness and correctness to the security requirement and its priority. Additionally, the prioritization scheme proposed in this work does not address the problem of security requirements with competing priority scores, that is two or more security requirement have the same priority weight. However, techniques such as, the AHP [45,52] the CVSS (Common Vulnerability Scoring System), and the bug bars, can help in calculating these missing values and solve the issue with competing security requirements.

5.3 Future Work

As a result of the work conducted in this thesis, several research opportunities, unexplored areas, and enhancements openings can be addressed in future work.

- ***Enhancing the SecREP Dataset***: The SecREP dataset can be further enhanced, in terms of size, completeness, correctness and diversity of the samples. A need is still dire for a large in scale dataset of software requirements classified as either security related as non-security related. That ensures to incorporate security requirements samples that are expressed as functional security requirements and non-functional security requirements.

  Additionally, different combinations of the SecReq [59] dataset and the NFR dataset [60] can still be tested to see if that would improve upon the SecREP dataset.

  Finally, there is an opportunity to use the SecREP extraction model, or similar models, to help extract security requirements samples from software systems

natural language artifacts, where these samples can be then added to the SecREP list and used to retrain the SecREP model or to train different models.

- ***Evaluating other Machine Learning models***: Other machine learning models for example, CNNs, RNNs, Decision tree, KNN algorithm, K-means…etc. Can still be experimented with using the SecREP dataset, the SecReq [59] dataset and the NFR dataset [60], or different combinations of them.
- ***Additional Case Studies***: The SecREP pipeline can be subjected to different case studies for different domains. For example, to extract security requirements form users reviews and feedback, or mobile applications reviews. Additionally, there is a comparison study opportunity, where the SecREP framework validity and performance can be tested against similar or related models.
- ***Enhanced Prioritization***: The prioritization scheme proposed by the SecREP framework, has many areas of improvements. First the prioritization can be further extended and automated to account for other important values regarding security requirements, for example risk calculation, threat impact, likelihood of a threat...etc. Second, a scheme can be added to the prioritization process to address the issue with competing security requirements with similar priority scores. Third, the scheme can be further automated to extract any human element, wherein the SecREP pipeline, software engineers need to finalize and review the comparison lists extracted by the process that correspond to their system assets, threats, and security properties. Finally, the prioritization scheme can be further tested on different case studies and can be compared and tested against other similar or related approaches.

References

[1] Zahra Shakeri Hossein Abad, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe, and Kurt Schneider. 2017. What Works Better? A Study of Classifying Requirements. (July 2017). Retrieved from http://arxiv.org/abs/1707.02358

[2] Malik Nadeem Anwar Mohammad, Mohammed Nazir, and Khurram Mustafa. 2019. A Systematic Review and Analytical Evaluation of Security Requirements Engineering Approaches. *Arabian Journal for Science and Engineering* 44, 11 (November 2019), 8963–8987. DOI:https://doi.org/10.1007/s13369-019-04067-3

[3] Ahmad Taher Azar, Hanaa Ismail Elshazly, Aboul Ella Hassanien, and Abeer Mohamed Elkorany. 2014. A random forest classifier for lymph diseases. *Computer Methods and Programs in Biomedicine* 113, 2 (2014), 465–473. DOI:https://doi.org/10.1016/j.cmpb.2013.11.004

[4] Manal Binkhonain and Liping Zhao. 2019. A review of machine learning algorithms for identification and classification of non-functional requirements. (2019). DOI:https://doi.org/10.1016/j.eswax.2019.10

[5] Barry Boehm. 2006. A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*, ACM, New York, NY, USA, 12–29. DOI:https://doi.org/10.1145/1134285.1134288

[6] Edna Dias Canedo and Bruno Cordeiro Mendes. 2020. Software requirements classification using machine learning algorithms. *Entropy* 22, 9 (September 2020). DOI:https://doi.org/10.3390/E22091057

[7] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. 2019. Machine learning and the physical sciences. (March 2019). DOI:https://doi.org/10.1103/RevModPhys.91.045002

[8] Luciano Gonçalves de Carvalho, Marcelo Fantinato, and Marcelo Medeiros Eler. 2020. Security requirements identification and prioritization for smart toys. *Electronic Commerce Research and Applications* 41, (May 2020), 100972. DOI:https://doi.org/10.1016/j.elerap.2020.100972

[9] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2007. Automated classification of non-functional requirements. In *Requirements Engineering*, 103–120. DOI:https://doi.org/10.1007/s00766-007-0045-1

[10] Dev Dave, Vaibhav Anu, and Aparna S. Varde. 2021. Automating the Classification of Requirements Data. In *Proceedings - 2021 IEEE International Conference on Big Data, Big Data 2021*, Institute of Electrical and Electronics Engineers Inc., 5878–5880. DOI:https://doi.org/10.1109/BigData52589.2021.9671548

[11] Onyeka Emebo, Aparna S Varde, and Olawande Daramola. *Common Sense Knowledge, Ontology and Text Mining for Implicit Requirements*.

[12] Wael Etaiwi and Ghazi Naymat. 2017. The Impact of applying Different Preprocessing Steps on Review Spam Detection. In *Procedia Computer Science*, Elsevier B.V., 273–279. DOI:https://doi.org/10.1016/j.procs.2017.08.368

[13] Donald G Firesmith. 2003. *Engineering security requirements*. Retrieved from http://www.jot.fm/issues/issue_2003_01/column6

[14] Peter Flach. 2012. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*.

[15] Kenneth Kofi Fletcher and Xiaoqing Liu. 2011. Security Requirements Analysis, Specification, Prioritization and Policy Development in Cyber-Physical Systems. In *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement - Companion*, IEEE, 106–113. DOI:https://doi.org/10.1109/SSIRI-C.2011.25

[16] Aayush Gulati, Shalini Sharma, and Parshotam Mehmi. 2012. Proposing Security Requirement Prioritization Framework. *International Journal of Computer Science, Engineering and Applications* 2, 3 (June 2012), 27–37. DOI:https://doi.org/10.5121/ijcsea.2012.2303

[17] Ethan Hadar and Amin Hassanzadeh. 2019. Big Data Analytics on Cyber Attack Graphs for Prioritizing Agile Security Requirements. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, IEEE, 330–339. DOI:https://doi.org/10.1109/RE.2019.00042

[18] Soma Halder. 2018. Hands-On Machine Learning for Cybersecurity: Safeguard your system by making your machines intelligent using the Python ecosystem. *Ozdemir, Sinan* (2018).

[19] Charles B. Haley, Jonathan D. Moffett, Robin Laney, and Bashar Nuseibeh. 2006. A framework for security requirements engineering. In *Proceedings of the 2006 international workshop on Software engineering for secure systems - SESS '06*, ACM Press, New York, New York, USA, 35. DOI:https://doi.org/10.1145/1137627.1137634

[20] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software. *ACM SIGKDD Explorations Newsletter* 11, 1 (November 2009), 10–18. DOI:https://doi.org/10.1145/1656274.1656278

[21] Gerhard Hansch, Peter Schneider, and Gerd Stefan Brost. 2019. Deriving Impact-driven Security Requirements and Monitoring Measures for Industrial IoT. In *Proceedings of the 5th on Cyber-Physical System Security Workshop - CPSS '19*, ACM Press, New York, New York, USA, 37–45. DOI:https://doi.org/10.1145/3327961.3329528

[22] Baber Hayat, Ribha Shakoor, Sahrish Mubarak, and Komal Basharat. 2017. A Goal based Framework by adopting SQUARE Process for Privacy and Security Requirement Engineering. *International Journal of Computer Applications* 169, 11 (July 2017), 31–34. DOI:https://doi.org/10.5120/ijca2017914873

[23] Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.

[24] Siv Hilde Houmb, Shareeful Islam, Eric Knauss, Jan Jürjens, and Kurt Schneider. 2010. Eliciting security requirements and tracing them to design: an integration of Common Criteria, heuristics, and UMLsec. *Requirements Engineering* 15, 1 (March 2010), 63–93. DOI:https://doi.org/10.1007/s00766-009-0093-9

[25] Tahira Iqbal, Parisa Elahidoost, and Levi Lucio. 2018. A Bird's Eye View on Requirements Engineering and Machine Learning. In *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, IEEE Computer Society, 11–20. DOI:https://doi.org/10.1109/APSEC.2018.00015

[26] J. Karlsson and K. Ryan. 1997. A cost-value approach for prioritizing requirements. *IEEE Software* 14, 5 (1997), 67–74. DOI:https://doi.org/10.1109/52.605933

[27] Aman Kedia. 2020. Hands-On Python Natural Language Processing: Explore tools and techniques to analyze and process text with a view to building real-world NLP applications. *Rasu, Mayank* (June 2020).

[28] Armin Kobilica, Mohammed Ayub, and Jameleddine Hassine. 2020. Automated Identification of Security Requirements. In *Proceedings of the Evaluation and Assessment in Software Engineering*, ACM, New York, NY, USA, 475–480. DOI:https://doi.org/10.1145/3383219.3383288

[29] Armin Kobilica, Mohammed Ayub, and Jameleddine Hassine. 2020. Automated Identification of Security Requirements: A Machine Learning Approach. In *ACM International Conference Proceeding Series*, Association for Computing Machinery, 475–480. DOI:https://doi.org/10.1145/3383219.3383288

[30] Loren Kohnfelder and Praerit Garg. 1990. The STRIDE Threat Model. Retrieved July 18, 2022 from Microsoft. Microsoft

[31] Rakesh Kumar and Rinkaj Goyal. 2019. On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. *Computer Science Review* 33, (August 2019), 1–48. DOI:https://doi.org/10.1016/j.cosrev.2019.05.002

[32] Zijad Kurtanovic and Walid Maalej. 2017. Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, Institute of Electrical and Electronics Engineers Inc., 490–495. DOI:https://doi.org/10.1109/RE.2017.82

[33] Romain Laborde, Sravani Teja Bulusu, Ahmad Samer Wazan, François Barrère, and Abdelmalek Benzekri. 2019. Logic-based methodology to help security architects in eliciting high-level network security requirements. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, ACM, New York, NY, USA, 1610–1619. DOI:https://doi.org/10.1145/3297280.3297437

[34] Axel van Lamsweerde. 2000. Requirements engineering in the year 00: a research perspective. In *Proceedings of the 22nd international conference on Software engineering - ICSE '00*, ACM Press, New York, New York, USA, 5–19. DOI:https://doi.org/10.1145/337180.337184

[35] Chuanyi Li, Liguo Huang, Jidong Ge, Bin Luo, and Vincent Ng. 2018. Automatically classifying user requests in crowdsourcing requirements engineering. *Journal of Systems and Software* 138, (April 2018), 108–123. DOI:https://doi.org/10.1016/j.jss.2017.12.028

[36] Nancy R. Mead and Ted Stehney. 2005. Security quality requirements engineering (SQUARE) methodology. *ACM SIGSOFT Software Engineering Notes* 30, 4 (July 2005), 1–7. DOI:https://doi.org/10.1145/1082983.1083214

[37] Nancy R. Mead, Venkatesh Viswanathan, and Deepa Padmanabhan. 2008. Incorporating Security Requirements Engineering into the Dynamic Systems Development Method. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, IEEE, 949–954. DOI:https://doi.org/10.1109/COMPSAC.2008.85

[38] Daniel Mellado, Carlos Blanco, Luis E. Sánchez, and Eduardo Fernández-Medina. 2010. A systematic review of security requirements engineering. *Computer Standards & Interfaces* 32, 4 (June 2010), 153–165. DOI:https://doi.org/10.1016/j.csi.2010.01.006

[39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Distributed Representations of Words and Phrases and their Compositionality*.

[40] Tom Mitchell. 1997. Machine Learning. (1997).

[41] Davoud Mougouei. 2017. PAPS: A Scalable Framework for Prioritization and Partial Selection of Security Requirements. (June 2017). Retrieved from http://arxiv.org/abs/1706.00166

[42] Bashar Nuseibeh and Steve Easterbrook. 2000. Requirements engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering - ICSE '00*, ACM Press, New York, New York, USA, 35–46. DOI:https://doi.org/10.1145/336512.336523

[43] Emebo Onyeka, Vaibhav Anu, and Aparna S. Varde. 2019. Identifying Implicit Requirements in SRS Big Data. In *Proceedings - 2019 IEEE International Conference on*

Big Data, Big Data 2019, Institute of Electrical and Electronics Engineers Inc., 6169–6171. DOI:https://doi.org/10.1109/BigData47090.2019.9006086

[44] Emebo Onyeka, Aparna S. Varde, Vaibhav Anu, Niket Tandon, and Olawande Daramola. 2020. Using Commonsense Knowledge and Text Mining for Implicit Requirements Localization. In *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, IEEE Computer Society, 935–940. DOI:https://doi.org/10.1109/ICTAI50040.2020.00146

[45] Keun-Young Park, Sang-Guun Yoo, and Juho Kim. 2011. Security Requirements Prioritization Based on Threat Modeling and Valuation Graph. In *Communications in Computer and Information Science*. 142–152. DOI:https://doi.org/10.1007/978-3-642-24106-2_19

[46] Maria Riaz, Jason King, John Slankas, and Laurie Williams. 2014. Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, IEEE, 183–192. DOI:https://doi.org/10.1109/RE.2014.6912260

[47] Lorijn van Rooijen, Frederik Simon Bäumer, Marie Christin Platenius, Michaela Geierhos, Heiko Hamann, and Gregor Engels. 2017. From user demand to software service: Using machine learning to automate the requirements specification process. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*, Institute of Electrical and Electronics Engineers Inc., 379–385. DOI:https://doi.org/10.1109/REW.2017.26

[48] A.D. Rubin and D.E. Geer. 1998. A survey of Web security. *Computer (Long Beach Calif)* 31, 9 (September 1998), 34–41. DOI:https://doi.org/10.1109/2.708448

[49] Mohd. Sadiq, Jawed Ahmed, Mohammad Asim, Aslam Qureshi, and R. Suman. 2010. More on Elicitation of Software Requirements and Prioritization Using AHP. In *2010 International Conference on Data Storage and Data Engineering*, IEEE, 230–234. DOI:https://doi.org/10.1109/DSDE.2010.23

[50] scikit-learn.org. scikit-learn Machine Learning in Python. Retrieved July 20, 2022 from https://scikit-learn.org/stable/

[51] Shalini Sharma and Ajit Singh Malik. 2012. A Novel Framework for Security Requirement Prioritization. *International Journal of Computer Applications* 38, 8 (January 2012), 9–14. DOI:https://doi.org/10.5120/4626-6868

[52] J, Sayyad Shirabad and Menzies, T.J. 2005. The PROMISE Repository of Software Engineering Databases. *School of Information Technology and Engineering, University of Ottawa, Canada* (2005). Retrieved July 19, 2022 from http://promise.site.uottawa.ca/SERepository

[53] Hakim Sultanov and Jane Huffman Hayes. 2013. Application of reinforcement learning to requirements engineering: requirements tracing. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, IEEE, 52–61. DOI:https://doi.org/10.1109/RE.2013.6636705

[54] Inger Anne Tondel, Martin Gilje Jaatun, and Per Hakon Meland. 2008. Security Requirements for the Rest of Us: A Survey. *IEEE Software* 25, 1 (January 2008), 20–27. DOI:https://doi.org/10.1109/MS.2008.19

[55] Hugo Villamizar, Marcos Kalinowski, Marx Viana, and Daniel Mendez Fernandez. 2018. A Systematic Mapping Study on Security in Agile Requirements Engineering. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 454–461. DOI:https://doi.org/10.1109/SEAA.2018.00080

[56] Achmad Widodo and Bo Suk Yang. 2007. Support vector machine in machine condition monitoring and fault diagnosis. *Mechanical Systems and Signal Processing 21*, 2560–2574. DOI:https://doi.org/10.1016/j.ymssp.2006.12.007

[57] Thant Z. Win, Rozlina Mohamed, and Jamaludin Sallim. 2020. Requirement Prioritization Based on Non-Functional Requirement Classification Using Hierarchy AHP. *IOP Conference Series: Materials Science and Engineering* 769, 1 (February 2020), 012060. DOI:https://doi.org/10.1088/1757-899X/769/1/012060

[58] Sang Guun Yoo, Hugo Pérez Vaca, and Juho Kim. 2017. Enhanced Misuse Cases for Prioritization of Security Requirements. In *Proceedings of the 9th International Conference on Information Management and Engineering - ICIME 2017*, ACM Press, New York, New York, USA, 1–10. DOI:https://doi.org/10.1145/3149572.3149580

[59] SecReq dataset. Retrieved July 18, 2022 from http://www.se.uni-hannover.de/pages/en:projekte_re_secreq.

[60] NFR Software Requirements Dataset. Retrieved July 19, 2022 from https://zenodo.org/record/268542#.YtgvEHbMKUk