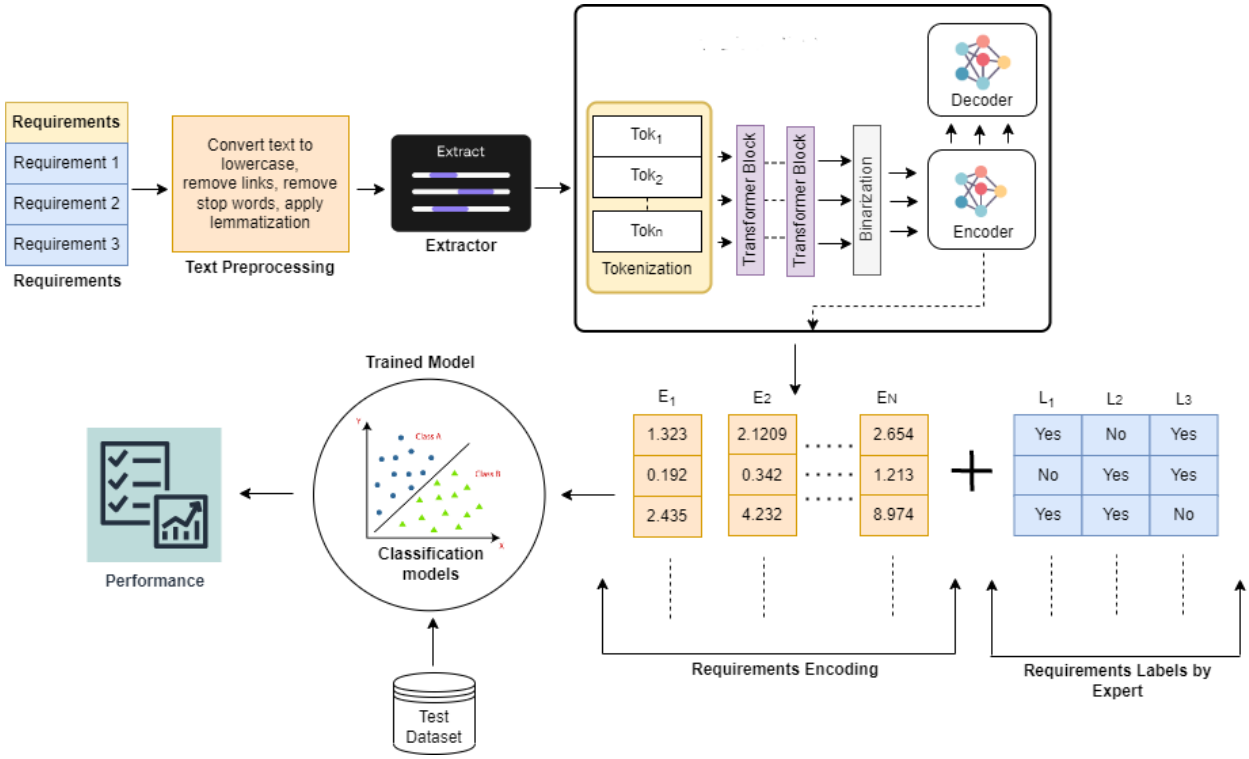
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Automated Requirements Review using Artificial Intelligence

Master's thesis in Computer science and engineering

SIVAJEET CHAND

CHANG LI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Automated Requirements Review using Artificial Intelligence

SIVAJEET CHAND
CHANG LI

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Automated Requirements Review using AI
SIVAJEET CHAND
CHANG LI

Supervisor:
Jennifer Horkoff, Department of Computer Science & Engineering
Cristina Martinez Montes, Department of Computer Science & Engineering
Advisor: Stefan Lindberg, APTIV
Examiner: Gregory Gay, Department of Computer Science & Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Figure shows the deep-learning architecture used

Typeset in LaTeX
Gothenburg, Sweden 2023

Automated Requirements Review using AI

SIVAJEET CHAND
CHANG LI
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Amidst the perpetually shifting landscape of automotive software development, this study sets out on a journey to outline the desirable properties that define an ideal requirement and subsequently harness these attributes to propel the semi-automation of the requirement review process. By utilizing these properties, the study endeavors to offer insightful feedback aimed at enhancing the quality of individual requirements. The heart of this endeavor lies in exploring machine learning to effectively automate the requirement review process.

This project has been performed in collaboration with an automotive company, APTIV. Several software companies, APTIV included, have embraced a requirement writing model that employs natural language as the preferred approach for articulating software requirements. The intrinsic nature of these requirements—often composed in natural language by various authors—renders them susceptible to errors, inconsistencies, and ambiguities. To mitigate these challenges, manual validation by developers and product owners ensues. However, this manual approach incurs escalated costs, resources, and exertion. It is further compounded by the potential for misinterpretation, leading to undesired software attributes. Thus, the quest for an innovative solution, one marked by consistency and automated review of handwritten requirements, becomes paramount.

This thesis endeavors to automate the requirement review process within the automotive industry by applying machine learning. Through the lens of automation, we delve into the complexities of requirement review. To gain insights into the manual requirement review practices of automotive experts, a series of interviews were conducted with nine professionals. These interviews focused on understanding the attributes they associate with requirements, their existing review strategies, and challenges stemming from poorly formulated requirements. The information gathered from these interviews suggests that unambiguity, consistency, and verifiability are the three most important requirement properties in the automotive industry and also laid the foundation for the subsequent model training.

Five distinct amalgamations of language models and classification techniques have been meticulously trained to predict whether requirements meet these three properties. Classical models—Support Vector Machine (SVM), Naive-Bayes, and Random Forest—converge with the CountVectorizer method. Furthermore, a SpaCy-based ensemble, encompassing Bag of Words and Convolutional Neural Networks (CNN), is designed for text classification. Additionally, the integration of an LSTM model

with Word Embedding via Keras and pad_Sequences has also been trained.

A careful assessment highlights that the SpaCy-based combination consistently achieves better F1 scores across various evaluation measures. Simultaneously, the LSTM-Word Embedding synergy manifests compelling results. Our finding suggests some combination of language models and classification techniques that can be used to automate the requirement review process and also try to evaluate the use of various word embedding techniques in the requirement review process.

Keywords: Computer, science, software engineering, automotive, requirements, machine learning, language models, thesis.

Acknowledgements

We extend our heartfelt appreciation to numerous individuals who have contributed directly and indirectly to the successful completion of this project.
First and foremost, we are immensely grateful to Aptiv for entrusting us with the opportunity to embark on this intricate and captivating thesis endeavor. Our gratitude knows no bounds as we extend our sincere thanks to Jennifer Horkoff, our supervisor at Chalmers University of Technology, and co-supervisor Cristina Martinez Montes. Their unwavering guidance and invaluable insights have been instrumental in shaping this thesis.
Stefan Lindberg, our supervisor at Aptiv, deserves a special mention for his exceptional understanding and continuous support. His provision of essential resources and clear directives during the thesis journey have been of immense help.
Our heartfelt thanks extend to the participants of our interviews at Aptiv, without whom our research would lack depth and relevance. Their willingness to share their insights has been invaluable.
Equally deserving of appreciation is our examiner, Gregory Gay, from the Department of Software Engineering. His support has been a pillar of strength throughout this thesis process.
Lastly, we wish to express our profound gratitude to our circle of friends. Their assistance, constructive critiques, unwavering morale, and constant encouragement have been a driving force during the entirety of this project.

Sivajeet Chand
Chang Li

Gothenburg, September 2023

Contents

List of Figures

List of Tables

1
Introduction

Requirements engineering is the branch of software engineering that deals with capturing stakeholders' needs and documenting them in the form of requirements. Requirements engineering can be summarised as a systematic and iterative process that involves cooperative analysis of a problem, documentation of the observations made, and verification of the accuracy of the gained understanding[1].

A requirement is a statement of something a system must do, how well it must do what it does, or a constraint that must be met [2]. Requirements are the foundation when stakeholders agree on what a software product needs to do. As such, it is important that they are well-written and easily understood. A well-written requirement is very important as it increases the probability of a successful system being developed [2]. Companies that are developing software for critical systems like automotive, aerospace, and medical sectors receive requirements coming from various sources like industrial standards, customer needs and procedures from internal quality management systems [4]. Most of the defects in delivered software originate because of deficient requirements and they are generally the most difficult and costly to repair [3]. As requirements gathering and analyzing is one of the initial stages when it comes to software development, it is very important to make sure that a quality check is performed at the start of the project. If we don't ensure that the requirements are entitled to certain quality criteria at the start of the project then it will be difficult to maintain the quality of the project in the later development stages. It will also be difficult to track back to the root cause of the problem.

The quality of a requirement can be ensured by making sure that the requirement meets certain desirable qualitative properties. According to the International Standard IEEE, each stakeholder, system, or software requirement should possess certain quality characteristics, i.e. necessary, appropriate, singular, unambiguous, complete, feasible, verifiable, correct, and confirming [5]. Though these characteristics are well-defined, quantifying these characteristics' value in a requirement is a difficult task [17]. Using natural language as a privileged form of communication among stakeholders and writing requirements, as opposed to a formal form which is generally not convenient for some stakeholders, i.e clients makes it even more difficult to quantify.

One of the major factors in deciding the quality of a requirement must be the evaluation of involved domain experts as their way of judging the quality of a requirement is strongly linked to the needs and necessities of the project. Most of the automated tools and techniques currently in use are yet to consider the experts' interpreta-

tion of quality while defining the quality level of a requirement [17]. Despite being one of the most common ways of reviewing requirements, manual review is a time-consuming and resource-intensive procedure.

The use of machine learning has helped automate and has shown practical importance in various domains. In software engineering, machine learning is used to automate different tasks [24]. This thesis work will investigate the use of machine learning to automate the requirement review process. We will present a methodology to evaluate the quality of requirements in an automatic way, based on the qualitative characteristics posed by the automotive domain experts. The experts will contribute initially to deciding on what quality properties are most important from their perspective in the automotive domain. We have interviewed 9 automotive domain experts.

In our pursuit to revolutionize requirement assessment, we are harnessing the power of machine learning models in tandem with natural language processing techniques (NLP) to effectively classify software requirements. These NLP techniques enable the textual requirements to be represented in a numerical vector form that the machine learning models get trained with. This method allows us to use machine learning to find detailed patterns in requirements and take advantage of the strong language understanding of NLP techniques. These model combinations have been trained with automotive requirements data which have been labeled by automotive experts based on various requirement properties.

Machine learning models, renowned for their ability to gain insights from data, are employed to learn and recognize the distinguishing features that characterize different requirement categories. These models try to extract features from the requirements and utilize this acquired knowledge to make accurate classifications.

Large-Language Models models have undergone extensive training on diverse textual data, endowing them with a profound understanding of language nuances, context, and semantics. By utilizing LLMs, we empower our system to understand the complex layers of meaning embedded within requirements, transcending mere keyword matching.

In this work, the combination of machine learning models with natural language processing techniques have been used to automate the requirement review process in automotive industry. This innovative approach holds the potential to streamline and enhance the accuracy of requirement review process, a cornerstone in the pursuit of more efficient software development processes.

1.1 Study Context

This thesis is being done in collaboration with Aptiv. Aptiv, an automotive technology supplier, receives thousands of requirements from its client companies. The need for communication among all stakeholders has demanded the use of natural

language as the form of writing requirements, as opposed to a formal form of language that might be inaccessible to some of the clients. As requirements are written using natural language, usually written by hand and by different people, they are very prone to mistakes, inconsistency and ambiguity. The requirements are manually validated by developers and product owners owing to increased costs, efforts and resources. They are also sometimes misunderstood or misinterpreted by the developers, resulting in unwanted software features in the later development phases. Therefore, it is highly important to come up with a solution that will be consistent and will efficiently and automatically review the hand-written requirements.

After some research, it was realized that there are some existing solutions for this task but most of these solutions use requirements as input but ignore domain-specific experts' opinions in the solutions suggested. As the work is related to a critical sector like automotive, it is of utmost importance to come up with a solution that automatically reviews requirements and tries to emulate an expert by using experts' knowledge while designing the solution.

1.2 Purpose of the Study

The purpose of this study is to determine the desirable properties of an automotive software requirement and to use these properties to semi-automate the requirement review process. Furthermore, this study aims to enhance the efficiency of the requirement review process by providing feedback to the requirement writers on how to improve their requirements. This feedback will help the requirement writers to understand the desirable properties of an automotive software requirement. This has been done by taking inspiration from the existing works to automatically review requirements [56][17] and to make the automated requirement review process suggested in this study aligned to how automotive experts review requirements currently, we have integrated the opinions of experts in the study in the form of interviews and requirements data labeling. An extensive study has been conducted to explore potential approaches for automating the requirements review process using machine learning and improving its efficiency.

The strategy which we have proposed is built upon the existing strategies in place to review requirements automatically. The proposed strategy has been evaluated and compared with existing strategies [56] [17] to demonstrate its effectiveness. The implementation of the proposed strategy is expected to enhance the efficiency of the requirement review process and help automotive software companies to produce better-quality software requirements.

1.3 Research Questions

The problem is to find a way to automatically access the quality of a requirement by verifying if a requirement meets certain desirable properties, which will emulate the way software experts access the quality. As a result, the following research questions

arise:

RQ1: What requirements properties should be used to judge software requirements in the automotive sector?

RQ2: Can the combination of various Natural Language Processing (NLP) techniques with machine learning models, such as CountVectorizer with Support Vector Machine (SVM), Random Forest, and Naive Bayes, Bag of Words with Convolutional Neural Network (CNN), and Word Embedding with Recurrent Neural Network (RNN), be used to automate the requirement review process? Furthermore, which combination of NLP technique and machine learning model yields the highest F1 score for this task?

1.4 Significance of the Study

Overall, the study aims towards simplifying the requirement review process, which has relevance for both researchers and practitioners. Our approach builds upon the existing work to find a good solution to automatically find requirements quality issues. Our approach also uses domain-specific experts' knowledge in our method to evaluate the requirement quality. An improvement in existing methodologies to review requirements will also help practitioners spend less time and effort on reviewing requirements.

1.5 Structure of the Thesis

In this paper, Chapter 2 gives the background of this study. Chapter 3 discusses and introduces the related work of this study. Chapter 4 detailed illustrates the research methodology used and how it was applied in this study. Chapter 5 shows the results of this paper. Chapter 6 is the discussion section and Chapter 6 provides a summary of this study.

2

Background

This chapter provides foundational information about the concepts this study used. It begins with a section discussing the basic ideas of requirements engineering and requirements properties. After that, it follows with discussion on the basics of Natural language processing and how it is useful in our work. We have propvided an introduction to machine learning and the machine learning algorithms we are using in this work. We have also discussed about deep machine learning algorithms that have been used in this work. We end this section by discussing the evaluation methods.

2.1 Requirement Engineering

Requirements Engineering is a systematic and disciplined approach within software engineering that focuses on identifying and comprehending the needs of stakeholders and subsequently documenting them in the form of requirements [67]. This process typically involves five primary phases in requirements engineering including requirement elicitation, requirement analysis, requirement specification, requirement validation, and requirement management.

To be more specific, requirement elicitation involves gathering and identifying the requirements from various stakeholders, such as end-users, customers, managers, and domain experts. Different techniques like interviews, surveys, workshops, and observation are used to capture their needs and expectations. Once the requirements are gathered, they need to be analyzed to ensure clarity, consistency, and feasibility. The requirement analysis step involves prioritizing requirements, resolving conflicts, and refining them to make them more specific and measurable. In the requirement specification phase, the requirements are formally documented in a clear and unambiguous manner. The documentation may include use cases, user stories, functional and non-functional requirements, data models, and diagrams to help the development team understand the system's desired behavior. In the fourth phase, requirement validation is used to review and validate the documented requirement to ensure that they accurately represent the stakeholders' needs and are feasible to implement. Requirement validation helps to identify any potential issues or misunderstandings early in the development process, reducing the risk of costly errors later on. The last phase is requirement management. In this phase, it involves tracking changes to requirements, ensuring traceability between requirements and the final product, and maintaining the integrity of the requirements throughout the

development process [67].

The primary goal of requirements engineering is to bridge the gap between the stakeholders' requirements and the actual software solution that will be developed. Requirements engineering can be summarised as a systematic and iterative process, which entails collaborative problem analysis, documentation of observations, and verification to ensure an accurate understanding of the requirements [1].

2.2 Requirements Quality

A requirement represents a statement outlining what a system must accomplish, the level of performance it should exhibit, or any constraints it must adhere to [2]. These requirements serve as the building blocks for stakeholder consensus on the desired functionality of a software product. Therefore, it is crucial that the requirements are well-written and easy to understand. Well-written requirements play a vital role in enhancing the likelihood of successfully developing the intended system [2]. A well-written requirement lays the foundation for a successful software development process, reducing the risk of misunderstandings, rework, and costly errors. It enhances communication and collaboration among stakeholders and the development team, leading to the creation of a high-quality software product that meets the intended objectives. Most of the defects in delivered software originate because of deficient requirements and they are generally the most difficult and costly to repair [3].

The requirement quality can be ensured by making sure that the requirement meets certain desirable qualitative properties. According to the International Standard IEEE, each stakeholder, system, or software requirement should possess certain quality characteristics, i.e. necessary, appropriate, singular, unambiguous, complete, feasible, verifiable, correct, and confirming [5].

- necessary: The requirement defines a fundamental capability, characteristic, constraint and/or quality factor.
- appropriate: The specific intent and amount of detail of the requirement is appropriate to the level of the entity to which it refers.
- singular: The requirement requires a single capability, characteristic, constraint or quality factor.
- unambiguous: There exists only one interpretation for each requirement (unambiguity) and understandability is interrelated, they could be even the same property.
- complete: All needs are covered by the set of requirements.
- feasible: The requirement can be met while adhering to system limitations such as budget, timeline, and technical considerations, all within an acceptable level of risk.
- verifiable: The requirement can be realized and can be verified to the customers' satisfaction.
- correct: The requirement precisely reflects the original entity need from which it was derived.

- consistent: The collection of requirements comprises distinct elements that are non-repetitive, devoid of contradictions or overlaps among themselves. Additionally, the units and measurement systems are consistent throughout, and the terminology employed within the set of requirements remains uniform.
- confirming: The individual items adhere to an approved standard template and writing style for requirements.

2.3 Natural Language Processing

Natural Language Processing (NLP) is a branch of artificial intelligence and computational linguistics that focuses on how computers can understand and process natural language text or speech [18]. It involves developing algorithms and models that enable machines to understand, interpret, generate, and respond to natural language text or speech. Natural language is the form of communication used by humans in daily interactions. In our work, Natural Language Processing(NLP) techniques are used to enhance the requirement properties detection such as unambiguity within the automotive domain. By harnessing NLP methodologies, we extract meaningful insights from textual data, capturing linguistic nuances and context.

NLP processing can be divided into lower levels, which involve morphological analysis, syntactic analysis, and semantic mapping, and higher levels, which encompass discourse and pragmatic analysis [19] [20]. There are two main phases to natural language processing: data preprocessing and algorithm development [21]. Figure 2.1 shows a brief phase of data preprocessing.
As shown in Figure 2.1, This phase involves various methods used for data preprocessing. Tokenization is used to split text into meaningful words or tokens. The Stopword Removal method removes common words(stopwords) from texts that do not carry much meaning such as "a", "an", and "the", etc to reduce noise and improve processing efficiency. In the phase of Stemming, words are reduced to their base or root form. Lemmatization produces valid words while stemming produces word stems, which may not always be valid words but still help in reducing inflections. For Part-of-Speech(POS) Tagging, each word in the text is tagged with its grammatical part, such as a noun, verb, adjective, etc. This information is crucial for understanding the structure of sentences. Once the text is preprocessed, the second phase of NLP involves developing and applying algorithms to perform specific tasks on the preprocessed data.

2.3.1 SpaCy

SpaCy is an open-source natural language processing library designed to process and analyze human language data in a fast and efficient manner. SpaCy uses a CNN-based deep neural network with some adjustments [25]. SpaCy is widely used for a variety of NLP tasks; for example, it can perform part-of-speech analysis, named entity recognition, dependency relationship characterization, and word embedding vector calculation and visualization for natural language texts, etc [25]. Figure 2.2 shows the pipeline of SpaCy.

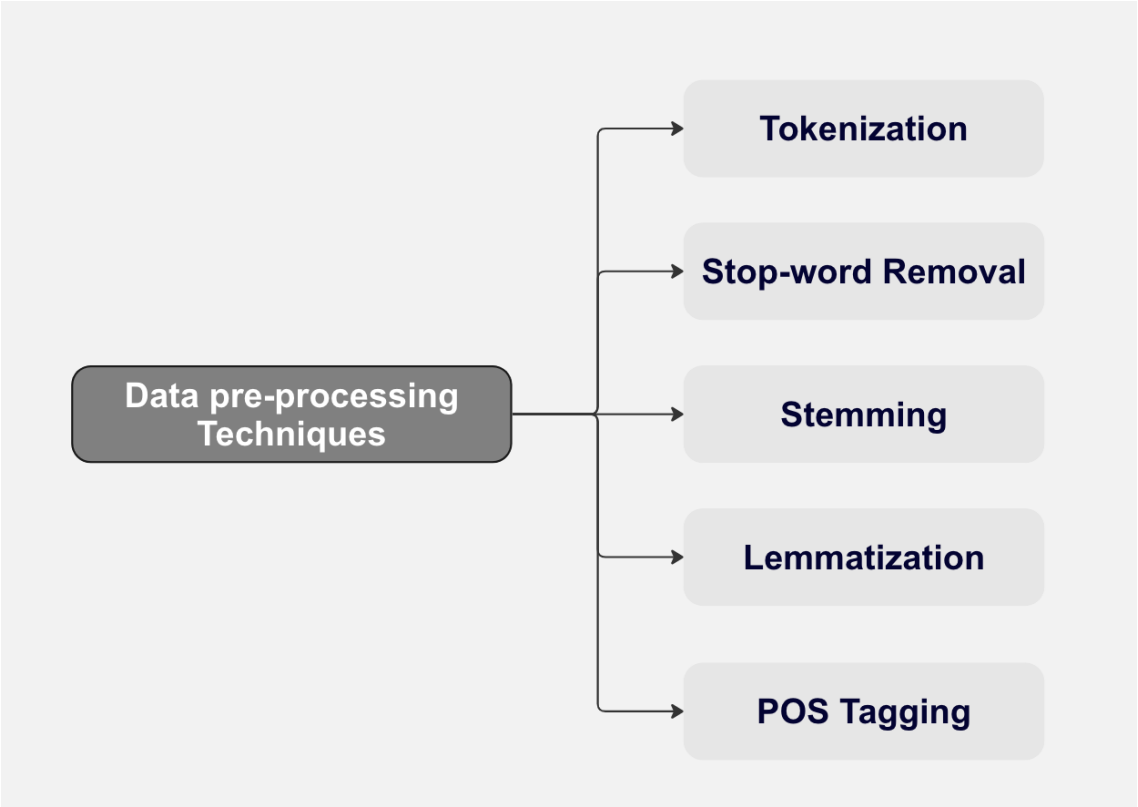


Figure 2.1: Phases of Data Preprocessing

As shown in Figure 2.2, When using spaCy, spaCy first tokenizes the text to produce a Doc object. The Doc is then processed in several different steps – this is also referred to as the processing pipeline. The pipeline usually includes a tagger, a lemmatizer, a parser, and an entity recognizer. Each pipeline component returns a processed doc, which is then passed to the next component to form the final doc.

2.3.2 CountVectorizer

CountVectorizer is used to convert a collection of text documents to a matrix of token counts on the basis of the frequency (count) of each word that occurs in the entire text [79], it is commonly used in NLP and machine learning tasks. For each

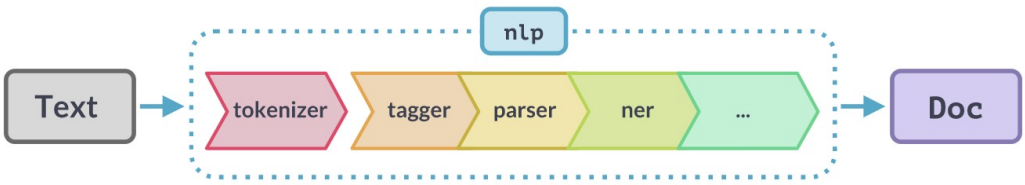


Figure 2.2: SpaCy pipeline

input text, it only considers the frequency of each vocabulary in that training text. There are three steps how to use CountVectorizer.

- Tokenization: Textual data is initially segmented into smaller components called tokens. Depending on the chosen settings, these tokens can encompass words, phrases, or even individual characters.
- Counting: For each document, the CountVectorizer counts the occurrences of each token.
- Creating the Matrix: The resulting matrix is an arrangement where rows represent documents, and columns represent the vocabulary token. The entries in this matrix denote the frequency of each token's appearance within the corresponding document.

Overall, CountVectorizer is a direct way to represent text data numerally. However, CountVectorizer doesn't consider the sequence of words or the underlying meaning of words, it only focuses on the frequency of words [79].

2.4 Machine Learning

Machine learning(ML) is the technique that improves system performance by learning from experience through computational methods and is a subfield of artificial intelligence (AI) [9]. Usually, the main task of machine learning is to develop learning algorithms that build methods from data. Machine learning can be categorized into various approaches, including supervised learning, unsupervised learning, and reinforcement learning, each addressing different aspects of learning from data [9]. Machine learning finds a wide array of applications, spanning image and speech recognition, natural language processing, recommendation systems, medical diagnosis, autonomous vehicles, fraud detection, and beyond. Its ability to automate intricate tasks and derive predictions or decisions from data-derived insights positions it as an essential element within contemporary technology and AI frameworks.

Within the scope of this thesis, we are engaged in a comprehensive assessment of three distinct supervised machine learning classification models: SCVM, Naive Bayes and, Random Forest. This section delves into the detailed explication of the aforementioned models we have incorporated into our study:

2.4.1 Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable and dependent feature vector through [30].

$$P(y \mid x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n \mid y)}{P(x_1, ..., x_n)} \tag{2.1}$$

where y means class variable, $x1$ through xn refers to dependent feature vector. Using the naive conditional independence assumption, the formula is as follows:

$$P(x_i \mid y, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i \mid y) \tag{2.2}$$

For all i, this relationship is simplified to

$$P(y \mid x_1, ..., x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, ..., x_n)} \tag{2.3}$$

Since $P(x_1, ..., x_n)$ is constant given the input, then we can use the following classification rule.

$$P(y \mid x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{2.4}$$

the formular2.4 can be converted to

$$\hat{y} = argmax P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{2.5}$$

Then the Maximum A posterior(MAP) estimation can be used to estimate $P(y)$ and $P(x_i \mid y)$, where $P(y)$ is then the relative frequency of class y in the training set[30].

Naive Bayes is a classification algorithm commonly used in binary classification tasks. Naive Bayes computes the probabilities of an input belonging to different classes and assigns it to the class with the highest probability [9]. Despite its seemingly oversimplified assumptions, naive Bayes classifiers have demonstrated effectiveness in various real-world scenarios, notably in tasks like document classification and spam filtering [30]. It's computationally efficient and can handle high-dimensional data with ease.

2.4.2 Support Vector Machine

Support Vector Machines (SVM) are powerful and widely used machine learning algorithms, primarily employed for binary classification tasks. The core principle behind SVM is to find the optimal hyperplane that maximizes the separation margin between data points from different classes. This hyperplane serves as the decision boundary, effectively categorizing new data points based on which side of the hyperplane they fall.[10]

The concept of "margin" is pivotal in SVM. The margin is the distance between the decision boundary (hyperplane) and the nearest data points from each class. SVM seeks to find the hyperplane that has the largest margin, as this leads to a more robust and generalizable model. The Figure 2.3 shows the decision function for a situation where the data is linearly separable, and it highlights the critical role

of three specific samples positioned precisely on the boundary. These samples are called Support vectors. These support vectors play a critical role in defining the optimal hyperplane and, by extension, the decision boundary.[42]

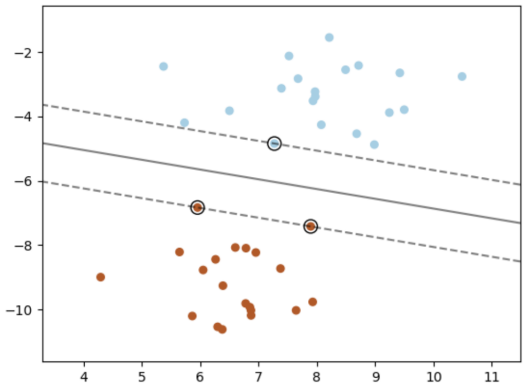


Figure 2.3: Suport vectors in SVM[10]

SVM is particularly effective when dealing with linearly separable data, where a clear gap exists between the two classes. However, SVM can also handle non-linear data through the use of kernel functions. Kernel functions transform the original feature space into a higher-dimensional space, potentially making the data linearly separable. Common kernel functions include the Radial Basis Function (RBF) kernel, polynomial kernels, and sigmoid kernels. The choice of the kernel depends on the specific characteristics of the data and the problem at hand.

2.4.3 Random Forest

The core idea behind Random Forest is to build a collection of decision trees, each trained on a different subset of the data and featuring random variations. By combining the predictions of these individual trees, Random Forest provides a more robust and accurate overall prediction.[48]

The key concept that underpins Random Forest's effectiveness is the ensemble technique. Ensemble methods combine the outputs of multiple models to create a more stable and accurate prediction. In the case of Random Forest, this ensemble is achieved by constructing an ensemble of decision trees, often referred to as a "forest."[48]

Each decision tree in the Random Forest is constructed using a bootstrap sample from the original dataset, which means that each tree sees a slightly different subset of the data. Additionally, at each split within a decision tree, a random subset of features is considered, leading to diverse decision trees. This randomness helps prevent overfitting, as the individual trees capture different aspects of the data.

During prediction, each decision tree in the Random Forest independently produces its own result, and the final prediction is determined by a majority vote (in the case of classification) or an average (in the case of regression) of the predictions from individual trees. This ensemble-based approach enhances the model's ability to generalize well to new, unseen data.[48]

2.5 Deep Learning

Deep learning is a specialized branch of machine learning. It is a methodology based on the concept of Artificial Neural Network (ANN) that has been gaining a lot of attention as it is producing quality results [22]. It is often referred to as deep neural networks, to learn complex patterns and representations from data. It is a specialized form of machine learning that excels at handling vast amounts of data and tackling tasks that require a high level of abstraction and hierarchical feature learning. Deep learning has achieved remarkable success in various tasks, including image and speech recognition, natural language processing, machine translation and etc. The ability to automatically learn hierarchical and abstract representations from large datasets has made deep machine learning a powerful tool for solving real-world problems across diverse domains. In our study, we have used deep learning techniques like Convolution Neural Network(CNN) and Recurrent Neural Networks(RNN).

2.6 Convolutional Neural Network

The Convolutional Neural Network (CNN) is one of the most popular deep neural networks, named after the mathematical operation of convolution between matrices. CNNs exhibit exceptional performance in various machine learning problems such as the renowned ImageNet classification dataset, computer vision tasks, and in natural language processing (NLP) [23]. CNN has multiple layers including convolutional layers, non-linearity layers, pooling layers and fully-connected layers. The convolutional and fully-connected layers have parameters but pooling and non-linearity layers don't have parameters.

Figure 2.4 shows the main structure of CNN architecture.

- Convolutional Layers: The core building blocks of a CNN are convolutional layers. Each layer consists of multiple filters (also called kernels), which are small windows that slide over the input image. The filters detect specific features, such as edges, textures, or patterns, in different regions of the image. As the filters slide across the image, they produce feature maps that capture the presence of these features.
- Activation Function: After the convolution operation, an activation function (commonly ReLU - Rectified Linear Unit) is applied element-wise to introduce non-linearity. This non-linearity is crucial to enable the network to learn complex relationships in the data.
- Pooling Layers: Pooling layers downsample the feature maps to reduce the spatial dimensions and the number of parameters in the network. The most

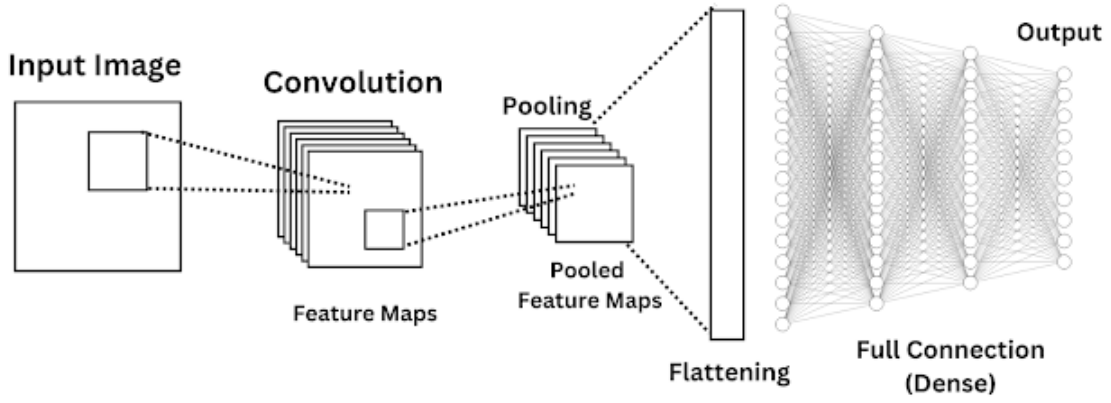


Figure 2.4: CNN architecture from the article [77]

common pooling operation is max pooling, which takes the maximum value within a specified region of the feature map. Pooling helps in reducing computation and making the network more robust to small spatial variations.

- Fully Connected Layers: After several convolutional and pooling layers, fully connected layers are used to combine the learned features and make final predictions. These layers are typical in traditional neural networks and help in performing high-level reasoning.
- Output Layer: The last layer of the CNN is the output layer, which provides the final prediction based on the learned features. For image classification tasks, the output layer typically uses softmax activation to produce a probability distribution over different classes.

In our work, we employ a SpaCy model that incorporates Convolutional Neural Networks(CNNs) to perform requirement review.

2.7 Recurrent Neural Networks (RNN)

Recurrent Neural Network(RNN), is a type of artificial neural network and is a generalization of feedforward neural network that has an internal memory. Unlike traditional feedforward neural networks, RNNs have connections that loop back on themselves, allowing them to maintain a memory of previous inputs as they process new ones. It performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network [11]. For making a decision, it considers the current input and the output that it has learned from the previous input.

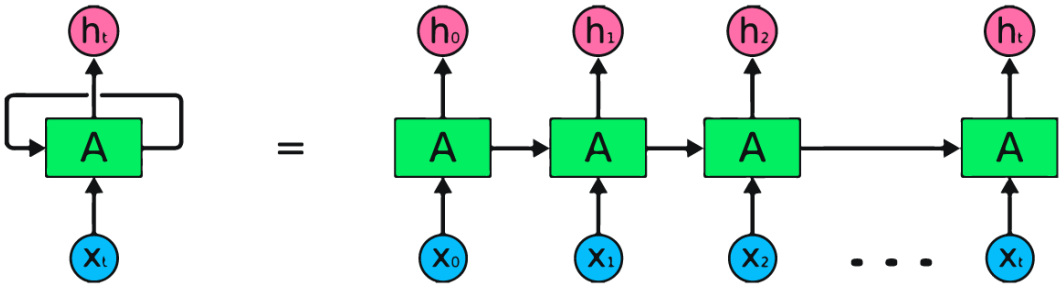


Figure 2.5: Unrolled Recurrent Neural Network from the article [76]

Figure 2.5 shows the basic RNN structure, where x_t is input at each time step t, h_t is hidden state at time t, First of all, it takes the X_0 from the sequence of input and then produces the output h_0, which together with X_1 is then the input for the next step. That is to say, the $h_0$and X_1 are the input for the next step. Similarly, the resulting h_1 and the following input X_2 become the input for the next step, and so forth. This way, it keeps the model to remember the context while training.

The key advantage of RNN is that it has the ability to capture temporal dependencies in sequential data [11]. This is achieved through the recurrent connections that enable information to flow one time step to the next, in this way allowing the network to reatil information from the earlier time steps while considering the current input. However, basic RNNs suffer from the "vanishing gradient" problem, where gradients used for learning diminish over time steps, making them unable to effectively capture long-range dependencies [14]. LSTM is an advanced version of RNN, that has been used in our work.

2.7.1 Long Short-Term Memory LSTM

To address the basic RNN shortage such as the "vanishing gradient", various more advanced RNN architectures have been developed, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) [11]. These architectures have specialized mechanisms to better capture and manage long-term dependencies, making them more suitable for practical applications. Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, the structure is as followed [74].

The LSTM structure consists of a Cell State and three gates: Forget Gate, Input

Gate, and Output Gate, the input gate formula is as follows:

$$\begin{aligned} i_t &= \sigma(W^i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= tanh(W_c \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{2.6}$$

where i_t is the input gate, $\tilde{C}_t$ is the cell state of LSTM. The input gate decides which new information should be added to the cell state based on the input at the current time step and the hidden state at the previous time step. It weights and sums the input data and the hidden state, and passes through a Sigmoid activation function to output a value between 0 and 1, indicating the importance of each input[14].

For the forget gate, the formula is as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.7}$$

The forget gate decides which old information should be forgotten or ignored based on the input of the current time step and the hidden state of the previous time step [14]. It weights and sums the content input data x_t and the hidden state h_{t-1}, and passes through a Sigmoid activation function to output a value between 0 (omit this)and 1(keep this), indicating the degree to which each element in the cell state C_{t-1} should be retained.
The Output gate formula presents as follows:

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * tanh(Ct) \end{aligned} \tag{2.8}$$

The output gate decides which information in the cell state should be passed to the next layer or the output layer based on the input and hidden state at the current time step. The sigmoid function decides which values to let through 0,1, that is o_t. h_t refers to the output, by means of the tanh activation function gives weightage to the values which are passed deciding their level of importance ranging from -1 to 1 and multiplied with output of Sigmoid.

Overall, the advantage of LSTM is that it can effectively handle long sequences and long-term dependencies through the gating mechanism, and avoid the problem of gradient disappearance or gradient explosion [14]. It has been widely used in tasks such as machine translation, language models, and speech recognition, and has become an important variant of recurrent neural networks.

2.8 Evaluation Methods

Binary classification is a type of supervised machine learning task where the goal is to classify input data into one of two distinct classes or categories. The term "binary" refers to the fact that there are only two possible outcomes or labels for each data instance. Some common binary classification tasks include spam detection,

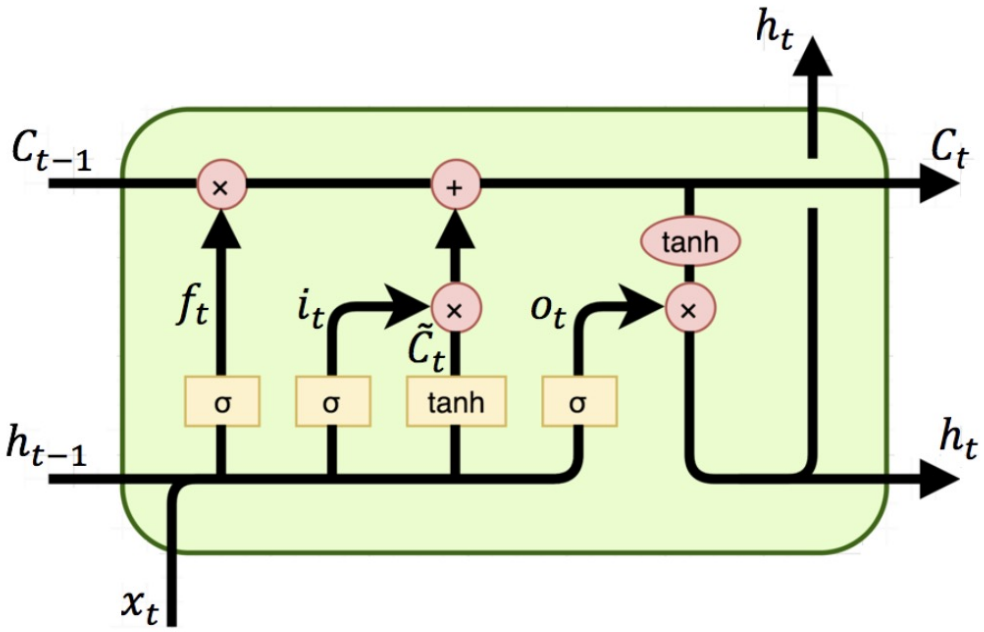


Figure 2.6: An illustration of the process done inside an LSTM memory from the article [13]

sentiment analysis, medical diagnosis, etc. For binary classification models, there are several metrics to evaluate the performance of models such as Accuracy, Precision, Recall, and F1 score. First of all, the confusion matrix [27] is the key way to evaluate model performance. Figure 2.7 introduces the confusion matrix.
As shown in Figure 2.7, the vertical axis represents the real label value, and the horizontal axis represents the label value predicted by the binary classifier. There are four key different concepts in the confusion matrix.

- True Positive(TP): The real label is true and the classifier predicts "True".
- True Negative(TN): The real label is true and the model predicts "False".
- False Positive(FP): The real label is False and the model predicts "True"
- False Negative(FN): The real label is False and the classifier predicts "False".

2.8.1 Accuracy

Accuracy refers to the ratio of the number of samples correctly predicted by the model to the total number of samples [28]. For accuracy score formula is as follows:

$$Accuracy = \frac{TP + FN}{TP + FP + FN + TN} \tag{2.9}$$

From the formula 2.9, where $TP + FP$ means the number of labels where the model predicts correctly. $TP + FP + FN + TN$ means the total number of labels. The higher the accuracy rate, the better the performance of the model. However, the

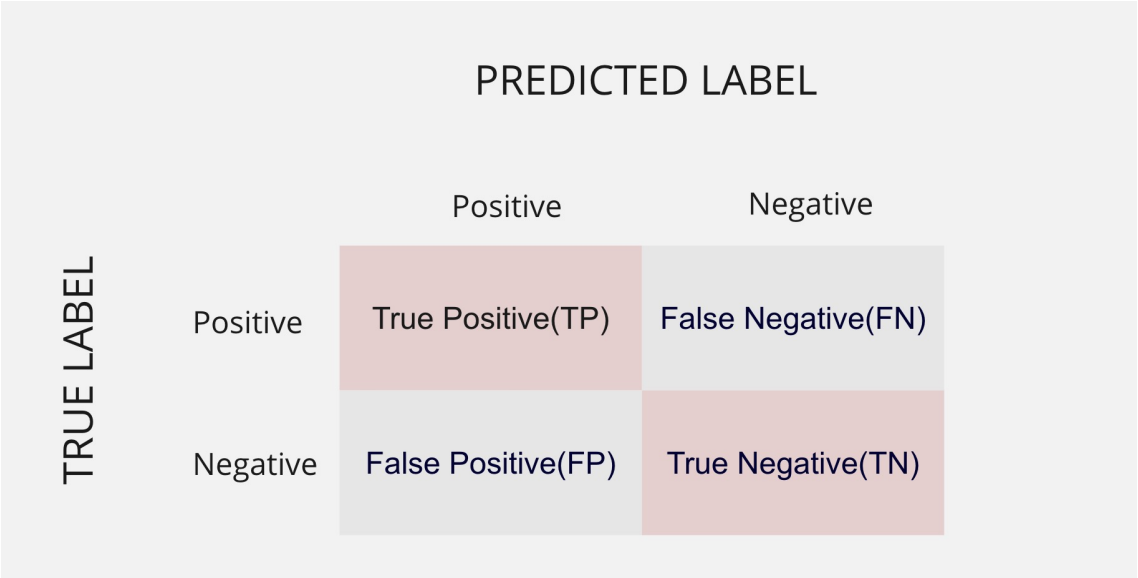


Figure 2.7: Confusion Matrix

accuracy rate does not fully represent the model's performance, especially in an imbalanced dataset [28], the accuracy rate may be dominated by some categories with a large number of samples while ignoring other categories. For example, when performing spam detection, normal email occupies the majority.

2.8.2 Precision

Sometimes the accuracy could not reflect the performance of the model especially for the imbalance dataset. Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive. Precision signifies the accuracy of the model's positive example predictions [28]. The precision can be described with the below equation.

$$Precision = \frac{TP}{TP + FP} \tag{2.10}$$

Where TP means the real label is true and the classifier predicts "true". $TP+FP$ in the mathematical formula represents the total number of positive examples predicted by the model. Precision is a good measure to determine when the cost of False Positive is high such as spam detection [28].

2.8.3 Recall

Recall, also known as true positive rate or sensitivity, measures the proportion of actual positive instances that were correctly identified by a model[29]. In mathematics formula is as follows:

$$Recall = \frac{TP}{TP + FN} \tag{2.11}$$

where the TP means the real label is true and the classifier predicts "true", and $TP + FN$ refers to total actual positive labels. Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative, such as in medical diagnoses or fault detection systems.

2.8.4 F1 score

The F1-score is a metric that combines both precision and recall, it provides a balanced assessment of a model's performance in binary classification tasks[28] [29]. It considers both the number of correctly predicted positive instances (precision) and the model's ability to capture all positive instances (recall). The F1 score is defined in mathematics as follows:

$$F1score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.12}$$

Where *Precision* means precision score and *Recall* means recall score. The F1 score is calculated using the harmonic mean of precision and recall, giving equal weight to both metrics [29]. It is especially useful when the class distribution is imbalanced or when false positives and false negatives have different consequences. A high F1 score signifies a well-balanced relationship between precision and recall, whereas a low F1 score indicates a notable disparity between these metrics. The F1 score is particularly valuable in scenarios with imbalanced class distributions or varying consequences of false positives and false negatives [29]. By considering both precision and recall, the F1-score prevents scenarios where a model might seem successful based on one metric while lacking balance due to class distribution imbalances.

3
Related Work

Software Engineering has demonstrated its compatibility and effectiveness for automation, with various techniques and approaches from the field of Machine Learning being increasingly integrated into numerous processes throughout the software lifecycle [15]. Notably, tools and frameworks are now placing emphasis on automating activities within the Requirements Engineering subdisciplines such as elicitation, analysis, specification, and validation [16], aiming to provide automated assistance in these areas.

Some papers are related to requirements, machine learning, and the intersection of software requirement engineering (RE) and Machine Learning(ML). This section briefly summarises some of these papers.

3.1 Requirement Engineering

Some existing work discusses the requirement qualities. The paper [8] introduces a newly established standard and explores the underlying principles and motivations behind its development. The authors encourage users of the standard to actively engage with the standards committee by providing feedback and valuable insights. This article is a significant contribution to a special issue dedicated to quality requirements, emphasizing the importance of ensuring high-quality standards in this domain.

Extensive work has been done to determine the quality of requirements. Heck and Zaidman [49] have proposed 28 different quality criteria for agile requirements specifications. Génova, Fuentes, Llorens [26] proposed a framework for evaluating and improving the quality of textual requirements in software engineering. It has presented a systematic relationship between 11 desirable properties like atomicity, precision, completeness, etc., and 15 quantitative measurable indicators (metrics) like size, readability, punctuation, etc. This paper illustrates five parts that are Quality Criteria, Quality Indicators, Evaluation Techniques, and Improvement Strategies. For Quality Criteria, this paper identifies five quality criteria for textual requirements: completeness, consistency, clarity, correctness, and verifiability. For each quality criterion, the authors propose several quality indicators that can be used to measure the degree to which a requirement meets the criterion. The authors suggest several techniques for evaluating the quality of requirements, including manual review, automated tools, and stakeholder feedback.

3.2 Requirement quality analysis

Recent work [24] provides a useful introduction to the intersection between Requirements Engineering (RE) and Machine Learning (ML) and highlights the potential benefits and challenges of using machine learning techniques in this field. More specifically, in the software domain, ML is mainly used to help automate different tasks. RE tasks contain requirements elicitation, analysis, validation, and management. At the same time in the software requirement domain, ML can be used in the RE tasks. There are two main contributions, the paper [24] starts by discussing the challenges of RE and the benefits of using ML techniques to address these challenges. It then provides a brief overview of ML techniques and common ML models such as decision tree, K-Nearest Neighbors (KNN), Naive Bayesian, etc, and their applications in various RE tasks. Secondly, It provides a survey of the current state-of-the-art in this field and identifies several future research directions such as improving the quality of requirements data, developing more interpretable machine learning models, and addressing ethical and legal issues related to using machine learning in software development.

Some studies have also used ML to analyze requirements' quality. Some papers focus on a single requirement property such as unambiguity, incompleteness and etc. Chantree and Roeck [52] have proposed a method that focuses on the identification and analysis of unambiguity in natural language requirements. It uses techniques such as linguistic analysis, or a combination of methods to detect and classify ambiguous expressions or phrases in requirements documents.

One paper [80] focuses on using the decision tree to detect ambiguity for software requirements. There are four steps. Firstly, the researchers labelled the training data manually. Then the training data is fed and trained into the fuzzy decision tree to build classifier. In the third step, unlabelled instances were estimated by using the classifier which is trained in the previous step for classifying ambiguous instances. In the final step, the highest classification ambiguous instances were selected and then deleted. Moreover, in cases where the collected instances were lower than predetermined thresholds, the researchers would select the next instance by repeating the whole four steps.

Sharma [81] proposed using Naive Bayes classifier to detect ambiguity in requirements. First of all, the bag of words(BOW) was used to tokenize the training data, and stop words and unique words were removed. The next step is to use n-gram model to get sequences of words focusing on using unigrams and bigrams with Part of Speech (POS) tagging. Finally, the Naive Bayes classifier was used to detect ambiguity in requirement documents by classifying the n-grams documents based on POS tagging. The paper [82] compared different machine learning classifiers' performance when detecting coordinating ambiguity in natural language. The coordinating ambiguity refers to more than one "and "," or" is used within one sentence. After comparing the performance of the different classifiers such as Naive Bayes, K-NN, Random Forest, and Random Tree, Naive Bayes got higher accuracy in de-

tecting coordinating ambiguity.

The study [83] also proposed the creation of a pattern recognition system to identify ambiguous patterns, employing Support Vector Machine (SVM) technology. With the help of SVM, every element was covert into distinct features, and independent classification. Meanwhile, ambiguity within handwriting digit recognition was successfully detected by using SVM. Ultimately, the ambiguity was identified by looking for classes that have a greater number of features in general.

Another paper [33] highlights the critical role of ambiguity identification and measurement tools in improving the quality of software systems and presents a set of requirements for developing such tools, and the model learns from past experiences.

There are some other works that focus on measuring whether requirements meet the properties or not. The work [35] proposes a method for using classification techniques to improve the analysis of informal requirements in a requirements analysis-supporting system, which can help ensure that the requirements are complete, consistent, and unambiguous. Compared to the works above, this work focuses on classifying requirements based on various properties, rather than only one property. The work presented in [56] applies case-based reasoning and neural networks. Case-based reasoning is an experience-based approach to solving problems where a solution to a previously solved similar task is used to solve a current issue in hand. In the work [17], machine learning techniques have been used to train a classifier to assess the quality of requirements. It introduces a methodology that harnesses machine learning techniques to automate the classification process. By training models on labeled datasets, the methodology enables the objective assessment of requirement quality, facilitating informed decision-making and improving overall project success. Aceituna and Walia [54] present a model-based requirements verification method, called NLtoSTD, which transforms natural language requirements into a State Transition Diagram (STD) that can help to detect and to eliminate ambiguities and incompleteness.

Our work will be building on existing work to find a good solution to automatically find requirements quality issues. The above works [81][82][80] have explored the use of various classical machine learning models like Decision Tree, Naive Bayes, K-NN, Random Forest, Random Tree and SVM. While our work has similarities with other works, including our use of various machine learning models such as Naive Bayes, Random Forest, and Support Vector Machine, what truly distinguishes our study is our concentrated attention on the automotive sector. This unique domain demands an approach tailored to the intricacies of its language nuances and specialized terminologies.

Furthermore, we place a strong emphasis on the integration of deep machine learning models, particularly Convolutional Neural Networks (CNNs), which represents a departure from the methods discussed in the cited papers [81][82][80]. This decision aligns with the complex nature of requirements, as requirements are written by

different people, using different formats and natural language.

The work utilizing a Decision Tree [80] uses countVectorizer for feature extraction. The Naive Bayes approach [81] combines the bag-of-words (BoW) model with n-gram analysis and Part of Speech (POS) tagging. This comprehensive approach captures both word frequencies and sequential relationships, enhancing ambiguity detection. This insight has guided us in considering similar techniques to enhance the accuracy of ambiguity detection. Similar to these works, we have incorporated the use of Word embedding techniques as feature representation for various machine learning models.

Though a lot of work has been carried out in the field, none of the work has taken domain-specific expert opinion while deciding on the desirable properties of a software requirement. As our work focuses on requirements for automotive systems, experts' domain knowledge will be used to decide the desirable properties.

Furthermore, most of the work focuses on general software requirement, but our work focus on automotive software requirements. The requirement properties which might be important for automotive software requirements might not be enough for requirements in general. Similarly, there might be many requirement properties which a requirement in general should meet but these properties might not be important in automotive requirements.
In essence, our research enriches the field by combining domain-specific data with advanced deep learning models, offering a fresh perspective on ambiguity detection tailored explicitly to the automotive industry. This holistic approach enhances the landscape of text classification methodologies and underscores the relevance of our study within this specialized domain.

4
Method

The work has centered around collecting experts' opinions on how a requirement should be reviewed and creating models based on data collected from the semi-structured interviews with industrial experts working at various levels and extents with requirements in their work. The chapter starts with an overview, which has been presented in Section 4.1. The technique used for the collection of requirements data has been presented in Section 4.2. Section 4.3 focuses on the collection of views and opinions from domain industrial experts through interviews and processing the information collected from interviews. Various machine learning models have been used to classify the requirements and they have been described in Section 4.4. Section 4.5 discusses the various ways that have been used to validate the models developed and also to evaluate the best solution out of the different approaches that have been tried in the work.

4.1 Overview

Requirements are crafted using everyday language and often by different people, they're naturally prone to errors, inconsistencies, and vagueness. Developers and product owners have to manually verify them, which can be quite resource-intensive. Unfortunately, there are instances where these requirements get misinterpreted, leading to unwanted features in the final software. That's why finding a solution that maintains consistency and can automatically review these handwritten requirements is of utmost importance.

Aptiv, as an automotive supplier company receives thousands of requirements from its client companies. The requirement repositories have 1000s of requirements data written in natural language and comes from various authors working in different industries and domains, and have different levels of expertise. Some requirements have been labeled based on various properties like specific, measurable, achievable, not ambiguous, complete, etc. These labelings have been done during the manual requirement review process. In order to know which of these desirable properties is more important in the automotive industry, we have performed interviews with experts with different levels of expertise working with requirements. The interviews have been thematically coded and analyzed to extract valuable information that will be useful in creating solutions to review requirements. Based on these interviews we have some properties that the experts think are more important than others. These properties have been examined in requirements by the use of machine learning mod-

els. More description of each step has been explained in the later sections. Figure 4.1, presents an overview of the workflow performed for the work.

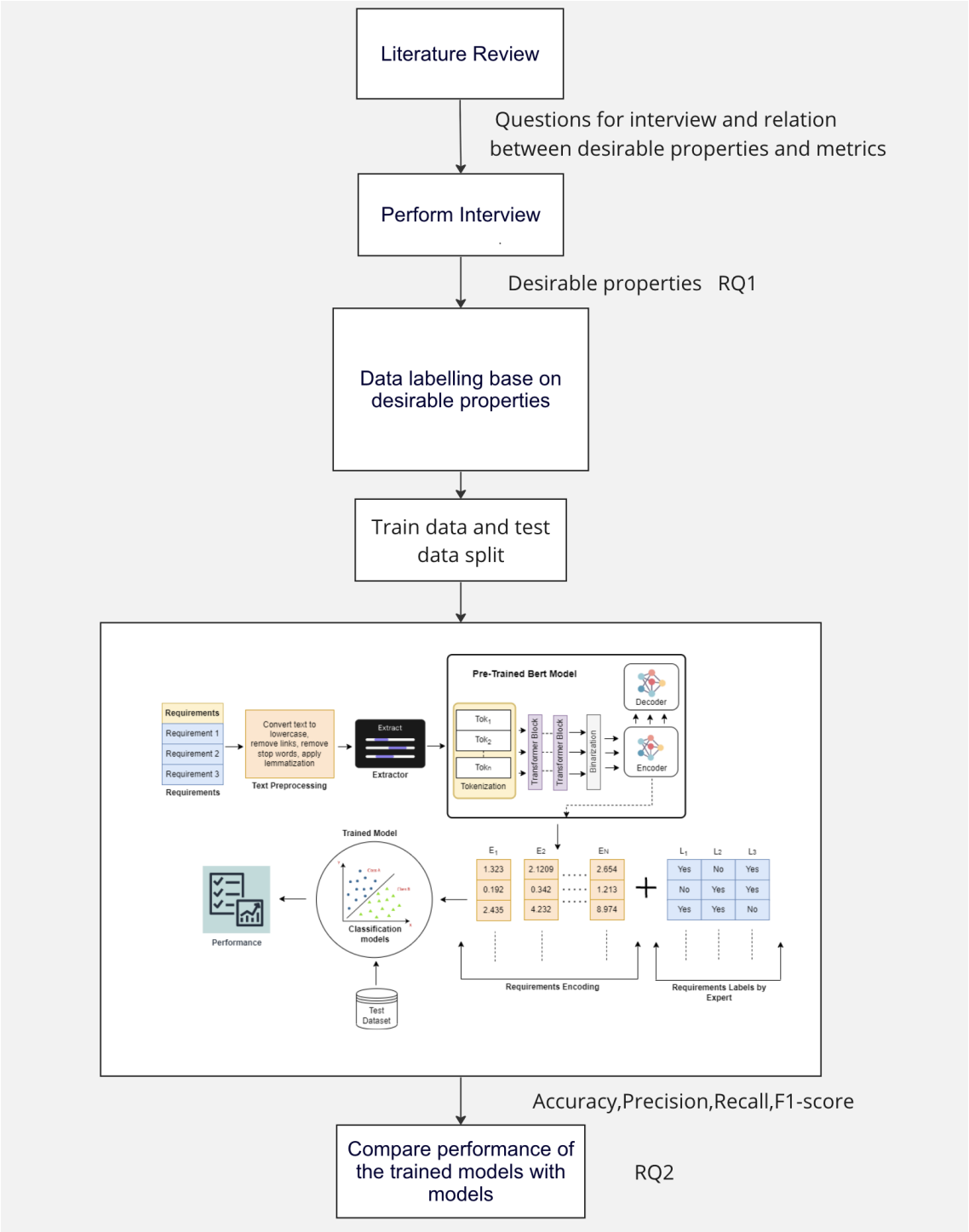


Figure 4.1: Study Workflow

In order to answer RQ1, we have performed interviews with automotive experts. Based on the interview answers we have selected the requirement properties that are

most desirable in the automotive domain. The performance of the different model combinations have been compared to answer RQ2. Based on the interview results, we have used 3 properties which the industrial experts think are most important in the automotive domain. We have used labeled requirements data to train and test models. We have used 5 combination of NLP techniques with ML models. NLP techniques have been used to convert textual data into numerical vectors which the machine learning models use as input. These vectors have been used to train ML models.

4.2 Data Collection

In this project, data has been collected in two forms: Interview data collection and requirement data collection. The interview data collection process involves gathering information from individuals through face-to-face or online interviews. It involves asking specific questions to the interviewees and recording their responses for later analysis. Interviews are often used in research studies to gather in-depth and detailed information from participants, particularly when exploring complex or sensitive topics. Interviews can also provide researchers with a better understanding of participants' perspectives, experiences, and attitudes.

The requirements data collection process involves the collection of requirement data that we have used to train and test the models we have designed in this project.

4.2.1 Interview

In our case, we wanted to be a bit flexible and were open to gathering more information than the pre-defined set of questions that we had. Runeson[66] suggests that semi-structured interviews may be particularly useful when the researcher wants to explore complex issues and gather detailed qualitative data but also wants to ensure that certain key topics are covered consistently across all participants. In this way, we selected to do semi-structured interviews.

The interview process was conducted in complete anonymity, ensuring that the participants' identities remained confidential. Every individual involved was provided with an interview consent form prior to the interviews, outlining the purpose and scope of the study, as well as their rights as participants. This approach was taken to uphold ethical standards and respect the privacy of those contributing to the research.

4.2.1.1 Interview Guide

Interview questions have been designed based on the research questions and the research topic we are exploring. The first draft of the question was framed and sent to our supervisor for review. After two iterations of reviewing questions for interviews, five questions were framed in the final draft. The interview design also involved designing the interview ethics. Interview ethics is crucial to protect participants'

rights, privacy, and well-being, and to uphold the integrity and credibility of the research process.

The whole interview workflow was divided into four parts. Firstly, we illustrated the purpose and use of the interview and ask permission to record the interviews of each expert. Then, some background questions are asked to grasp the domain experts' backgrounds. Apart from that, some questions related to requirements were asked to get in-depth information related to requirements. A requirement should meet certain properties to ensure the goodness of the requirement. Not all requirements that a company receives meet these properties. It is also always a question that what properties a requirement should meet. In order to make it more standardized, we have used characteristics of requirements specified in IEEE international standard as a list of properties a requirement should likely meet[5]. So, when we performed interviews, we presented this list to the experts attending the interview and we took their opinion on which of the characteristics mentioned in the list they think are more important than others. Finally, we showed our plan to use machine learning methods to judge those requirements and sought insights from experts to gather their opinions on the proposed approach for automating the requirements review process.

Table 4.1 shows the questions asked during the interview.

PART1	
1	What role do you play?
2	How many years have you had your role?
3	Area of expertise.
PART2	
1	How would you define a requirement?
2	To what extent do you use requirements in your day-to-day work?
3	What strategy you use currently to validate a requirement?
4	Have you ever faced problem in your work, because of badly written requirement. Can you give an example?
5	What qualitative properties do you think a good requirement should meet in the automotive sector? Necessary, appropriate, singular, unambiguous, complete, feasible, verifiable, correct, confirming (Characteristics of individual requirements) Complete, consistent, feasible, comprehensible, able to be verified (Characteristics of a set of requirements)
PART3	
1	Do you think using quantitative metrics related to properties will be an adequate and reliable way to review requirements?

Table 4.1: Detail interview questions

Data Collection using interviews included identifying interviewees, contacting interviewees, scheduling interviews, and performing interviews. Because of the limited time of six months and limited access, we have identified nine automotive experts with different backgrounds and experience levels. The details of the experts have been provided later in Section 4.2.1.2. Each interview took around 30-45 minutes. As the interviews were semi-structured, we were open for them to present their opinion and experiences apart from topics mentioned in the questions.

4.2.1.2 Interviewee Demographics

The interviewees we target are domain experts in the automotive sector. These interviewees work at APTIV in different roles. Meanwhile, we try to gain a deep and comprehensive understanding under investigation[64]. Therefore, we use purposive sampling [63] [64] as the sampling strategy, which can also help us to focus on the specific participants who mostly provide rich and valuable data. The experts we interviewed already have varied levels of experience in the automotive domain. Experts from various levels of experience were chosen to provide a more balanced and nuanced view of the topic. Experts with more experience will have a better understanding of the complexities related to the topic and can also provide a historical perspective [65]. On the other hand, experts who have less experience will be more updated with the emerging trends and technologies in the area [65]. Additionally, they may be more attuned to the challenges and opportunities facing newcomers to the field, which can be helpful for those who are just starting out [65]. The experts selected for the interview work in different roles such as Product owner, Software engineer, Software architect, System engineer, Product owner for testing, etc. In this way, more useful information can be grasped than interviewing experts with similar experience levels or experts belonging to similar domains and we can find valuable information from a different perspective.

Table 4.2 provides the details of the expert who participated in the interviews, the experts we have interviewed have different backgrounds, and levels of experience and hold different kinds of roles in the company. In order to maintain confidentiality, we have replaced the name of interview participants with P1, P2, P3, etc, as presented in the Table.4.2.

4.2.2 Requirements

At Aptiv, obtaining the labeled requirements for this project proved to be a challenging task. The first step was to conduct data collection as the required data was not readily available. The documentation of the requirements has been done using a system that is not universally accessible and requires multiple levels of authorization. Although software developers can access the software requirements, system requirements, and stakeholder requirements for a specific project, accessing requirements from other projects is not possible. As we are targeting the entire automotive industry, we have gathered data from various projects. For this work, we had access to three projects, so we have collected requirements from three projects. The projects from where the requirements have been collected cover different areas of

Interview Participants List			
Experts	**Role**	**Area of expertise**	**Experience**
P1	Software Engineer	Embedded Developer	5
P2	Software Architect	Functional Safety, Architecture Integration, Processes	11
P3	System Lead, Product Owner	Stakeholder Requirement, Creating System Requirements	15
P4	System Engineer	Review Requirements, Traceability, Test	5
P5	Product Owner for Test	Verification and Validation	16
P6	Software Developer	Autosar, Fault handling, Application Layer	4.5
P7	Advanced Engineering	R&D, Testing New Technology	3.5
P8	Software Developer	Autosar, SWE3	10
P9	Project Software Developer	Software Development, Autosar	10

Table 4.2: Experts Details

the Automotive industry and they follow a specific pre-determined structure. The areas of these projects are Vehicle Connectivity, Radar Perception and Camera Sensors. Projects from different areas have been selected, so that the machine learning models will grasp insights from diverse areas of the Automotive industry. Unfortunately, there was no automated method available for collecting requirements from the documents. Thus, we manually created the dataset by extracting requirements from the requirement specification documents and placing them into a sheet. During the collection process, we recorded the requirements as they were originally written, without any modifications. This is because errors, ambiguities, inconsistencies, and other issues in writing are crucial for our work.

Because of confidentiality associated with Industrial data, efforts have been made to acquire the necessary permissions to present some examples of requirements in this report. Here we present three examples that are part of the requirement dataset used.

	Requirement Examples
1	This request shall notify VIU that VCU is shutting down. VIU needs to be aware that VCU is shutting down for its diagnostic state machine.
2	To ensure vehicle functionality the CAN ports supporting wake-up shall be able to wake up the ECU and the ECU shall transmit a Wake up signal on LP1CAN.
3	To ensure vehicle functionality a wake-up request from the VCU shall cause the ECU to wake up.

Table 4.3: Requirement Examples

4.3 Data Analysis

4.3.1 Interview Data Analysis

Braun and Clarke[31] published a seminal paper that outlines a popular and widely used method for analyzing qualitative data in the field of psychology. This paper provides practical guidance for conducting a thematic analysis, such as selecting a suitable dataset, generating initial codes, developing themes, and interpreting and reporting results.

Thematic analysis is an independent qualitative descriptive approach. It is mainly described as a method for identifying, analyzing, and reporting patterns(themes) within data.[31]. It involves analyzing and categorizing data to uncover meaningful patterns and themes that may be relevant to a particular research question. As we have performed interviews, the importance of thematic analysis of interviews lies in its ability to provide rich and details insights into the perspectives, behavior in particular situations, and interviewees' experiences of the topic of interest. Performing thematic analysis, we believe that we can get a deeper understanding of the underlying meaning and contexts of the data collected from the interview and draw more informed conclusions.

As mentioned in Table 4.1, we had five interview questions that provided information about industrial experts' views and usage of requirements in their work. In order to analyze the data, we followed a combination of a top-down and bottom-up approach [61] to extract information. Braun and Clarke say both approaches can be useful in different contexts, but they emphasize the importance of starting with a flexible and open-minded approach that allows the themes to emerge from the data [61].

As we are two individuals working on this thesis work, we have performed the thematic analysis separately, with one of us doing five and the other doing four, and have validated each other's code twice. Then, we discussed again and decided on codes.

Here are the six steps of Braun and Clarke's that we have followed to perform the thematic analysis:

4.3.1.1 To know the data

To get more familiarized with the interview data, we read and re-read the interview transcripts multiple times to develop a comprehensive understanding of the content. This included identifying and noting key ideas, concepts, and patterns that emerged from the data. We have read the transcript several times to ensure we haven't missed out on any key concepts.

4.3.1.2 Creating codes

The next step involves generating initial codes or labels that describe the content of the data. Codes were determined based on the theoretical framework in place. We pre-determined what data we were planning to look at in the interview transcripts and based on that we developed codes. We also added some codes as they emerged from the transcript and were thought to be important. Codes have been assigned based on the importance of a particular word, theme, or a few words that we think are important for the study.
The codes that we developed from the interviews data have been discussed below:

- *Requirements Definition:* This code provides an overview of experts' understanding of requirements. Many people might have different views on what they understand a requirement as.

- *Extent Used:* This code provides an idea of how much the expert has been involved with requirements in his/her day-to-day work. This particular code is very important for our work as it has been used later on to provide weight to the experts' views and opinions. Experience level and extent of use provide an overview of how much the expert understands the importance of requirements. An expert working closely with requirements will have a better understanding to judge the goodness of requirements.

- *Badly written requirements:* This code focuses on getting information regarding how a badly written requirement might look from an expert's perspective. What can be the potential impact of a badly written requirement and what problem it might lead to?

- *Requirement review:* This code provides an idea of the currently used strategy or tools in the automotive industry to review requirements. Do experts in the industry use any tool or do they review these requirements manually? Who is responsible to review requirements?

- *Extent Used:* This code provides an idea of how much the expert has been involved with requirements in his/her day-to-day work. This particular code is very important for our work as it has been used later on to provide weight to the experts' views and opinions. Experience level and extent of use provide an

overview of how much the expert understands the importance of requirements. An expert working closely with requirements will have a better understanding to judge the goodness of requirements.

- *Requirements properties:* This code is vital for our work as it provides an idea of how to define the goodness of the requirement. Requirements should meet certain desirable properties but not all of them are equally important from an automotive industry point of view. This code provides us with the properties that the experts think are more important compared to others.

4.3.1.3 Finding themes

After generating initial codes, we reviewed codes to identify potential themes. Themes are generally a group of codes that share common ideas or characteristics.

4.3.1.4 Review of themes

This step involves reviewing the identified themes to ensure they accurately represent the data. We have checked whether the themes are relevant to the research questions, whether they capture the essence of the data, and whether they provide a coherent and meaningful interpretation of the data. We have also ensured that each code belongs to one and only one theme and that all the data has been analyzed.

4.3.1.5 Refinement of themes

Once we finished reviewing the themes, we defined and named each theme uniquely. We have ensured that the theme definition is concise, precise, and captured the essence of the theme. We have also tried to ensure that the name is simple and reflects the content of the theme.

4.3.1.6 Documenting the findings

Finally, we write up the findings of the thematic analysis, including a description of the themes, examples of the data that illustrate each theme, and any relevant quotes or observations. We have also provided a clear interpretation of the findings and their implications, including any limitations or challenges. We have reflected on the analysis process and led the research in the direction based on the finding from the analysis.

4.3.2 Requirement Data Labeling

As mentioned in Section 4.2.2, Requirements data has been collected for three projects from various areas within the Automotive industry. This project has been carried out in collaboration with APTIV and they currently have no tool to review requirements automatically. Requirement review is performed manually by experts working on projects. All the teams working on these projects perform manual reviews of the requirements based on qualitative properties. They have labeled

requirements with respect to various quality criteria during the manual review process. These labeled requirements are maintained on sheets along with each requirement's ID. The data we have used in this project has 1509 labeled requirements, with around 500 requirements coming from each area. These labels have been done by experts working on the projects. The quality criteria that the experts have used to label requirements are: applicable, not ambiguous, complete, consistent, testable, feasible, and implementable. For this work, we have created a sheet where we have collected the requirements from the three projects and based on the requirements ID, we have collected the labels created by the experts for each requirement. The number of unambiguous requirements in the dataset is 1324 and ambiguous requirements is 185. The number of consistent requirements are 1296 and inconsistent requirements are 213 and the number of verifiable requirements are 1283 and non-verifiable requirements are 226.

To ensure the accuracy and validity of the requirement labels, a validation process was undertaken. While performing the interviews, the experts were asked, if they were available and willing to label some requirements for our work. Five out of nine experts agreed to label requirements. Three of these five experts were contacted again to label 100 requirements each. These three experts were selected based on the projects they work on and their experience level. In this process, the existing labels were temporarily removed, and the requirements were sent to experts for re-labeling. The original labels were not provided to the experts. This process has been performed to verify the labels we have found from the manual review process.

To ensure consistency and reliability, a thorough comparison was conducted between the initial labels and the labels provided by the experts. This step aimed to identify any discrepancies or variations in the labelling. We have used Cohen's Kappa [84] to calculate the similarity between the labels, in the case of ambiguity the score is 0.7618, for consistency, the score is 0.6476 and for verifiability the score is 0.6685. The comparison showed substantial agreement between the original labels and the newly assigned labels. This high level of the agreement demonstrates the consistency and reliability of the labelling process. The matching labels provide confidence in the accuracy and quality of the requirements dataset.

4.4 Machine Learning Techniques Used

Machine learning techniques have been employed to effectively classify whether requirements meet qualitative properties or not. Models have been trained on the labelled requirements dataset to check if a requirement is ambiguous, consistent and verifiable. These three properties were selected based on the result of the interview which has been discussed later in Section 5.1. These techniques enable the development of classification models that learn from labelled data and can subsequently classify new, unlabeled requirements. In this work, we have used five classification techniques that are primarily used in machine learning tasks: Convolutional Neural network, Recurrent Neural network, Naive Bayes, Random Forest algorithm and Support Vector Machine. These techniques have been widely used for classification

tasks in various domains. We have employed five different techniques to examine the most effective techniques out of these five. All these techniques have their own strengths and weaknesses and their performance varies based on the characteristics of the data the model is trained on. We are also using word embeddings to convert requirements from textual representations to numerical representations. Word embeddings are essential for text classification because they address the limitations of traditional methods that treat words as discrete and independent entities. Word embeddings provide a way to represent words in a continuous and dense vector space, capturing semantic relationships and contextual information.

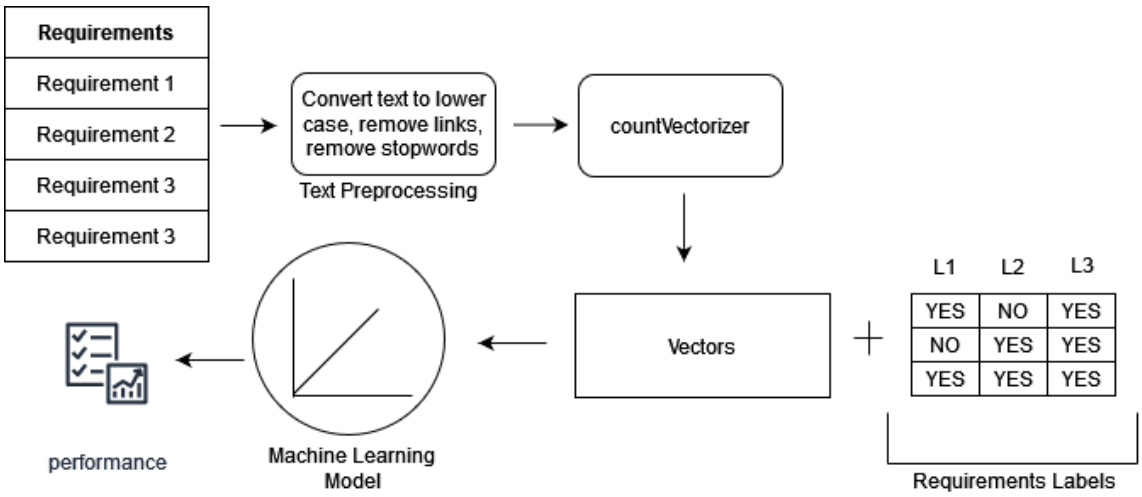


Figure 4.2: Pipeline of the proposed machine learning approach

The proposed methodology encompasses five distinct combinations, each tailored to handle specific aspects of the textual data. The first three combinations involve the application of the CountVectorizer technique to transform textual data, followed by classification using different algorithms - Naive Bayes, Random Forest, and Support Vector Machine. These combinations leverage traditional machine learning techniques known for their efficiency and interpretability in text classification tasks. Figure 4.4 shows a graphical overview of the machine learning pipeline used in the three combinations used.

To extend the exploration of cutting-edge NLP approaches, we also utilize deep learning techniques in the remaining two combinations. The fourth combination adopts the Bag-of-Words (BoW) representation, followed by processing with a Convolutional Neural Network. CNNs excel in extracting hierarchical features from data and have demonstrated significant success in NLP tasks.
In the fifth combination, we introduce a preprocessing step using the PadSequence technique, converting textual data into fixed-length sequences suitable for Recurrent Neural Networks. RNNs are well-suited to capture the temporal dependencies present in natural language, making them an ideal choice for text classification. Figure 4.4 shows the pipeline used to train two combinations that uses deep learning models.

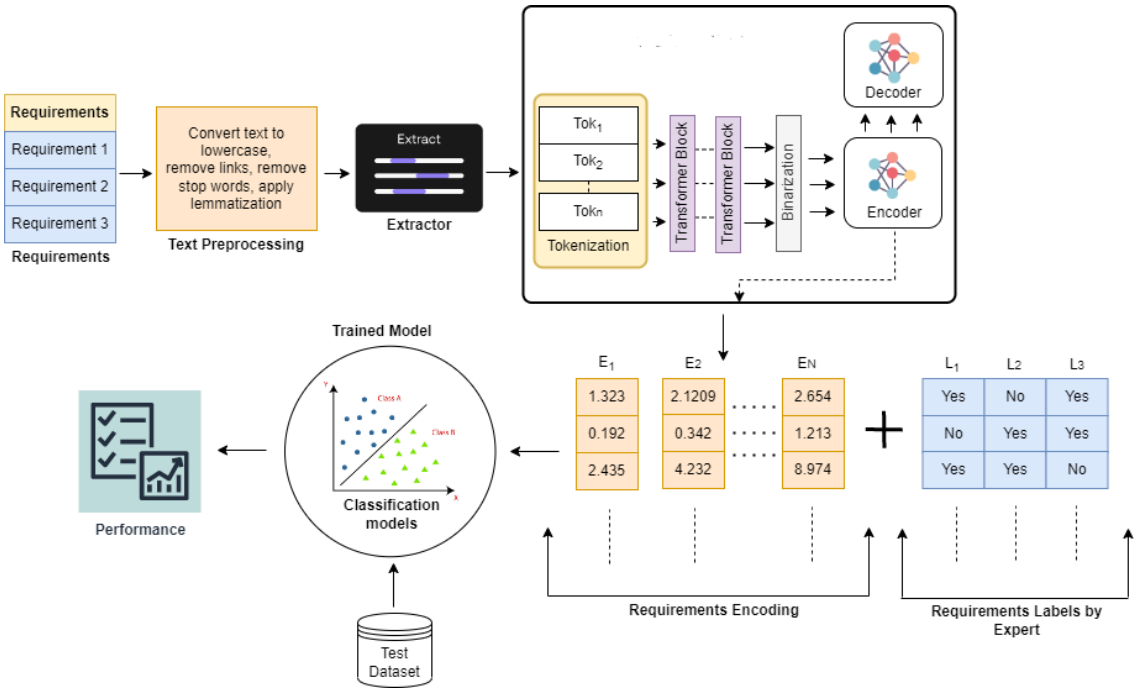


Figure 4.3: Pipeline of the proposed deep learning approach

4.4.1 Train-Test Split

We used train_test_split helper function in the scikit-learn library [73] to split the dataset into train set and test set. It can easily generate a random split of your dataset into training and test sets. Figure. 4.4.1 shows the implementation of train_test_split function in our work.

```
[9] from sklearn.model_selection import train_test_split
    X_train, X_test, Y_train, Y_test = train_test_split(data[X_cols], data['Labeling'],test_size=0.2, random_state=42)

    print(X_train.shape)
    print(X_test.shape)

    (1032,)
    (259,)
```

Figure 4.4: Train-test Split

In order to avoid over-fitting and to check how our models perform on new data, we have reserved 259 labelled requirements for testing our models and not using them to train. These 259 test data come from the train-test split that we have performed and this same 259 test set has been used for testing all the machine learning models we have trained.

When assessing various configurations or "hyperparameters" for estimators, there is a potential risk of overfitting on the test set. This occurs because the parameters

can be adjusted until the estimator achieves optimal performance. Consequently, there is a possibility that knowledge about the test set may inadvertently influence the model, resulting in evaluation metrics that no longer accurately reflect its generalization performance. This phenomenon is commonly referred to as "leakage" of test set information into the model. In order to avoid this, we have also used K-fold Cross Validation, with k value as five. It involves splitting the available dataset into k equally sized subsets or folds. The model is then trained and evaluated k times, each time using a different fold as the evaluation set and the remaining k-1 folds as the training set. It helps in mitigating over-fitting by training and evaluation on multiple subsets and assessing average performance. We used KFold() method in the scikit-learn library [73] to implement K-fold Cross Validation.

4.4.2 Data Preprocessing

In the initial stage of text preprocessing, the textual data, comprising various requirements written in natural language, undergoes several crucial steps. Firstly, the sentences are tokenized to segment them into individual tokens, allowing for a more granular analysis. Next, we perform stop-word removal, which involves eliminating common, non-informative words that typically do not add value to the classification process.

To ensure uniformity and facilitate word embedding, all text is converted to lowercase, minimizing discrepancies due to letter case variations. Additionally, any hyperlinks or URLs present in the requirements are removed, as they are not pertinent to the classification task and could introduce noise.

4.4.3 Word embedding

After the text preprocessing steps, the pre-processed textual data is transformed into a numerical representation using three word-embedding techniques: CountVectorizer, Bag-of-Words (BoW), and pad_sequences. In the traditional machine learning model, we have used countVectorizer for word embedding whereas in LSTM(RNN) we have used pad_sequences. The SpaCy model that uses CNN, does word embedding by itself, using Bag-of-Words.

4.4.3.1 CountVectorizer

This technique converts the textual data into a sparse matrix, representing the frequency of each word within the corpus. The resulting matrix provides a numerical representation of the textual content, which is suitable for training the classification models.

4.4.3.2 Bag-of-Words (BoW)

Similar to CountVectorizer, BoW also captures word frequency. However, BoW creates a dense vector representation of the textual data, representing the occurrence

of each word in the requirements. This approach provides a straightforward and efficient numerical representation for further processing.

4.4.3.3 Pad_sequences

For sequences of varying lengths, such as sentences of different word counts, padding is essential to ensure consistent input dimensions for the classification models. The pad_sequences technique is utilized to add padding to the numerical representations, facilitating uniformity in data processing.

4.4.4 Classification Models

The next phase involves training and testing various classification models for the requirement classification task. The models encompass Support Vector Machines (SVM), Random Forests, Logistic Regression, Neural Networks, and Gradient Boosting Machines, each chosen for its specific capabilities in handling different classification scenarios. These models are fine-tuned and optimized to maximize their performance in categorizing the numerical representations into appropriate requirement classes. In this work, we have used five different classification models - Naive Bayes, Random Forest, Support Vector Machine, Convolutional Neural Network and Recurrent Neural Networks.

4.4.5 Performance Evaluation

The effectiveness of the proposed methodology is evaluated through a performance assessment of the trained classification models. Metrics such as accuracy, precision, recall, and F1-score are employed to gauge the models' proficiency in handling the converted textual data. Comparative analyses are conducted to identify the most suitable combination of word embedding techniques and classification models for the requirement classification task.

4.5 Combination of countVectorizer with machine learning classifiers

In this task, the textual data comprises various requirements written in natural language, and CountVectorizer efficiently converts this text into a numerical representation based on word frequency. By constructing a sparse matrix with rows representing individual requirements and columns corresponding to unique words, CountVectorizer captures the importance of each word in the requirements based on its frequency of occurrence.

This advantage is particularly crucial for requirement classification, as frequently occurring words often carry essential semantic meaning and can serve as discriminative features for categorizing requirements into different classes. Moreover, the

sparse nature of the resulting matrix helps in handling large datasets with extensive vocabularies, reducing memory usage and computational complexity. Overall, CountVectorizer's ability to emphasize frequent words and its computational efficiency makes it a valuable and practical choice for requirement classification tasks due to various reasons:

- Domain-Specific Vocabulary: Requirements engineering has its own set of domain-specific terminology. Frequent words could encompass these specialized terms that hold particular significance within RE. By capturing these terms, the model becomes attuned to the specific language of the domain.
- Contextual Insights: Some words might appear frequently due to their contextual relevance within requirements. These contextually important words could be associated with certain types of requirements, such as security or performance. CountVectorizer's focus on frequent words helps the model decipher such nuances.
- Key Terminology: In many domains, certain words or phrases serve as pivotal indicators of specific concepts or topics. Emphasizing frequent words allows the model to recognize and give more weight to these crucial terms, aiding in accurate classification.

Figure 4.5 shows the implementation of countVectorizer to convert textual data to numerical form. The requirement data has been converted into a matrix of size 821*2567. Figure 4.5 also shows the feature names of the converted numerical data.

```
[27] from sklearn.feature_extraction.text import CountVectorizer
     vector = CountVectorizer(stop_words ='english')
     X_train=X_train.fillna(' ')
     vector.fit(X_train)

     X_train_transformed =vector.transform(X_train)

     X_train_transformed

     <821x2567 sparse matrix of type '<class 'numpy.int64'>'
             with 11750 stored elements in Compressed Sparse Row format>

     vector.get_feature_names_out()

     array(['000', '05', '085', ..., 'ô1õ', 'ôcutõ', 'ôred_sens'], dtype=object)
```

Figure 4.5: countVectorizer Implementation

There are various reasons we considered using the countVectorizer with simple machine-learning models [85]:

- Simplicity: CountVectorizer is straightforward to implement and does not require complex pre-processing steps or hyperparameter tuning. It directly converts text into a numerical representation based on word frequency, making it a convenient option, especially for quick prototyping and simple text classification tasks.

- Efficiency: CountVectorizer is computationally efficient and memory-friendly, as it generates a sparse matrix representation. This efficiency is crucial when using simple machine learning classifiers that might not handle high-dimensional feature spaces well.
- Robustness to Data Size: CountVectorizer performs well with limited amounts of training data. This can be beneficial when working with small datasets like our requirement dataset where other more advanced word embedding techniques, such as deep learning-based methods, might not have sufficient data for effective training.
- Performance with Simple Classifiers: For certain straightforward classification tasks with linearly separable or relatively simple decision boundaries, countVectorizer, when combined with simple classifiers like Naive Bayes or Logistic Regression, can perform competitively. Complex word embedding techniques might introduce unnecessary complexity and not necessarily lead to better performance in such cases.
- Interpretable Features: The word frequency-based approach of CountVectorizer results in features that are interpretable and easily understandable as presented in the figure 4.5. This interpretability can be valuable for gaining insights into which words are most influential in the classification process and for building trust in the model's predictions.

Though there are various reasons to use countVectorizer with simple machine learning models, it is not an effecient embedding technique to use with deep learning models because countVectorizer generates fixed-length vectors representing the frequency of words in a document. This format is well-suited for traditional machine learning algorithms that often expect fixed-size input. In contrast, deep learning models, like Recurrent neural networks (RNNs) or Convolutional neural network, can handle variable-length sequences directly through techniques like padding and masking. countVectorizer provides simple term frequency information, which might not capture the nuanced semantic relationships between words. Deep learning models like CNN or RNN, especially those using pre-trained word embeddings, or contextual embeddings, can better represent the semantic meaning of words in varying contexts.

4.5.1 Naive Bayes Classifier

Naive Bayes is a family of supervised learning algorithms that utilize Bayes' Theorem [70]. Naive bayes is simple and easy to implement and generally effective when the dataset is small. Here are a few reasons why we used Naive Bayes classifier [86]:

- Simplicity and efficiency: Naive Bayes is a lightweight technique that is computationally efficient. It requires less computational resources as compared to other complex algorithms.
- Effective with limited training data: Naive Bayes perform well even with limited training data. It can make reasonably accurate predictions even with small training datasets.

- Feature independence: Naive Bayes considers each feature to be independent of the other. In our case, words in requirements are features, while other algorithms try to find similarities between words, Naive Bayes looks at each word(features) independently.

The scikit-learn module naive_bayes [70] was used to implement the Naive Bayes classifier. The Naive Bayes has many variants, Complement Naive Bayes has been used for this work, as Complement Naive Bayes is effective when there is an imbalance in training data.

4.5.2 Random Forest Classifier

A random forest is a machine-learning algorithm that combines multiple decision tree classifiers. It creates an ensemble by fitting decision trees on different subsets of the dataset and employs averaging to enhance predictive accuracy and mitigate over-fitting [71]. Random Forest was on of the techniques used due to various reasons [87]:

- Ensemble learning: Random Forest is an ensemble learning method that combines multiple decision trees. It reduces over-fitting by combining the prediction of multiple decision trees. As requirements data used in the work belong to a limited area, an attempt has been made to reduce over-fitting.
- Non-linear relationship: Random forest can capture non-linear relationships between features and target. AS requirements are text data, the relationship between text and classes can be complex and non-linear.

The random forest algorithm was implemented using the method Random Forest Classifier from the open source machine learning library scikit-learn [71]. Initially, the implementation relied mostly on the default values of the model's hyperparameters. However, as the accuracy demonstrated promising results compared to other classification methods, a hyperparameter optimizer was introduced to enhance the performance even further. The optimizer focused on fine-tuning five specific hyperparameters of the classifier.

- n_estimators: number of trees in the random forest,
- max_features: the maximum size of each random subset of features to be considered when splitting a node,
- max_depth: maximum depth of each tree,
- min_samples_split: the minimum number of data points in a node before it could be split,
- min_samples_leaf: the minimum number of data points that had to be passed forward to a leaf node for the split to be executed.

To optimize the classifier's performance, a hyperparameter optimizer named RandomizedSearchCV [72] was employed. This optimizer was provided with a range of values for each of the five hyperparameters. By randomly sampling and evaluating different combinations of hyperparameter values, the optimizer identified the optimal settings that yielded the highest performance for the classifier. Based on the optimal setting the optimizer yielded, the hyperparameter values we have used are n_estimators=99, max_features=2, max_depth=2, min_samples=2 and

min_samples_leaf=1.

4.6 Convolutional Neural Network with Bag of Words

Convolution Neural Networks(CNNs) are multi-layered artificial neural networks with the ability to detect complex features in data, for instance, extracting features in image and text data [68]. The Convolutional Neural Network is one of the techniques which was used in the work due to various reasons:

- Automatic Feature Extraction: In our task, we have text data but we have not defined any feature of the data. The only input to the model is the requirement text itself. CNNs have the ability to automatically learn and extract relevant features from raw data. Explicit feature definition is not required, thus alleviating the need for defining features explicitly.
- Transfer learning: CNNs can leverage pre-trained models for transfer learning in text classification tasks. As we are not defining certain features explicitly, a pre-trained CNN on some English data corpus will be helpful to enable to model to capture general linguistic patterns, semantics, and contextual understanding. As the model is pre-trained it can be fine-tuned to work even when we provide a smaller labelled dataset. The pre-trained model used for this work has been later discussed in 4.6.0.2.
- Robust to noise and variations: CNNs are robust to noise and variations, which is an advantage to our task. The requirements are written by various stakeholders and using natural language, which makes the dataset prone to variations in the style of writing.

We implemented CNN using the open-source machine learning library SpaCy[69].

4.6.0.1 Bag-of-Words

We have used the Bag-of-Words representation of the requirements data, and have used this representation for training the CNN model.

The utilization of Bag-of-Words (BoW) representation in conjunction with Convolutional Neural Networks (CNNs) for textual data has emerged as a promising approach to tackle various challenges. One key benefit of BoW is its ability to transform variable-length text data into fixed-length input vectors, facilitating the application of CNNs which typically require inputs of fixed dimensions. The convolutional layers in CNNs excel at detecting local patterns and capturing word associations, thereby capturing the semantic meaning embedded within the text. Additionally, BoW helps in reducing the dimensionality of high-dimensional text data, enhancing the computational efficiency of the CNN model while mitigating the risk of overfitting.

A bag-of-words is a way of extracting features from textual data for use in mod-

elling. A bag-of-words representation is a method used to analyze and represent a text by considering the frequency or presence of individual words within a document.

For example, take the sentences s1 = 'I am a boy and I laugh' and s2 = 'I am playing football' as our corpus. The vocabulary than is I, am, a , boy, and, laugh, playing, football. Now we count, how many times the words occurred in a sentence and create its vocabulary.
So, S1 bag-of-words representation becomes [2, 1, 1, 1, 1, 1, 0, 0] and S2 bag-of-words representation becomes [1, 1, 0, 0, 0, 0, 1, 1].

4.6.0.2 Loading a pre-trained model

SpaCy library has pre-trained models that can be used to further train, fine-tuned and adopted for classification tasks. SpaCy library has many pre-trained model options to choose from. As the requirements are written in English, we used a pre-trained English language model that use CNN. Using this pre-trained English model enables our model to understand the relation between words in a requirement, as the requirements have been written in English.

```
[ ]  import spacy
     #CNN ENGLISH
     nlp = spacy.load("en_core_web_sm")
```

Figure 4.6: Loading English CNN Model

The "sm" in the model name stands for "small," indicating that it is a smaller-sized model that includes word vectors, syntax, and named entities. We have used a smaller-sized model because we want our model to learn the relationship between words, training a larger model would have needed more resources and time. This model is trained on a large corpus of English text and is capable of performing various NLP tasks [78].

4.6.0.3 Model Training with labelled requirements dataset

The pre-trained model was trained using 1056 labeled requirements and validated through cross-validation. The initial implementation was based on the models' default values. Some hyper-parameters have been changed to improve the performance later. To optimize the classifier's performance, a hyperparameter optimizer named RandomizedSearchCV [72] was employed. Based on the value we derived from the optimizer, the hyper-parameter values we have used are epoch = 10 and batch size = 8.

- epoch : how many times the model will iterate over the entire dataset during training
- batch size : number of training examples used in each iteration

4.7 LSTM with Keras Embedding layer and pad_sequences

Long Short-Term Memory (LSTM) networks can be particularly advantageous for requirement classification tasks due to their capability to effectively capture and model sequential dependencies within text data. There are various reasons why LSTM can be a useful solution:

- Sequential Information: Requirements are often written in a structured manner, where the order of words and phrases matters. LSTM networks excel at learning patterns and relationships in sequences, making them well-suited for capturing the inherent sequential information in requirements.
- Feature Extraction: LSTMs can automatically learn relevant features from the data. They can capture both short-term and long-term dependencies, enabling them to identify complex patterns and nuances present in requirements.
- Stateful Memory: LSTMs have a memory cell that can remember information over long sequences. This can be beneficial for understanding context and making classification decisions based on accumulated knowledge.

The LSTM model[74] excels in capturing intricate sequential patterns within text data, making it ideal for understanding the contextual relationships between words in requirements. The pad_sequences function complements the LSTM by ensuring consistent sequence lengths, enabling efficient batch processing and fostering a seamless flow of information through the network. The Keras Embedding layer facilitates the transformation of words into dense vectors, encapsulating semantic meaning.

LSTM networks have performed quite well in many text classification tasks.[75] Their unique architecture makes them well-suited for capturing sequential dependencies and understanding the context of textual data. Requirements can involve long-range dependencies, where information from earlier parts of a requirement can impact the interpretation of later parts. LSTM's ability to mitigate the vanishing gradient problem allows it to capture these long-term dependencies, resulting in more accurate understanding of complex requirement structures [74].

4.7.1 Wordembedding with Keras' Embedding layer and pad_sequence method

The Keras Embedding layer converts words into dense vectors, effectively capturing their semantic meanings. By using this technique, we transform textual requirement data into numerical representations that carry contextual information. Additionally, we employ the pad_sequences function to ensure uniform sequence lengths, a crucial requirement for training neural networks. This approach not only facilitates the handling of variable-length text sequences but also prepares our data for effective utilization by subsequent layers, such as LSTMs. Figure 4.7.1 shows the use of pad_sequence in the requirement dataset before using it to train a LSTM Model.

```
[8]  import tensorflow
     from tensorflow.keras.preprocessing.text import Tokenizer
     from tensorflow.keras.preprocessing.sequence import pad_sequences

     max_len = 274  # Maximum length of the input sequence
     vocab_size = 36986  # Vocabulary size

     tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
     tokenizer.fit_on_texts(data['Requirement'])
     sequences = tokenizer.texts_to_sequences(data['Requirement'])
     padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='post', truncating='post')
```

```
[10] padded_sequences

     array([[ 329,   49,   16, ...,    0,    0,    0],
            [   2,  693,   38, ...,    0,    0,    0],
            [  34,  107,   38, ...,    0,    0,    0],
            ...,
            [  79,    2,  155, ...,    0,    0,    0],
            [   8, 3269,   87, ...,    0,    0,    0],
            [   3,  263,    2, ...,    0,    0,    0]], dtype=int32)
```

Figure 4.7: Pad_sequence implementation

4.7.2 Training a LSTM model

A LSTM was trained with labelled requirements. The initial implementation was based on the models' default values. Some hyper-parameters have been changed to improve the performance later. To optimize the classifier's performance, a hyperparameter optimizer named RandomizedSearchCV [72] was employed. Based on the value we derived from the optimizer, the hyper-parameter values we have used are embedding_dim = 128, hidden_dim = 265, output_dim = 10, n_layers = 2, and the dropout =0.5.

- embedding_dim : represents the dimensionality of the word embeddings that the LSTM model learns
- hidden_dim : hidden units, determine the capacity of the LSTM to learn complex patterns
- output_dim : number of classes or categories into which the model will classify the input sequences
- n_layers : number of layers refers to the depth of the LSTM architecture
- dropout : regularization technique used to prevent overfitting in neural networks

5
Results

This chapter describes the results and interpretations from the interviews and five different embedding-classification models combination built in this thesis.

5.1 Interview Results

5.1.1 Themes

We have identified several themes from the open-ended questions in the interviews:

- Knowledge about requirements
- Experience due to badly written requirement
- Requirements properties
- Current requirement reviewing in place

5.1.1.1 Knowledge about requirements

Based on the input provided by the participants, a requirement can be defined as a specific need or expectation that must be met by a product, service, or system. It should be clearly defined, measurable, and specific. The requirement should explain the customer's expectations, and it should be understood in the same way by all stakeholders involved in the project.

The interview data showcases a diversity of perspectives regarding the definition of requirements. One participant emphasized that a requirement should "explain customer expectations in my terms" (P1), highlighting the need for clear communication that bridges the gap between customer desires and project outcomes. Another participant stressed the importance of clear definition, stating that a requirement should "define clearly what should be achieved by the product" (P2). This emphasizes the role of requirements in aligning all stakeholders towards a common goal.

A participant with a software-oriented background shared that for them, system requirements are the linchpin. They mentioned that "system requirement should clearly say what is the output it requires and what all inputs do I have" (P7). This viewpoint underscores the pivotal role of comprehensive system-level specifications in shaping subsequent software requirements.

The iterative nature of requirements was a theme highlighted by another participant who explained how they evolve: "requirement is nothing but it can be input to a

software development playbook exactly you need to develop" (P6). This statement captures the dynamic process of transforming high-level stakeholder needs into actionable software directives.

One participant's focus on clarity and precision was evident in their definition: a requirement is something that needs to be "measurable and specific" (P9). This viewpoint underscores the significance of quantifiable criteria for evaluating project success. Additionally, the importance of mutual agreement and quality was captured by a participant who mentioned that a requirement "kind of defines the quality of the performance" of a software or physical entity (P9).

In summary, the interview participants collectively paint a comprehensive picture of requirements as the essential blueprint guiding project activities. These insights highlight the importance of understanding customer expectations, clarifying goals, and ensuring alignment across stakeholders. Whether seen as a bridge between customers and projects, a driver of clarity, or a metric of quality, requirements emerge as a cornerstone in successful project execution.

5.1.1.2 Experience due to badly written requirements

Poorly written requirements can lead to confusion, errors, and delays in project execution, as highlighted by the participants' examples. One participant stated, "Sometimes it creates confusion...the person who wrote the requirement might have a different expectation than the one who actually is trying to implement it" (P1). Another participant emphasized the lack of sufficient time dedicated to writing requirements, stating, "It's much too less time spent on writing requirements compared to other parts of development" (P2). The consequences of ambiguous requirements were exemplified by a participant who mentioned, "We have few requirements where it has been interpreted in different ways" (P3). Additionally, language barriers and mistranslations were identified as problematic, with one participant sharing, "The specification was written in French...defect and default is the same word in French" (P5). These examples highlight the importance of clear and detailed requirements to avoid misunderstandings, conflicts, and errors in project implementation. Participant 4 shared an example of language-related confusion, stating, "We had two requirements that said the same thing in the English sense but not in German sense. It was requesting different things for the same property but in German, it meant different things for different properties." Participant 7 highlighted the impact of mismatches in requested levels of verification, mentioning, "In the code, we could only set up one level of requirements, but they asked for two levels." Participant 8 emphasized the difficulties in testing activities caused by unclear requirements, stating, "If the requirements are not clear, definitely as a testing team, we cannot go ahead with our test case development and testing."

In all of these cases, the poorly written requirements led to confusion, errors, and delays in the project. It's important to write clear and detailed requirements to avoid these types of issues. From the responses provided by the team, it is clear that badly

written requirements can create confusion and lead to conflicts between different teams. It can also result in misinterpretation of the requirements, which can lead to errors in the final product. For example, poorly translated requirements can result in incorrect actions being taken by the development team. Additionally, poorly defined or unclear requirements can make it difficult for testing teams to develop test cases and properly test the product. In some cases, the lack of clear requirements can also make it challenging to determine when the project is actually finished. Overall, it is very important to ensure that the requirements on which the team is working are properly written meeting certain characteristics. It also emphasizes the importance of having a good requirement validation or review process.

5.1.1.3 Current requirement reviewing in place

Based on the responses provided by the participants in the interview, it appears that the process of requirement review is largely manual, and involves analyzing the requirements, understanding them, and ensuring that they are feasible and testable. Different strategies are employed by different participants, such as holding meetings with different teams to validate requirements, checking for clarity and measurability, and testing in real-world scenarios.

One participant highlighted the importance of understanding and interpreting requirements, stating, "Yeah, I read through it, try to understand it and interpret it. So it's manual, yeah" (P1). This initial comprehension is deemed crucial for the subsequent validation process.

The consensus among participants is that validating requirements involves a combination of factors. One participant outlined a structured process involving collaboration among the software team, stakeholders, and customers to assess testability, interpretability, and implementation feasibility. This emphasizes the interactive nature of the validation process, ensuring alignment across different perspectives: "Strategy, we follow the process...to validate the requirements...acceptable or not" (P3).

Validation also includes subjective assessments to determine if the requirements adhere to established criteria. Participant 4 mentioned considering aspects like specificity, measurability, and adherence to SMART criteria: "A lot of like subjective feeling...smart, specific, testable and measurable...is this clearly describing what they are asking us to do".

The theme of comprehensibility is underscored by Participant 5, who highlighted the need for contextual understanding: "You need the context of several requirements to understand...try to validate it functionally, is it understandable what they want to do and does it make sense". This suggests that a holistic perspective is required to ensure requirements make sense within the larger framework.

Overall, the interview data show that validation of the quality of requirements is an essential part of the software development process. The prevailing strategy for

requirement validation involves a meticulous manual process that combines understanding, collaboration, and subjective assessment. This approach emphasizes the importance of clear, quantifiable, and contextually coherent requirements to drive successful project outcomes.

5.1.1.4 Requirements properties

IEEE [5] list requirement properties for a single requirement and requirement properties for a set of requirement. During the interview, we asked the experts to list some of the relatively more important requirement properties.

We have prepared a list to show what these experts think about the characteristic and which characteristics they think are important for a requirement to meet in the automotive industry. Here is the summarized list of the answers from each participant:

- P1: "ambiguous, feasible, and verifiable"
- P2: "feasibility, consistency"
- P3: "unambiguous, verifiable, correctness, and possibly feasible"
- P4: "unambiguous, verifiable, consistent, and necessary"
- P5: "necessary, appropriate, consistent, and sometimes verifiable"
- P6: "unambiguous, complete, feasible, and possibly verifiable"
- P7: "verifiable, correct, testable, and specifying the type of verification"
- P8: "unambiguous, complete, verifiable"
- P9: "comprehensible, unambiguous, and verifiable"

Figure 5.1 shows what properties are more important to experts working in the automotive sector. The interview study shows that most of these experts think a requirement should be verifiable and unambiguous. Out of nine experts, seven mentioned that a requirement should be verifiable whereas six mentioned the importance of the requirement being unambiguous. There were a few more properties that the experts also mentioned, out of these nine experts three thought the requirements should be feasible, and similarly, three thought it should be consistent. A few of them also mentioned the importance of requirements being comprehensible, appropriate, correct, and necessary, while non of them mentioned confirming and singular as important.

5.1.2 Frequency of use

After doing a thematic analysis for the qualitative script, the frequency is calculated as follows.
Based on the interview data collected, we realized that different experts have different levels of expertise and the frequency of dealing with requirements varies from expert to expert. Some of the experts primarily work with requirements whereas some only use requirements when they are required to or asked to. We felt that people who work with requirements more often should be given more priority on

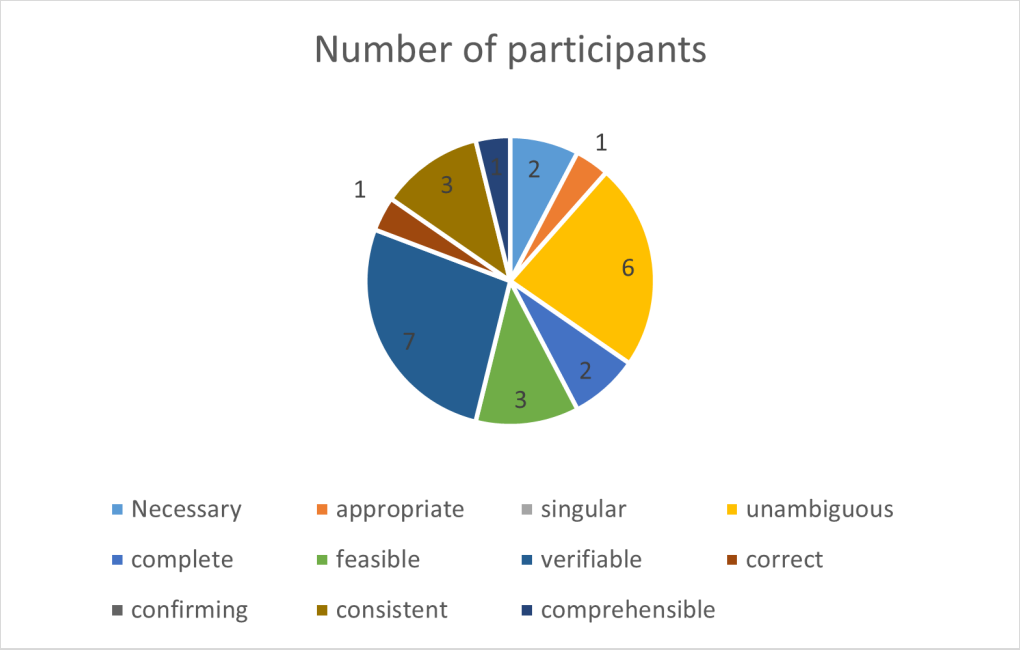


Figure 5.1: Requirement Property frequency

their opinions rather than the ones who use them comparatively lesser.

We have categorized the frequency of using requirements into three categories: 'high', 'low', and 'medium'. The experts that belong to the 'low' category said that they use requirements only when they are asked to, and the experts that belong to the 'medium' category said that they use requirements about 2 to 3 times a week. Experts who have been placed in the 'high' category work with requirements every day in their work. Figure 5.2 shows the number of participants put into each category based on their answers during the interview. Most(4) of the experts we interviewed said that they deal with requirements every day, 3 experts said that they use requirements moderately while 2 of them said they deal with requirements quite lesser.

5.1.3 Prioratising experts' opinion

Considering the situation that experts who are often using requirements have a better understanding of which requirement properties are more important to the domain in focus, the frequency of using requirements for each expert can be worked as a variable weight indicator.

As we saw in the above Section 5.1.1.4, each expert thinks differently and has their own priority of characteristics while judging the goodness of requirements. We felt that just counting how many experts think each characteristic is important doesn't provide us with the actual picture of which properties a requirement should meet in the automotive industry. As we saw in section 5.1.2, the frequency of each expert working with requirements varies from very less to completely working with

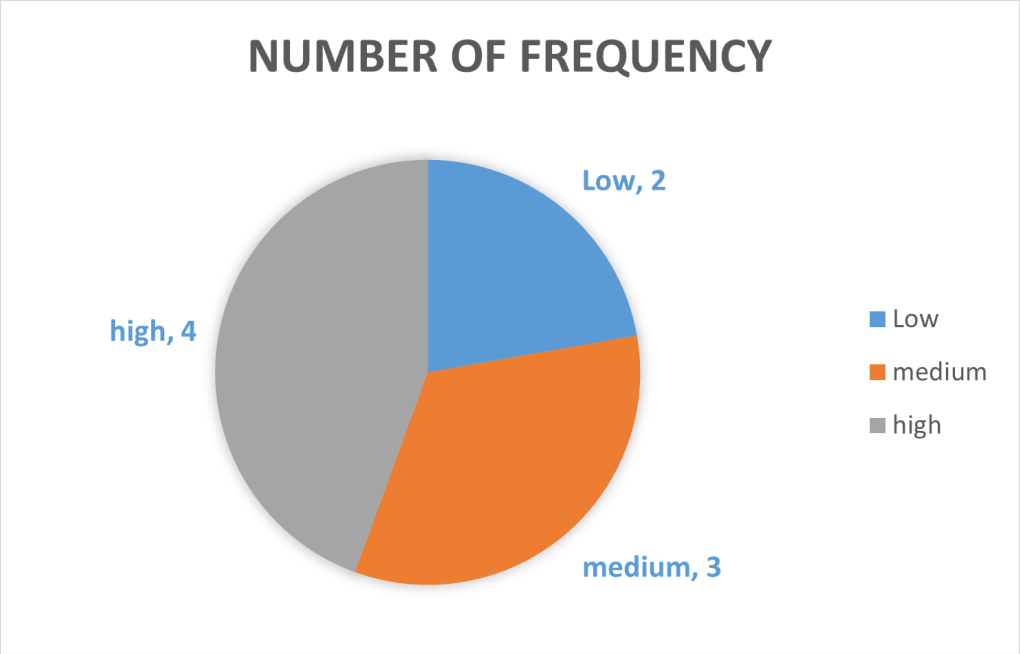


Figure 5.2: Using requirement frequency

requirements every day, we felt the opinion of each expert should be prioritized differently. We have tried to give importance to the opinion of experts based on their frequency of dealing with requirements in their day-to-day work. We have obtained a weightage system to prioritize the experts who are working more with requirements. Experts who are working with requirements every day and who have been put into the 'high' category in section 5.1.2 have been given a weightage of 3, and experts who have been put into the category of 'medium' have been put provided with a weightage of 2, similarly, experts in 'low' category has been provided with a weightage of 1. These weightage have been used in the next section to calculate the score of each properties.

5.1.4 Score calculation of each property

The score of each property as shown in Table 5.1 has been calculated using a weighted sum approach. This approach takes the category in Section 5.1.2 and the properties mentioned by the expert to calculate the score. Here we explain our approach using an example. Let's say we want to calculate the score of "unambiguous", We first check how many experts belonging to the category "high" mention "unambiguous" to be important and we multiply that number with 3(weightage), similarly we check for "medium" category and "low" category as well.

Total score = 3*(participant in 'high' category saying the property is important) + 2*(participant in 'medium' category saying the property is important) + 1*(participant in 'low' category saying the property is important)

The calculated score has been presented in Table.5.1. It can be seen that like the

Participant	P1	P2	P3	P4	P5	P6	P7	P8	P8	Score
Frequency	**L**	**M**	**H**	**H**	**M**	**H**	**M**	**H**	**L**	
Weight	**1**	**2**	**3**	**3**	**2**	**3**	**2**	**3**	**1**	
Necessary					yes		yes			4
Appropriate					yes					2
Singular										
Unambiguous	yes		yes	yes		yes		yes	yes	14
Complete						yes		yes		6
Feasible	yes	yes				yes				3
Verifiable	yes		yes	yes	yes		yes	yes	yes	15
Correct			yes							3
Confirming										0
Consistent		yes		yes	yes					7
Comprehensible									yes	1

Table 5.1: Requirement score calculation

frequency calculation in section 5.1.2, Both 'unambiguous' and 'verifiable' attain a very high score compared to the other requirements properties. 'consistent' and 'complete' also have a satisfactory high score of 7 and 6 respectively. All the other properties have attained a relatively low score.

Based on the score, we have selected "unambiguous", "consistency", and "verifiability" as the three properties which are most desirable in the automotive sector.

5.2 Classifying requirements based on unambiguity

This section discusses the result obtained using the classification methods discussed in the section 4. This iteration aimed to evaluate which set of word-embedding technique and classification algorithm provided the best performance when constructing a classification model for classifying the requirement based on whether they are ambiguous or not. Table 5.2 shows the performance scores of different combinations used in the thesis work.

While our dataset does contain both unambiguous and ambiguous requirements, it's important to note that the prevalence of unambiguous requirements is higher. This data distribution has a significant impact on our evaluation process.

Though we may observe high accuracy of more than 80% across all models in our analysis, it's crucial to recognize that accuracy alone might not provide the most insightful evaluation of our models' true performance. This is particularly important when dealing with datasets that exhibit class imbalances, where certain classes or outcomes are significantly more prevalent than others. In such scenarios, a high ac-

Text processing techniques	ML Models	Accuracy	Precision	Recall	F1
countVectorizer	Naive Bayes	0.8678	0.4583	0.224	0.300
countVectorizer	Random Forest	0.8782	0.4285	0.146	0.217
countVectorizer	SVM	0.7604	0	0	-
SpaCy(Bag of Words)	SpaCy(CNN)	0.9280	0.83	0.78	0.804
Pad_sequences	LSTM	0.8338	0.70	0.83	0.76

Table 5.2: Performance scores on classifying requirements based on Unambiguity

curacy metric can be deceptive, giving a false impression of the model's effectiveness. The model might appear to perform well simply by predicting the majority class frequently, even if it fails to identify instances from the minority class accurately. This can lead to a situation where the model appears accurate but is not truly effective for the intended task, especially when we're dealing with the underrepresented class.

Given this context, we're placing a greater emphasis on metrics that provide a more insightful understanding of our models' capabilities. Therefore, while acknowledging the presence of high accuracy in our models, we are placing greater emphasis on more informative metrics like Precision, Recall, and the F1 score. These metrics provide a more nuanced understanding of our models' performance, particularly when dealing with imbalanced data.

The traditional approach using CountVectorizer in conjunction with various classifiers yielded varying levels of success. The combination with Naive Bayes achieved a decent accuracy of 0.8678, yet its Precision (0.4583) and Recall (0.224) were relatively modest, leading to an F1 score of 0.300. The collaboration with Random Forest showed a slightly higher accuracy of 0.8782 but suffered from lower Precision (0.4285) and Recall (0.146), resulting in a low F1 score of 0.217. The combination with SVM struggled, achieving an accuracy of 0.7604 but has a Precision and Recall score of 0. One of the potential reason of getting Precision and Recall score of 0 can be attributed to the imbalance in dataset. Both Precision and Recall score expect True-positive(ambiguous correctly detected) to be a non-zero value. But in this case, SVM predicts all the test case data as unambiguous as our model has been trained with a training set which has mostly unambiguous requirements, resulting in a True-positive value of 0. The traditional approach using countVectorizer in conjunction with classical machine learning models like has proven inadequate in delivering satisfactory results when compared to the performance of deep learning models. The imbalance in the data exacerbates the challenges faced by these traditional methods, making them struggle to handle the nuances within the dataset.

On the other hand, the utilization of advanced techniques proved more promising. The SpaCy-based ensemble approach, combining Bag of Words and CNN, demonstrated remarkable performance with an accuracy of 0.9280, Precision of 0.83, Recall of 0.78, and an impressive F1 score of 0.804. This ensemble proved particularly adept at capturing the complexity of the data, showcasing its superiority in classifying requirements based on unambiguity.

Text processing techniques	ML Models	Accuracy	Precision	Recall	F1
countVectorizer	Naive Bayes	0.917	0.53	0.33	0.406
countVectorizer	Random Forest	0.901	0.66	0.142	0.233
countVectorizer	SVM	0.8304	0	0	-
SpaCy(Bag of Words)	SpaCy(CNN)	0.9280	0.814	0.793	0.803
Pad_sequences	LSTM	0.8765	0.765	0.723	0.743

Table 5.3: Performance scores on classifying requirements based on Consistancy

The application of deep learning techniques also showcased notable results. The use of Keras Embedding with LSTM achieved a decent accuracy of 0.8338, with balanced Precision (0.70) and Recall (0.83), leading to an F1 score of 0.76. While strong, this approach fell slightly short of the SpaCy-based ensemble, indicating the effectiveness of the latter in tackling data imbalance and optimizing Precision and Recall.

In conclusion, the SpaCy-based ensemble, specifically the combination of Bag of Words and CNN, stood out as the most effective approach, highlighting the importance of leveraging advanced techniques when dealing with imbalanced datasets.

5.3 Classifying requirements based on consistancy

The results presented highlight the performance of various models and feature combinations with a specific focus on Precision, Recall, and the F1 score, emphasizing their consistency when facing new data. These metrics are particularly crucial in real-world scenarios where maintaining reliable and consistent performance is essential. Table 5.3 shows the performance score of different combination of LLMs and classification algorithms used in our thesis work.

The CountVectorizer with Naive Bayes combination demonstrates commendable accuracy (0.917) and a reasonable Precision of 0.53, indicating that it correctly classifies a significant portion of positive instances. However, its Recall (0.33) suggests that there is room for improvement in capturing more actual positive cases, leading to an F1 score of 0.406. This model displays a promising foundation but may need fine-tuning to boost its ability to identify a wider range of positive cases in new data.

Similarly, the CountVectorizer with Random Forest achieves high accuracy (0.901) and a relatively higher Precision (0.66), indicating that it is cautious in making positive predictions. However, its Recall (0.142) implies that it may miss a significant portion of actual positives, reflected in its modest F1 score of 0.233. This suggests the model's need for a more balanced approach to better capture positive instances while maintaining a reasonable Precision.

In contrast, the CountVectorizer with SVM exhibits lower accuracy (0.8304) and 0

as Precision and Recall scores, respectively. The reason of can be again attributed to the imbalance in dataset and a True-positive value of 0. These results underscore challenges in effectively classifying instances using the chosen CountVectorizer features, indicating a need for reevaluation or alternative approaches.

The SpaCy-based ensemble (Bag of Words and CNN) consistently performs exceptionally well, maintaining an accuracy of 0.9280 and achieving high Precision (0.814) and Recall (0.793), leading to the F1 score of 0.803. This ensemble showcases a strong, reliable performance in handling new data, striking a commendable balance between identifying positive instances and maintaining precision.

The Keras Embedding with LSTM approach, while not surpassing the SpaCy-based ensemble, maintains a reliable performance with an accuracy of 0.8765, a decent Precision (0.765), Recall (0.723), and an F1 score of 0.743. This approach demonstrates consistency in its ability to handle new data effectively, making it a reliable choice for scenarios where a balanced performance is desired.

In conclusion, the SpaCy-based ensemble, particularly the Bag of Words and CNN combination, remains the standout performer, showcasing the importance of leveraging advanced techniques for consistent and reliable model outcomes. The discussion reinforces the significance of Precision, Recall, and the F1 score, particularly when aiming to ensure consistent performance in the face of new data, where a balanced and reliable model is of utmost importance.

5.4 Classifying requirements based on verifiability

The evaluation based solely on verifiability provides a specialized perspective on the models' performance, focusing on their ability to classify requirements based on this specific criterion. This approach is particularly valuable when the primary concern is ensuring that the generated models are adept at discerning the verifiability of requirements, a crucial aspect in many software engineering scenarios. Table 5.4 shows the performance scores of different combination of LLMs and Classifier used in this thesis work.

The CountVectorizer with Naive Bayes model, while showing decent accuracy (0.892), exhibits a lower Precision (0.214) and Recall (0.1) with an F1 score of 0.136. This suggests that the model's performance, while reasonable overall, faces challenges in accurately identifying requirements with high verifiability. The relatively low Recall indicates that a significant number of highly testable requirements may be misclassified, pointing to areas where the model can be enhanced for better coverage of such requirements.

Similarly, the CountVectorizer with Random Forest approach demonstrates an accuracy of 0.8782, but a modest Recall (0.146) implies that the model might not

Text processing techniques	ML Models	Accuracy	Precision	Recall	F1
countVectorizer	Naive Bayes	0.892	0.214	0.1	0.136
countVectorizer	Random Forest	0.8782	0.4285	0.146	0.217
countVectorizer	SVM	0.7604	1	0.133	0.235
SpaCy(Bag of Words)	SpaCy(CNN)	0.9617	0.93	0.867	0.833
Pad_sequences	LSTM	0.8338	0.70	0.83	0.76

Table 5.4: Performance scores on classifying requirements based on Testability

capture a substantial portion of requirements with high verifiability. The Precision (0.4285) suggests a degree of caution in making positive predictions, but there's a need to improve Recall to ensure that more high verifiability requirements are correctly classified.

On the other hand, the CountVectorizer with SVM model, while showcasing a high Precision (1), struggles with a low Recall (0.133). This means that the model is highly specific, possibly identifying only the most obvious instances of high verifiability while missing a considerable number of them. This pattern emphasizes the importance of balancing Precision and Recall to ensure a more comprehensive identification of high verifiability requirements.

The SpaCy-based ensemble (Bag of Words and CNN) impressively achieves an accuracy of 0.9617, indicating strong performance in classifying requirements based on verifiability. High Precision (0.93) and Recall (0.867) values, leading to an F1 score of 0.833, reinforce the robustness of this approach in capturing high verifiability requirements. This indicates that the SpaCy-based ensemble is particularly adept at identifying requirements that exhibit strong verifiability characteristics.

The Keras Embedding with LSTM combination, although not surpassing the SpaCy-based ensemble, still demonstrates respectable performance with balanced Precision (0.70) and Recall (0.83), leading to an F1 score of 0.76. This suggests that the model remains reliable in classifying requirements based on verifiability, offering a solid alternative, especially in situations where a slightly different balance between Precision and Recall is desired.

In conclusion, the SpaCy-based ensemble, notably the Bag of Words and CNN combination, stands out as a highly effective model for classifying requirements based on verifiability. This outcome underscores the importance of selecting models that align closely with the specific criterion of interest, ensuring that the generated classifications are both accurate and consistent in capturing the desired characteristic, in this case, the verifiability of requirements.

6

Discussion

This section begins by revisiting the research questions outlined in Section 1.3 to assess their answers within the context of this study. These questions aim to identify the key properties employed in reviewing requirements in the automotive industry and to assess the effectiveness of utilizing a machine learning approach for automating the requirement review process. Furthermore, this section will establish connections between this thesis and the broader field of study, proposing potential avenues for future research. We will also address potential threats to the validity of the study and acknowledge any limitations impacting the work's applicability.

6.1 RQ1: Requirements Properties to judge Software requirements

Based on the interview data collected from participants in the automotive sector, several key properties have emerged as crucial for judging software requirements in this industry. These properties—unambiguity, verifiability, and consistency—play pivotal roles in ensuring the successful development of automotive software systems that meet stringent standards and address the unique challenges of the automotive domain. These findings reflect the industry's emphasis on precision, safety, practicality, and seamless integration, highlighting the importance of these properties in guiding the creation of reliable and innovative automotive software solutions.

Clear and unambiguous requirements ensure a common understanding among stakeholders, reducing the risk of errors and improving development efficiency [2]. Verifiable requirements enable rigorous testing, critical for meeting strict standards and ensuring the software performs as expected. Consistency, on the other hand, harmonizes design decisions and interactions, resulting in a cohesive product that meets the complex needs of the automotive industry. By prioritizing these properties, software development in the automotive sector can achieve innovative, safe, and efficient solutions that adhere to industry standards while fulfilling the diverse demands of this dynamic domain.

6.2 RQ2: Use of Machine learning to automate requirement review process

The result section 5 earlier reveals the performance of several machine learning models in automating the requirement review process based on unambiguity, consistency, and testability. Among the models evaluated, the SpaCy-based ensemble (specifically, the combination of Bag of Words and CNN) consistently yielded the best F1 scores across these criteria.The potential reason of it working better than other model combinations might be the pre-trained data. The model already had a good understanding of advanced linguistic features, including named entity recognition, part-of-speech tagging, and dependency parsing, which make the model more suited to understand the requirements structure. One more potential reason might be the use of CNN, Convolutional Neural Networks are proficient in capturing local patterns in data, making them suitable for text analysis tasks. Both CNN and RNN(LSTM) can also capture features at different levels of abstraction, leading to better generalization across diverse datasets. The combination of countVectorizer and simple machine learning models were not able to perform as good as the complex models. Though they are simple to implement and does well in certain situation, they might have struggled to extract meaningful information from the data. This suggests that SpaCy ensemble approach is well-suited for automating requirement reviews, as it demonstrated high Precision, Recall, and F1 scores. This indicates that it can effectively capture and classify requirements based on the specified characteristics, making it a strong candidate for automating the requirement review process. The SpaCy-based ensemble (specifically, the combination of Bag of Words and CNN) can be very useful in an industrial setup as it performs quite well and it is also easy to implement. This trained model that we have implemented can be used as a library to review requirements automatically and software engineers can check if the requirements meet certain desirable properties.

Other works in this area have also considered using NLP techniques for requirements quality detection. [80] uses a bi-gram model to words into vectors and [81] uses Bag of Words approach, and has performed quite well, as discussed in 6.2.

Other studies in unambiguity detection have also shown exciting results in this area [80] [81]. [80] uses a decision tree to detect unambiguity and has an accuracy score of 0.91, precision of 0.91, and recall score of 0.92. [81] uses Bag-of-words with Naive Bayes to detect unambiguity in requirements and has a precision score of 0.86. Though the SpaCy-based ensemble approach has a better score in terms of accuracy but has a comparable score to precision. This comparatively lower score of precision can be attributed to the use of a smaller dataset of 1509 requirements and data imbalance. Meanwhile, most existing work focuses on unambiguity detection. There aren't any comparable results for the requirement consistency detetion and requirement verifiability.

6.3 Use of the thesis work in Industry

Though significant work has been conducted in this thesis work to contribute towards requirement review process. We also think, this study will be very useful when it comes to industry. Here we try to present the practical scenario of how this work can be used in industrial setup.

Customers send requirements to the client company in the initial phase. Based on the requirements received in the client side, the developers develop features of the software and deliver it to the customer. These requirements are written using natural language by engineers working in the customer side. These requirements are validated in the client side using requirement review processes. At Aptiv, the requirements are reviewed by system engineers manually and then sent to developers. The models trained in this thesis work can be used by system engineers and software developers during the review process as a tool to check if the requirements meet the most desirable properties.

6.4 Threats to Validity

6.4.1 Internal Threats

During the performing thematic analysis period, in the case of coding to find themes, there might be a risk of bias. It is only two individuals when performing the coding, but we have to try to reduce the threats of validation by coding separately and trying to validate each other's codes twice, then discussing and deciding codes.

There is a potential threat due to your relative lack of experience in requirements when we perform coding. But we studied a lot of previous research work related to similar subject and discuss it with the supervisor regularly to mitigate the risk of threats.

We acknowledge that the utilization of a smaller dataset in this study poses a potential threat to the validity of our findings as we only have 1509 requirements collected in 3 different projects. However, we have taken several measures to mitigate this threat. First of all, the requirement collected in 3 different projects covers various aspects of the requirements in the automotive domain. Apart from that, we have employed cross-validation techniques across different subsets of the data.

We know requirement labeling by different experts in this study could be a potential threat to the validity of our findings. In order to mitigate this threat, we have asked three domain experts to relabel the requirements and have used Cohen's Kappa to calculate the similarity between the labels and to ensure consistency and reliability. All three Cohen's Kapper values are between 0.6 to 0.8, recognized as substantial agreement [84].

Data imbalance for the requirement dataset also considers a potential threat. Because of the nature of the requirement dataset, the proportion of requirements that are labeled 0 occupied little in the requirement dataset. However, some methods are used to reduce this threat of validation. First, we use different evaluation metrics to evaluate the model's performance. By evaluating the performance, we use diverse metrics such as precision, recall, and F1-score instead of only using accuracy. Apart from that, we also use different machine learning algorithms to do the binary classification.

6.4.2 External Threats

The scope of this study is focused on the analysis and evaluation of requirements in the context of automotive software. As a result, the characteristics and metrics that will be used to determine the quality level of the requirements will be specifically tailored and biased toward the unique features and requirements of the automotive industry. It is important to acknowledge that this bias towards automotive software may limit the generalizability of the study's findings to other software domains, as the requirements and quality metrics used in the automotive industry may differ significantly from those used in other sectors. However, by identifying and analyzing the unique requirement properties and metrics relevant to the automotive industry, the study aims to provide a more precise and targeted review for improving the quality of requirements in this specific domain, which can potentially inform and benefit other industries that share similar requirements and characteristics.

6.4.3 Future work

The findings from the current study pave the way for several intriguing avenues of future research in the realm of automated requirement review. Leveraging the insights gathered from the evaluation of machine learning models and language model representations, several directions can be explored to enhance the effectiveness and robustness of requirement review automation.

To provide a more comprehensive and insightful comparative analysis, training and evaluating a broader spectrum of machine learning models would be highly beneficial. Beyond the models examined in this study, future research could include ensemble models, gradient boosting algorithms, or even more complex deep learning architectures, ensuring that a thorough exploration of the model landscape is conducted. Such an approach will offer a deeper understanding of the strengths and weaknesses of various models, ultimately guiding the selection of the most effective models for automated requirement review.

Primarily, an intriguing path involves broadening the scope of exploration in NLP techniques by delving into a wider array of techniques beyond those scrutinized in this study. While the SpaCy-based ensemble, notably the Bag of Words approach, demonstrated superior performance, there exist numerous other NLP techniques, in-

cluding TF-IDF, Word2Vec, GloVe, and BERT, which hold the potential to further enrich the preservation of semantic meaning in requirements. Exploring these additional techniques in conjunction with diverse machine learning models may unearth latent patterns and subtle intricacies in requirement attributes, potentially leading to more precise and reliable automated reviews.

Incorporating a variety of pre-trained models is another promising avenue for future work. While the current study evaluated several models, the landscape of pre-trained models is rapidly evolving. Exploring the potential of emerging pre-trained models, such as T5, XLNet, or RoBERTa, offers the opportunity to harness more advanced semantic understanding, which can be pivotal in capturing intricate requirement attributes.

Furthermore, enhancing the quality and size of the training dataset can significantly impact model performance. Acquiring a larger and more diverse dataset that encompasses a broader range of requirements, contexts, and domains will contribute to the generalization of the developed models. This expansion would help alleviate potential biases and limitations arising from the current dataset, making the automated requirement review system more adaptable and versatile.

7
Conclusion

In this thesis work, an attempt has been made to automate the requirement review process in automotive industry using machine learning. In order to establish an understanding on how experts in automotive industry manually review requirements, we interviewed nine industrial experts, with questions focusing on the properties they the requirements with, the current strategy in place to review requirements and problems they face while reviewing requirements or due to badly written requirements. The data collected during the interviews helped us answer the RQ1 and also formed the base for model training.

We have trained five combination of large-language models and classification models. Three classical machine learning models like SVM[10], Naive-Bayes[70] and Random forest[71] have been used with combination with countVectorizer. SpaCy based ensamble has been implemented that uses Bag of Words and CNN to perform text classification. We have also implemented LSTM model in combination with Word Embedding by Keras and pad_Sequences. Among the models evaluated, the SpaCy-based ensemble consistently yielded the best F1 scores across these criteria. The combination of LSTM and Word Embedding also yielded satisfactory result. These results enabled us to answer RQ2.

In conclusion, the integration of the SpaCy-based ensemble, particularly the fusion of Bag of Words and Convolutional Neural Networks, presents a highly valuable asset within an industrial context. Its impressive performance, coupled with its straightforward implementation, renders it an indispensable tool. This proficiently trained model can make the requirements review process easier by automating the task and mitigating the manual requirement review in-place. By empowering software engineers to evaluate whether requirements align with desired properties, this solution not only streamlines processes but also ensures a higher level of standardization and quality assurance in industrial workflows.

Bibliography

[1] Loucopoulos P, Karakostas V (1995) Systems requirements engineering. McGraw-Hill, New York LaTeX.

[2] Sarah c. Gregory, John Terzakis. Intel. Writing Good Requirements. 2017 IEEE International Requirements Engineering ConferenceLisboa, Portugal

[3] IEEE Computer Society. (2014). SWEBOK guide to the software engineering body of knowledge, Version 3.0 (https://www.computer.org/education/bodies-ofknowledge/software-engineering).

[4] Eito-Brun, R., Amescua, A. (2017). Dealing with software process requirements complexity: an information access proposal based on semantic technologies. Requirements Engineering, 22(4), 527–542

[5] Systems and software engineering — Life cycle processes — Requirements engineering, ISO/IEC/IEEE 29148, 12-13.

[6] Nuseibeh B, Easterbrook S. Requirements engineering: a roadmap[C]//Proceedings of the Conference on the Future of Software Engineering. 2000: 35-46.

[7] Kotonya G, Sommerville I. Requirements engineering: processes and techniques[M]. Wiley Publishing, 1998.

[8] Bøegh J. A new standard for quality requirements[J]. IEEE software, 2008, 25(2): 57-63.

[9] Zhou Z H. Machine learning[M]. Springer Nature, 2021.

[10] Noble W S. What is a support vector machine?[J]. Nature biotechnology, 2006, 24(12): 1565-1567.

[11] Renita Priya, Xinyuan Wang, Yujie Hu, and Yu Sun. A deep dive into automatic code generation using character based recurrent neural networks. In 2017 International Conference on Computational Science and Computational Intelligence (CSCI), pages 369–374, 2017.

[12] Staudemeyer R C, Morris E R. Understanding LSTM–a tutorial into long short-term memory recurrent neural networks[J]. arXiv preprint arXiv:1909.09586, 2019.

[13] dProgrammer lopez. Rnn, lstm gru, 2019.

[14] Aditi Mittal. Understanding RNN and LSTM, https://aditi-mittal.medium.com/, Oct 12, 2019.

[15] Zhang D, Tsai J J P. Machine learning and software engineering[J]. Software Quality Journal, 2003, 11: 87-119.

[16] Wiegers K, Beatty J. Software requirements[M]. Pearson Education, 2013.

[17] Eugenio Parra, Christos Dimou, Juan Llorens, Valentín Moreno, Anabel Fraga. A methodology for the classification of quality of requirements using machine learning techniques, Information and Software Technology.

[18] Chowdhary K R, Chowdhary K R. Natural language processing[J]. Fundamentals of artificial intelligence, 2020: 603-649.

[19] Gustavo Carvalho, Diogo Falcão, Flávia Barros, Augusto Sampaio, Alexandre Mota, Leonardo Motta, and Mark Blackburn. Nat2testscr: Test case generation from natural language requirements based on specifications. Science of Computer Programming, 95:275–297, 2014. Special Section: ACM SAC-SVT 2013 + Bytecode 2013.

[20] Alzahraa Salman. Test Case Generation from Specifications Using Natural Language Processing. PhD thesis, 2020.

[21] Malik P, Mittal V, Nautiyal L, et al. NLP techniques, tools, and algorithms for data science[J]. Artificial Intelligence for Signal Processing and Wireless Communication, 2022, 11: 123.

[22] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Neural computation, 18(7):1527–1554, 2006.

[23] O'Shea K, Nash R. An introduction to convolutional neural networks[J]. arXiv preprint arXiv:1511.08458, 2015.

[24] Tahira Iqbal, Parisa Elahidoost, Levi Lucio. A Bird's Eye View on Requirements Engineering and Machine learning, 2018 25th Asia-Pacific Software Engineering Conference (APSEC).

[25] Vasiliev Y. Natural language processing with Python and spaCy: A practical introduction[M]. No Starch Press, 2020.

[26] Génova G, Fuentes J M, Llorens J, et al. A framework to measure and improve the quality of textual requirements[J]. Requirements engineering, 2013, 18: 25-41.

[27] Liang J. Confusion matrix: Machine learning[J]. POGIL Activity Clearinghouse, 2022, 3(4).

[28] Canbek G, Sagiroglu S, Temizel T T, et al. Binary classification performance measures/metrics: A comprehensive visualized roadmap to gain new insights[C]//2017 International Conference on Computer Science and Engineering (UBMK). IEEE, 2017: 821-826.

[29] Davis J, Goadrich M. The relationship between Precision-Recall and ROC curves[C]//Proceedings of the 23rd international conference on Machine learning. 2006: 233-240.

[30] H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS

[31] Braun V, Clarke V. Using thematic analysis in psychology[J]. Qualitative research in psychology, 2006, 3(2): 77-101.

[32] Chantree F, Nuseibeh B, De Roeck A, et al. Identifying nocuous ambiguities in natural language requirements[C]//14th IEEE International Requirements Engineering Conference (RE'06). IEEE, 2006: 59-68.

[33] Kiyavitskaya N, Zeni N, Mich L, et al. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications[J]. Requirements engineering, 2008, 13: 207-239.

[34] de Sousa T C, Almeida Jr J R, Viana S, et al. Automatic analysis of requirements consistency with the B method[J]. ACM SIGSOFT Software Engineering Notes, 2010, 35(2): 1-4.

[35] Ko Y, Park S, Seo J, et al. Using classification techniques for informal requirements in the requirements analysis-supporting system[J]. Information and Software Technology, 2007, 49(11-12): 1128-1140.

[36] Domingos P. A few useful things to know about machine learning[J]. Communications of the ACM, 2012, 55(10): 78-87.

[37] Duda R O, Hart P E, Stork D G. Pattern classification and scene analysis[M]. New York: Wiley, 1973.

[38] Rennie J D, Shih L, Teevan J, et al. Tackling the poor assumptions of naive bayes text classifiers[C]//Proceedings of the 20th international conference on machine learning (ICML-03). 2003: 616-623.

[39] Sahami M. Learning Limited Dependence Bayesian Classifiers[C]//KDD. 1996, 96(1): 335-338.

[40] McCallum A, Nigam K. A comparison of event models for naive bayes text classification[C]//AAAI-98 workshop on learning for text categorization. 1998, 752(1): 41-48.

[41] Joachims T. Text categorization with support vector machines: Learning with many relevant features[C]//Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005: 137-142.

[42] Wang S I, Manning C D. Baselines and bigrams: Simple, good sentiment and topic classification[C]//Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). 2012: 90-94.

[43] Quinlan J R. Induction of decision trees[J]. Machine learning, 1986, 1: 81-106.

[44] Breiman L, Friedman J, Olshen R, et al. Classification and regression trees–crc press[J]. Boca Raton, Florida, 1984.

[45] Chen Y. Convolutional neural network for sentence classification[D]. University of Waterloo, 2015.

[46] Zhang Y, Wallace B. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1510.03820, 2015.

[47] Mikolov T, Karafiát M, Burget L, et al. Recurrent neural network based language model[C]//Interspeech. 2010, 2(3): 1045-1048.

[48] Breiman L. Random forests[J]. Machine learning, 2001, 45: 5-32.

[49] Heck, P., Zaidman, A. (2018). A systematic literature review on quality criteria for agile requirements specifications. Software Quality Journal, published online 15 September 2016.

[50] Bernard H R. Research methods in anthropology: Qualitative and quantitative approaches[M]. Rowman Littlefield, 2017.

[51] Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., Moreno, V. (2013). A framework to measure and improve the quality of textual requirements. Requirements Engineering, 18(1), 25–41.

[52] F.J. Chantree, A. De Roeck, B. Nuseibeh, A. Willis, Identifying Nocuous Ambiguity in Natural Language Requirements, Fac. Maths Comput. Doctor of, 2006, p. 203.

[53] T.C. de Sousa, J.R. Almeida, S. Viana, J. Pavón, Automatic analysis of requirements consistency with the B method, ACM SIGSOFT Softw. Eng. Notes 35 (2010) 1, http://dx.doi.org/10.1145/1734103.1734114.

[54] D. Aceituna, G. Walia, H. Do, S.-W. Lee, Model-based requirements verification method: conclusions from two controlled experiments, Inf. Softw. Technol. 56 (2014) 321–334, http://dx.doi.org/10.1016/j.infsof.2013.11.004.

[55] Saldaña J. The coding manual for qualitative researchers[J]. The coding manual for qualitative researchers, 2021: 1-440.

[56] H. Jani, A. Islam, A Framework of Software Requirements Quality Analysis System using Case-Based Reasoning and Neural Network, Ieeexplore.Ieee.Org., n.d., pp. 152–157

[57] Clarke V, Braun V. Teaching thematic analysis: Overcoming challenges and developing strategies for effective learning[J]. The psychologist, 2013, 26(2).

[58] Thomas D R. A general inductive approach for analyzing qualitative evaluation data[J]. American journal of evaluation, 2006, 27(2): 237-246.

[59] DiCicco-Bloom B, Crabtree B F. The qualitative research interview[J]. Medical education, 2006, 40(4): 314-321.

[60] Burns N. The practice of nursing research: conduct, critique & utilization[M]. WB Saunders Co, 1993.

[61] Braun, V., Clarke, V. (2006). Using thematic analysis in psychology. Qualitative Research in Psychology, 3(2), 77-101. https://doi.org/10.1191/1478088706qp063oa

[62] Systems and software engineering — Life cycle processes — Requirements engineering ISO/IEC/IEEE 29148 Second edition 2018-11

[63] Suri H. Purposeful sampling in qualitative research synthesis[J]. Qualitative research journal, 2011, 11(2): 63-75.

[64] Palinkas L A, Horwitz S M, Green C A, et al. Purposeful sampling for qualitative data collection and analysis in mixed method implementation research[J]. Administration and policy in mental health and mental health services research, 2015, 42: 533-544.

[65] Sosik, J. J., Godshalk, V. M. (2000). Leadership styles, mentoring functions received, and job-related stress: A conceptual model and preliminary study. Journal of Organizational Behavior, 21(4), 365-390.

[66] Runeson, P., Host, M., Rainer, A., Regnell, B. (2012). Case study research in software engineering: Guidelines and examples. John Wiley Sons.

[67] "Software Requirements (Third Edition)" by Karl Wiegers and Joy Beatty. Microsoft.

[68] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks.

[69] SpaCy, https://spacy.io/models

[70] SK-learn.Naive_Bayes, Accessed: 16, May, 2023. Available: https://scikit-learn.org/stable/modules/naive_bayes.html

[71] sklearn.ensemble.RandomForestClassifier, Accessed: 16, May, 2023. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[72] Sklearn.model_selection.randomizedsearchcv, Accessed: 16, May, 2023. Available: https : / / scikit - learn . org / stable / modules / generated / sklearn.model_selection.RandomizedSearchCV.htmlsklearn.model _selection.RandomizedSearchCV.

[73] Cross-validation: evaluating estimator performance, Accessed: 17, May, 2023. https://scikit-learn.org/stable/modules/cross_validation.html

[74] Sepp Hochreiter, Jurgen Schmidhuber. Long Short-term Memory. Software Quality Journal, Neural Computation, 1997.

[75] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, Francis C.M. Lau. A C-LSTM Neural Network for Text Classification, https://doi.org/10.48550/arXiv.1511.08630.

[76] Abid Ali Awan, Recurrent Neural Network Tutorial (RNN), Accessed: Mar 2022. Available: https://www.datacamp.com/tutorial/tutorial-for-recurrent-neural-network

[77] Sidharth, Convolutional Neural Network (CNN): Architecture Explained | Deep Learning Accessed: July 10, 2023. Available: https://www.pycodemates.com/2023/06/introduction-to-convolutional-neural-networks

[78] en_core_web_sm, SpaCy, https://github.com/explosion/spacy-models/releases/tag/en_core_web_sm-3.6.0

[79] CountVectorizer, Scikit-learn, https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer

[80] Polpinij J, Ghose A. An automatic elaborate requirement specification by using hierarchical text classification. Proceedings International Conference on Computer Science and Software Engineering, CSSE 2008. 2008;1:706-9.

[81] Sharma R, Bhatia J, Biswas KK. Machine learning for constituency test of coordinating conjunctions in requirements specifications. Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering - RAISE 2014. 2014:25-31.
Unambiguity_NB

[82] Maroulis G. Comparison between Maximum Entropy and Naïve Bayes classifiers : Case study ; Appliance of Machine Learning Algorithms to an Odesk ' s Corporation Dataset Georgios Maroulis Submitted in partial fulfilment of the requirements of Edinburgh Napier University. 2014(January).

[83] Polpinij J. An ontology-based text processing approach for simplifying ambiguity of requirement specifications. 2009 IEEE Asia-Pacific Services Computing Conference, APSCC 2009. 2009:219-26.

[84] McHugh M L. Interrater reliability: the kappa statistic[J]. Biochemia medica, 2012, 22(3): 276-282.

[85] Neri Van Otten, CountVectorizer Tutorial In Scikit-Learn And Python (NLP) With Advantages, Disadvantages Alternatives. https://spotintelligence.com/2023/05/17/countvectorizer/Advantages_and_disadvantages

[86] Pavan Vadapalli Naive Bayes Explained: Function, Advantages Disadvantages, Applications in 2023. https://www.upgrad.com/blog/naive-bayes-explained/
[87] Random forest Algorithm in Machine learning: An Overview. https://www.mygreatlearning.com/blog/random-forest-algorithm/

A

Appendix 1