CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Anaphoric Ambiguity Detection in Software Requirements Specification

by

Haram Tanveer

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2024

I dedicate this thesis to the pillars of my life, My Family and My Teachers, whose unwavering support and guidance have shaped my academic journey. A heartfelt expression of gratitude goes to My beloved parents, whose boundless love, prayers, and encouragement have been my constant driving force. Their belief in my abilities has been the foundation of my success in every aspect of life. I extend my sincere thanks to my supervisor, whose unyielding confidence, mentorship, and invaluable insights have propelled me to reach this significant milestone. A special dedication is reserved for my husband, whose unwavering support, understanding, and encouragement have been my source of strength throughout this academic endeavor. His presence and belief in my capabilities have been a guiding light, making this achievement possible. This work is a reflection of the collective efforts and encouragement of these remarkable individuals, and I am profoundly grateful for their presence in my life.

CERTIFICATE OF APPROVAL

Anaphoric Ambiguity Detection in Software Requirements Specification

by

Haram Tanveer

(MCS203035)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Onaiza Maqbool	QAU, Islamabad
(b)	Internal Examiner	Dr. Abdul Basit	CUST, Islamabad
(c)	Supervisor	Dr. Aamer Nadeem	CUST, Islamabad

Dr. Aamer Nadeem
Thesis Supervisor
June, 2024

Dr. Abdul Basit
Head
Department of Computer Science
June, 2024

Dr. M. Abdul Qadir
Dean
Faculty of Computing
June, 2024

Author's Declaration

I, **Haram Tanveer** hereby state that my MS thesis titled "**Anaphoric Ambiguity Detection in Software Requirements Specification**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

(Haram Tanveer)
Registration No: MCS203035

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "**Anaphoric Ambiguity Detection in Software Requirements Specification**" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

(Haram Tanveer)
Registration No: MCS203035

Acknowledgement

I express my deepest gratitude to Allah Almighty for endowing me with the wisdom and strength needed to complete this dissertation. Being an MS graduate at Capital University of Science and Technology has been both magnificent and challenging, and I am grateful for the experiences and growth it has provided.

My heartfelt thanks go to my dedicated supervisor, Dr. Aamir Nadeem, whose guidance, assistance, and profound knowledge have been instrumental in shaping this research. I am sincerely grateful for his constant support, motivation, and unwavering patience. His invaluable feedback and constructive suggestions have significantly contributed to the success of this thesis.

A special acknowledgment is reserved for my family, whose constant motivation has been a driving force behind reaching this milestone. I extend a word of applause to my friends and classmates for their assistance in sharing knowledge and resources crucial for conducting this research.

Lastly, I want to express my deepest gratitude to my husband. His unyielding support, encouragement, and understanding have been my pillars of strength. His belief in my abilities has been a guiding light, and I am profoundly thankful for his presence throughout this academic endeavor.

(Haram Tanveer)

Abstract

This study tackles the vital dilemma of Anaphoric Ambiguity detection for Software Requirement Specifications (SRS). The research opens with a clarification of the importance of anaphoric ambiguity in software development projects, which contributes to the accuracy of requirement interpretation. A comprehensive literature review recognizes the limitations associated with contemporary methodologies, thus unveiling a need for proposed research. The well-organized methodology described in the research includes strong research design, large data collection procedures, and ethics issues.

At the center of the research is an elaborate NLP pipeline, including 8 modules for a detailed linguistic analysis of SRS. The basis of the experimentation is the dataset, DAMIR, and a rich landscape of different machine learning algorithms that are subjected to meticulous evaluation. The pinnacle of these efforts is experimentation, which reveals positive findings. The accuracy, precision, recall, and F-measure percentages demonstrate the validity of the anaphoric ambiguity detection model.

This work not only advances the theoretical knowledge about anaphoric ambiguity in SRS but also offers practical tools to improve the accuracy of the software development process. In conclusion, the study reflects on possible directions for further research, stating that these outcomes can improve efficiency and success of future software development projects.

Contents

List of Figures

List of Tables

Abbreviations

ADA Boost	Adaptive Boosting
BERT	Bidirectional Encoder Representations from Transformers
DAMIR	Dataset for Anaphoric Ambiguity Detection in Requirements
DT	Decision Tree
FNN	Feed-forward Neural Network
GPT	Generative Pre-trained Transformer
IDE	Integrated Development Environment
kNN	k-Nearest Neighbors
LR	Logistic Regression
MCC	Matthews Correlation Coefficient
NB	Naïve Bayes
NLP	Natural Language Processing
NPS	Noun Phrases
NP	Noun Phrase
POS	Part-of-Speech
PRC	Precision-Recall Curve
PRP	Personal Pronoun
RF	Random Forest
ROC	Receiver Operating Characteristic
SRS	Software Requirements Specifications
SVM	Support Vector Machine

Chapter 1

Introduction

A branch of software engineering (SE), called requirements engineering (RE), aims to define and document the objectives, desired properties, capabilities, and limitations of software-intensive systems. Requirements effectiveness determines project success to a great extent, particularly by mitigating issues such as delays in projects beyond budgets, mismatched expectations by users, and unreliable systems. Historical events, including the failure of the FBI Whit File project in 2000, which ghastly cost $257 million, are enough to underline the consequences of insufficient requirements [1].

In the vicinity of RE lies the SRS software requirements specification, which is a key document describing expected features of the to-be system. This document is meant for product managers, subject-matter experts, and developers. Most of the SRSs prefer the NL medium since it helps to build common understanding among diverse stakeholders who have different backgrounds and fields of specialisation.

On the other hand, Natural Language (NL) requirements are highly vulnerable to many quality defects, such as ambiguity, incompleteness, and inconsistency [2]. The primary objective of this dissertation is to use AI-powered automation to boost the performance of NL requirements evaluations by requirements engineers. A particular emphasis is placed on addressings ambiguity – a common challenge in NL where a piece of text can be interpreted in multiple ways. Manual ambiguity resolution is arduous, time-consuming, and error-prone, demanding domain

knowledge for accurate interpretation [3]. Since domain-specific vocabularies are utilized in various SRSs, the integration of domain knowledge into automated ambiguity handling is crucial for precision.

The present research in Requirements Engineering (RE) is constrained by five primary limitatiions:

Despite the widespread existence of ambiguity within requirements, the current body of RE work lacks dedicated efforts towards managing specific types of syntactic ambiguity addressed in this study. These ambiguity types encompass coordination, prepositional-phrase attachment, and anaphoric ambiguity.

The ongoing RE research predominantly centers around the identification of ambiguity in requirements. However, it falls short in terms of offering guidance or suggestions pertaining to the potential interpretations of these ambiguous requirements.

Prevailing domain-specific methods designed for detecting ambiguity in requirements are not easily adaptable to domains beyond their original development contexts. The absence of automated mechanisms impedes the extension of such solutions to address ambiguity across diverse domains.

Automated approaches employing natural language processing (NLP) fail to fully capitalize on the capabilities of NLP. Recent advancements in the the NLP domain, particularly the widespread utilization of large-scale language models like BERT [4], give rise to opportunities for novel RE automation solutions as well as a reevaluation of existing methodologies.

This limitation is interconnected with the previous one. Within RE, the exploration of question answering (QA) is confined within narrow boundaries [5], and as far as our knowledge extends, it has not yet been examined in the contexts of ensuring the quality of requirements.

The characteristics from ISO 29148 [6] for the quality assessment of natural language requirements are listed below:

Complete: The stated requirement ought no further amplification because it is measurable and adequately designates the ability and characteristics to fulfill the stakeholder's requirement.

Consistent: The need is free of disagreements with other requirements.

Feasible: The need is technically possible, does not need major technological advancements, and fits within system limitations with acceptable risk.

Implementation Free: The objective is to achieve implementation neutrality. The requirement specifies the necessary outcome, without specifying the specific methods or approaches to attain it.

Necessary: The requirement delineates a fundamental ability, attribute, constraint, or criterion of quality. The condition remains relevant and has not been rendered obsolete by the passage of time.

Singular: The requirement declaration consists of a single requirement without any conjunctions.

Traceable: The requirement is upward traceable to explicit documented stakeholder statement(s) of demand. The requirement is also downwards traceable to the explicit requirements in the lower tier requirements specification or other system description artifacts.

Unambiguous: The requirement is communicated in such a way so that it can be understood in solely one way. The requirement is expressed simply and is easy to understand.

Inconsistency arises when a specification includes conflicting, contradictory narratives of the expected behavior of the system to be constructed or of its domain. Such conflicting narratives. may arrive (a) because of conflicting goals between the different parties that contribute to the specification (stakeholders), or (b) as a result of uncoordinated changes raised in the specification during the usual expansion of the requirements. Inconsistency is a significant problem that pervades all aspects of software development. It makes it unattainable to devise and implement a system that admires its specification. In some cases, inconsistencies may

be promptly determined by the programmer during the execution of the system. This indicates that one of the inconsistent requirements will be preferred over the others, usually without performing in-depth research of the results. Undetected inconsistency may direct to incorrect and unreliable systems, whose flaws are discovered only when it is too late usually during operation.

1.1 Ambiguity

Ambiguity is a pervasive occurrence in human languages and is fundamentally a possession of linguistic phrases. There are different definitions of ambiguity: "the capability of being understood in two or more possible senses or ways" and "uncertainty". Uncertainty indicates a lack of sureness regarding something, often due to gaps in the writer's or reader's or both's knowledge. A word, phrase, or sentence is termed ambiguous if it can be reasonably analyzed in more than one method. It is challenging to find a word that does not have at least two potential meanings, and an isolated sentence, detached from its context, is often ambiguous.

Ambiguity is an intrinsic attribute of natural language (NL) and a vital characteristic that makes NL flexible in various contexts. It refers to the understanding of text to hold numerous interpretations. As the majority of the artifacts for requirements analysis are written in natural language [7], in requirements engineering (RE), ambiguity has thus long been identified as a challenge.

Multiple understandings of the requirements can direct to incorrect implementation, especially in cases of unacknowledged ambiguity. Formal and constrained languages have been suggested as an alternative to providing structures for evading ambiguity, however, they lack the richness of NL for the expressiveness of concepts. A comprehensive investigation of the nature of the ambiguity phenomena in requirements specifications is carried out in [7]. They directed a detailed examinations of the relationship between ambiguity and two other phenomena, that of abstraction and the absence of information. This in-depth analysis of

various forms of ambiguity has resulted in providing a characterization of the different levels at which ambiguity can be personified in requirements specification documents. Various types of ambiguities are:

TABLE 1.1: Types of ambiguities [7]

Type of ambiguity	**Description**
Lexical Ambiguity	Based on a single word e.g. school, bank
Syntactic Ambiguity	Structural ambiguity
Semantic Ambiguity	Logical ambiguity
Pragmatic Ambiguity	Multiple interpretations of a sentence

Alessio [8] discussed the following types of defect classes for different types of ambiguity:

Anaphoric ambiguity - Anaphora arises in a text whenever a pronoun (e.g., he, it, that, this, etc.) denotes to a previous part of the text. The directed part of the text is normally called the antecedent (e.g., The system shall send a message to the receiver, and it shall provide an acknowledge message - it = system or receiver?). The possible antecedents for the pronouns are noun phrases (NP), which can be identified using a shallow parser.

Coordination ambiguity - This happens when the use of the coordinating conjunctions (e.g., and, or) directs to multiple possible interpretations of a sentence.

Vague terms - Vagueness arises whenever a sentence acknowledges borderline cases, i.e., cases in which the truth value of the sentence can't be decided.

Modal adverbs - These (e.g., positively, permanently, clearly) are modifiers that define a quality related to a predicate.

Passive voice - The usage of passive voice is a deficiency of clarity in requirements, and can guide to ambiguous interpretations in those cases in which the passive verb is not heeded by the subject that perform the action uttered by the verb (e.g., The system shall be shut down – by which actor?).

A clear and unambiguous expression of requirements stands as a pivotal factor that ultimately paves the way for the successful development of software. While it's possible to articulate software requirements using a formal language, this approach hasn't been universally adopted to ensure the comprehensibility of Software Requirements Specifications (SRS) and other related documents. Consequently, alternative methods are favored to mitigate the presence of ambiguities.

Despite the emergence of insightful disambiguation tools proposed by different authors over time, a void exists in the analysis and evaluation of these tools. Natural Language Processing-based methods are still undergoing refinement.

1.2 Natural Language Processing

Algorithms cannot directly analyze textual data due to the fact that "text often comprises unstructured documents lacking specific rules for document composition". Nevertheless, computers possess the capability to sift through extensive document collections and extract pertinent information from words and paragraphs. A vast body of literature exists concerning the examinations of text data. Pioneering contributions in this domain were made by Manning and Schütze [9], who laid the groundwork for statistical Natural Language Processing (NLP) and provided valuable tools. To extract information, it's necessary to convert the text into a numerical form. This transformation process involves two distinct stages: text preprocessing and feature engineering. The specific preprocessing tasks vary based on the textual attributes intended for representation.

Presently, Natural Language Processing (NLP) is a thoroughly studied field with a multitude of applications. A prominent instance is Siri, an automated assistant integrated into Apple's iPhone, capable of executing mobile tasks like contacting individuals or sending text messages, while also providing a responses based on user input, whether textual or spoken. This encompasses a wide array of NLP techniques, including speech recognition, disambiguation of word meanings, natural language generation, and text-to-speech capabilities. What this quotation

underscores is the significance of context within natural language. In a conversation, people have preknowledge about the topic under discussion, how the words used are understood and who they are conversing with. In addition, contextual clues are embedded in the dialogue, including the tone of an utterance or words that are intentionally accented by a speaker. The best way to explain such a sentence is 'My mom will kill me if she hears about it'. Literally, this indicates that in case a mother learns of something her reaction would be so drastic as murdering her child. Nevertheless, in this setting kill is a hyperbole by 'delivering a strong caution' most likely of what the child has done. However, the real motivation for the mother's response remains unclear.

However, usually it becomes clear that 'kill' is used figuratively meaning a response of the strong reaction not literal death. It is precisely this ambiguity of interpretation brought by context that constitutes a problem for computers in processing sentences. Since computers are unlikely to change the way they understand sentences depending on context and subject, it is clear that NLP has a long path in front of it.

Natural Language Processing (NLP) is a branch of Artificial Intelligence that refers to the use of computer technologies aimed at the handling and interpretation of natural language found in speech or written communication. NLP requirements are crucial in the software industry as well, especially at the beginning of system description within software development. Sadly, these programs frequently have the glitches in them that even some stem from an understanding natural language itself. Ambiguities are the main sources of such language-related problems. Addressing this, the Quality Analyzer for Requirement Specification (QARS) tool has been developed to automatically analyze natural language requirements.

The effectiveness of software largely hinges on how accurately it reflects its specification content. Given that this content is derived from actual user needs, effectively encapsulating it in a way that eliminates ambiguities is a critical task. Software requirements can be articulated in two primary ways: firstly, through a formal specification approach, and secondly, through an informal specification

approach. Both formal and semiformal languages offer a benefits in reducing ambiguities to some extent. However, the semiformal approach is often favored due to its alignment with stakeholders' skillsets; these requirements are subsequently translated into formal languages.

In the case of the informal approach, various techniques rooted in Machine Learning, ontology, and linguistics are subsequently employed to mitigate ambiguity errors.

In our research, we predominantly use an NLP pipeline comprising the subsequent components: (1) Tokenizer, responsible for dividing the text into individual tokens. For instance, the sentence "what time is it?" is tokenized into five units: [What, time, is, it,?]; (2) Sentence Splitter, used to segment the text into distinct sentences; (3) Part-Of-Speech (POS) Tagger, tasked with assigning POS tags, such as nouns, verbs, or pronouns, to each token within every sentence; (4) Lemmatizer, which identifies the base form (lemma) of each token. For instance, the lemma for "playing" is "play"; (5) Constituency Parser, employed to detect structural units within sentences, like verb phrases and noun phrases; (6) Dependency Parser, responsible for identifying grammatical relationships between tokens in sentences, such as subjects and objects; (7) Coreference Resolver, utilized to find references to the same entities within the text; and finally, (8) Semantic Parser, employed to extract information regarding the meanings of words.

1.3 Machine Learning

Machine learning is the area of artificial intelligence that has the goal of creating models based on a set of example data that is training data. These models subsequently use this knowledge to generate predictions and decisions based on new, unseen data (test data) [10].

Machine learning systems can be classified into four categories based on the kind of guidance they receive during their training. (1) Supervised learning algorithms necessitate the provision of desired solutions (labels) within the training data.

This category involves two primary tasks: classification, dealing with categorical targets, and regression, handling continuous numerical targets. (2) In contrast, unsupervised learning algorithms do not depend on training labels and strive to learn autonomously without external guidance. (3) Semisupervised learning algorithms operate with partially labeled data alongside predominantly unlabeled data, often blending supervised and unsupervised methodologies. (4) Reinforcement learning systems center around agents that observe their surroundings, take actions, and receive positive or negative rewards for each action [11].

This thesis predominantly engages with supervised methods. Within this context, we provide a concise overview of supervised learning models as well as introduce additional concepts that we experiment with, like addressing data imbalances and utilizing ensembling methods. These algorithms acquire knowledge from labeled data, utilizing input and corresponding output information. This approach facilitates the automatic classification of unfamiliar (unlabeled) data. Autonomous evaluation of the quality of requirements is based on input information that includes several parameters, whereas for output our data are evaluations by experts for each requirement about 'ambiguous' and 'Unambiguous' quality attributes. This means that we can employ supervised machine learning methods with success.

In NLP, machine learning algorithms play a critical role—especially when dealing with complex problems such as anaphoric ambiguity detection in natural requirements language. Machine learning methods offer a novel solution to the conundrum of language, where anaphoric ambiguities are resolved specifically. This section outlines a importance of machine learning algorithms for anaphoric ambiguity detection in improving natural requirements linguistics through software development.

1. Complex Linguistic Structures: The features of natural language being a dependent relationship that is complex because linguistic structures are often ambiguous in nature make manual analysis and rule-based approaches incapable of providing an adequate identification of the anaphoric ambiguities present. However, machine

learning algorithms excel at identifying the intricate patterns and links that occur in language, making it much easier to identify subtle allusions that otherwise would have gone unnoticed when using a human-written rules.

2. Contextual Understanding: What is anaphoric ambiguity depends, essentially, on the environment in which terms are used and their affiliation to entities previously mentioned. The algorithms implement machine learning, getting much context according to which they make the most probable selection out of miscellaneous possible antecedents for anaphoric references and thereby increasing precision. This contextual comprehension and processing capability has great value in the software requirement domain where perceptive of context is primary prerequisite for effective communication.

3. Scalability and Generalization: Manual detection of anaphoric ambiguities across a large dataset is labor-intensive and time-consuming, because the scope for natural language requirements can be vast. It is possible to train machine learning algorithms with a variety of data sets so that generalization occurs, and anaphoric ambiguities can be detected in many contexts, languages, or domains on a larger scale and thereby making the process more efficient.

4. Learning from Data: The labeled datasets for machine learning algorithms are annotated by human experts to reflect anaphoric ambiguities. This learning process allows algorithms to absorb the nuances of language and context, so they can make better decisions in real-life settings. The more the algorithm learns, the better it becomes in its accuracy while diminishing any need for human intervention.

5. Adaptability: Language is not static but changes over time, resulting in different ways of presenting the same ideas and local variations. As the language changes, machine learning algorithms can adjust and modify their models, thus keeping them relevant and useful in identifying a linguistic ambiguities in modern requirements.

6. Integration into Software Development Workflow: When the machine learning algorithms are incorporated into the software development life cycle, anaphoric ambiguity detection process is automated through automation. This integration

streamlines the needs-analysis phase, minimizing the opportunities for misunderstandings and mistakes at a very early point in development.

7. Handling Large Data Volumes: The emergence of modern software projects brings a significant amount of the textual data in the form of requirements, user responses and documentation to these projects. This means that anaphoric references in complex data, like textual datasets, could be detected with reliability by machine learning algorithms.

8. Continuous Improvement: Yet, real use and feedback could change machine learning models. This continuous improvement ensures that the models adapt to language habits and user demands over time, which provides long-term accuracy and applicability.

Summarily, the employment of machine learning algorithms in NLP translates into more precise, efficient, and effective anaphoric ambiguity detection in natural language requirements. In addition, these algorithms allow for the implementation of elaborate language, context, and reference contacts that not only lead to better communication quality but also reduce misinterpretations and affect software production positively.

1.4 Motivation

In the world of software development, the translations of human intent into practical code relies heavily on natural language requirements. These requirements act as a crucial links throughout software projects, enabling communications from design to development and verification stages. However, the inherent complexity of language introduces ambiguities, causing delays, misinterpretations, and design flaws, leading to cost overruns. Anaphoric ambiguity, a subtype dependent on context, exemplifies the linguistic challenges faced in software development. As software systems increasingly centre around human interactions, the precision of requirements becomes paramount. Ambiguities in requirements can misguide individuals, introduce design errors, and impede software development.

1.5 Problem Statement

Anaphoric ambiguities occur when pronouns or phrases refer to earlier mentioned items, leading to different interpretations. For example, in the requirement "The system must show the results once they are ready," it is unclear what 'they' refers to without proper context. These ambiguities pose significant challenges for software development teams, as misunderstandings early on can lead to major issues later. Resolving anaphoric ambiguities requires careful consideration of context, domain knowledge, and user perspectives.

However, an equally critical challenge is the complexity and volume of features needed to accurately detect and resolve these ambiguities. The abundance of features can lead to overlapping or less informative data, increasing computational costs and reducing model accuracy. The difficulty lies in identifying the most relevant features while eliminating redundant and noisy ones, which is essential to enhance model generalization and performance.

This research addresses the problem of complex feature sets and reduced accuracy by employing feature reduction techniques. By integrating wrapper methods and correlation analysis, the study aims to refine the feature set, ensuring the selected features significantly contribute to the model's predictive capabilities. This approach not only simplifies the feature space but also improves the accuracy and effectiveness of anaphoric ambiguity resolution in natural language requirements.

1.6 Research Scope

Focusing on the ambiguous anaphoric detection area as one of the important spheres pertaining to software development, this thesis explores anaphor identification within the requirements of natural language. Anaphora is the recourse to any pronoun, noun, or phrase pointing to some antecedent element in the text that can be established only by virtue of context cues. The goal of this mission is to delineate, develop, and evaluate approaches and technologies that will

identify anaphoric ambiguities and therefore enhance quality requirements while preventing future problems.

The opportunity of this research includes numerous important dimensions:

1. Identification Techniques: The study investigates several modes of automatically identifying situations where anaphoric ambiguities are found in natural language specifications. This revolves around exploring syntactic and semantic analyses, machine learning models, as well as other computational linguistic approaches for the proper referring to the ambiguity detection.

2. Contextual Understanding: This study regards a segment of contextual knowledge that plays a role in anaphoric ambiguity. It talks about the ways in which various contextual cues, such as domain knowledge, the preceding sentences, and user intention, can be used to correct the ambiguous resolution of anaphoric references.

3. Dataset and Evaluation: A crucial part of the study is selecting candidate natural language requirements set with anaphoric ambiguities through curation. The research will provide criteria and methodologies for evaluating the performance of the proposed detection approaches.

4. Impact on Software Quality: The paper points towards the impact that anaphoric ambiguity settlement in natural language demand could have on bettering subsequent stages of the software build procedure. This includes better design, less rework, and an overall higher quality of software.

5. Practical Implementations: This research will investigate implementations of anaphoric ambiguity detection in realistic software development settings. This includes studying how automation tools can be applied in the requirements elicitation and validation processes.

6. Limitations: The study, which attempts to achieve total coverage, recognises anaphoric ambiguity resolution as part of the challenge of natural language understanding. As such, it may not explain all possible complexities and issues of language interpretation.

7. Comparative Analysis: The research could include comparing the proposed anaphoric ambiguity detection techniques with other approaches that are already available. This analysis will help us understand how effective and efficient the outlined methodologies are and what limitations they have.

8. Future Directions: As the discipline advances, techniques that can be applied across domains and among various languages will also emerge through research, with problems associated with language use changes identified.

In short, this thesis contributes to the study of anaphoric ambiguity detection within the framework of linguistic requirements by discussing several methods, evaluating their results performance-wise, and talking about practicism. Thus, this reasearch aims to work on the above-mentioned challenge as one of its biggest aspects and strive to achieve accuracy, precision, and unambiguity in the process of designing a software system, answering the following research questions.

RQ-1: How can the accuracy of anaphoric ambiguity detection in Software Requirement Specifications (SRS) be enhanced?

The methodology for addressing RQ-1 involves a far-reachings literature review to identify existing methods of anaphoric ambiguity detection in Software Requirement Specifications (SRS). This will be followed by experimental reasearch incorporating experiments or case studies to evaluate and propose enhancements to the accuracy of current detection methods. The process includes data analysis, potentially leveraging machine learning or natural language processing techniques, to validate & quantify the improvements in anaphoric ambiguity detection within the contexts of SRS.

RQ-2: Which machine learning algorithms performs better in anaphoric ambiguity detection among commonly used algorithms?

To address this research question, we evaluate the performance of several widely-used machine learning algorithms in detecting anaphoric ambiguities within software requirements. This involves comparing algorithms such as Naïve Bayes, Decision Table, AdaBoost, Logistic Regression, k-Nearest Neighbors, SVM, and Feed-forward Neural Networks. The evaluation focuses on metrics like precision, recall,

and F-measure to determine which algorithm offers the highest accuracy and reliability in identifying and resolving anaphoric references. This comparison will provide insights into the most effective techniques for enhancing the clarity and precision of software requirement specifications.

RQ-3: Can the feature set be reduced without effecting the performance of machine learning algorithms?

This research question investigates whether it is possible to streamline the feature set used in machine learning algorithms for anaphoric ambiguity detection without compromising their performance. By applying feature reduction techniques such as wrapper methods and correlation analysis, we aim to identify and eliminate redundant or less informative features. The goal is to maintain, or even enhance, the accuracy, precision, and recall of the algorithms while reducing computational complexity and mitigating the risk of overfitting. This exploration seeks to optimize the balance between model simplicity and predictive power.

Chapter 2

Literature Review

In the context of the vast picture that human communication presents, language is nothing more than a network of interdependent words and thoughts. We have a lot of linguistic tools that we use in verbal and written communication to present our meaning. These are all relevant features, but what stands out among them is the one that concerns pronouns—linguistic routes that connect referents and hold our speech together.

Anaphoric ambiguity is at issue when a pronoun or expression refers to a word that can be the potential antecedent. The possibility of misunderstandings, misrepresentations, and poor communication is also due to this level of ambiguity. Anahporic ambiguity resolution is the key component for any task using NLP technology, like machine translation, opinion detection, or a question-answering system.

Intensive activity is organised nowadays in the fields of linguistics, computational linguistics, and artificial intelligence due to anaphoric ambiguity detection, which has recently gained much importance within these sciences. This highly evasive issue has received significant interest among scholars and researchers alike, directed at creating robust algorithms and models that have the ability to untangle the intricate maze of pronoun references.

Thus, we start our literature review and discuss the latest research in the area of resolving anaphoric ambiguity using state-of-the-art techniques. Based on the

literature review of published studies, we attempt to clarify what innovations have been made and what difficulties have been faced while also providing insights into potential directions that follow-up studies could take in this very dynamic area.

Understanding Anaphoric Ambiguity: Anaphoric Ambiguity is the term of this section, where its methods, types, and linguistic realisation will be discussed. We will also discuss the need for ambiguity resolution that is associated with natural language understanding and processing.

Historical Perspectives: In this very section, we will focus on the evolution of anaphoric ambiguity detection, in which major studies and concepts that have inspired contemporary research will be addressed.

Approaches to Anaphoric Ambiguity Resolution: This section of the paper will look at several possible methods on how to handle anaphoric ambiguity, ranging from rule-based solutions to cutting-edge ML and DL models, to name a few.

Evaluation Metrics and Datasets: The researchers evaluate the effectiveness of anaphoric ambiguity detection systems by using relevant evaluation metrics and corpus datasets. In this section, this paper will focus on the common benchmarks used and their limitations.

Challenges and Future Directions: As any growing field does, anaphoric ambiguity detection comes with many challenges. We will determine these barriers and make some guesses about how future research could extend beyond the current limits.

In this literature review, we aim to clarify the concepts of anaphoric ambiguity detection and its importance for natural language processing. By untangling the strands of speech, we strive to come up with more precise and economical communication mediums that fill the gulfs between mankind and machinery.

2.1 Existing Methodologies

In 2021, a review was done by Apurwa Yadav and co-workers [12], which included an intensive analysis of the critical role of NLP in AI development, as evidenced

by some prestigious AI assistants such as SIRI, Natasha, and Watson. It emphasises the diverse applications of NLP beyond AI assistants, especially in terms of its effectiveness in solving RE disambiguation. The paper explains the importance of clear requirement documentation in avoiding software misconceptions. The methodology of the study critically analyses and evaluates various disambiguation tools, thus revealing that, despite some showing promise, ambiguities still persist. The paper addresses the balance of formality on the one hand and user friendliness on the other, always mindful that approaches based in NLP are a constantly developing art. The paper considers techniques involving CNL, style guides, knowledge-based approaches, and Transfer Learning approaches, focusing on the lack of universal analysis and evaluation in previous research. The paper separates automated tools from semi-automated tools and identifies promising avenues that favour machine learning and knowledge-based approaches to improve reliability. Although ambiguity remains a permanent threat, the paper ends on an optimistic note, identifying tools that have promising results that actually could take the formal language place. To this end, the paper skillfully analyses the diverse landscape of NLP-based disambiguation in RE, highlighting the need for further innovation in this dynamic domain.

However, the review is not without its imperfections, as it attempts to shed light on ambiguity resolution in natural language processing.

However, a weakness can be the absence of depth in methodological reporting. Considering that the topic is saturated with various natural language processing approaches and ambiguity resolution techniques, this article might not provide a detailed overview of all these techniques. Consequently, more technically knowledgeable readers may find the descriptions too general.

In addition, the paper is likely to provide an insufficient generalisation of the suitability and effectiveness of the reviewed methodologies in real-life situations. While it shows several methods, some practical examples or case studies could be required so that we know how these methods work in different environments. Summarizing, this research article may be viewed as a first glance at the ambiguity resolution methods, but its limitations in depth and the adequacy of practical

use analysis must be considered when referring to its contents from an angle of comprehensive understanding of that field.

Berry's [13] article discussed an important aspect of natural language processing, focusing on the problem of ambiguity in requirements documents. As the use of natural language in communication and documentation increases, ambiguity becomes a daunting obstacle in several fields, such as computing. Berry's work is of great importance in regards to this topic, as he focuses on the prevalence and implications of ambiguity within requirements documents. This paper highlights the potential risks of using ambiguous language, as it can result in mistranslations, misinterpretations, and ultimately poor software development. Berry indeed analyses different types of ambiguity that may occur in requirements, starting from the lexical to the structural level. The author uses case studies and shows practical cases to reveal the ambiguities that occur in real life and describe how they affect development. Berry's article not only gives an insight into the persistent nature of ambiguity but also presents ideas on how to neutralise it. Therefore, the relevance of context, structured documentation, and collaborative approaches becomes crucial in addressing ambiguity. In the field of software development, which becomes more and more sophisticated and based on precise information exchange, practitioners, researchers, and educators will greatly benefit from the insights presented in this article. Berry's study can be integrated into the methodology section, which already includes other studies, to provide a broader understanding of how ambiguity in natural language requirements documents affects them. This research can be considered a reference point from which ambiguity in the software development process is identified and addressed, which leads to an improvement in the overall quality of the software and project success.

Yang et al. [14] offer a detailed survey of the problems created by anaphora ambiquity in natural language specifications. The authors discuss one of the quite complicated topics linked to how pronouns and other anaphoric expressions may lead to ambiguities and uncertainty in requirement specifications that can delay software development. The study then attempts to apply a structured approach

by analysing numerous instances of anaphoric ambiguity and proposing methods for dealing with these issues.

The research makes an invaluable contribution to requirements engineering by sounding the depths of noun phrases in natural language. The authors' empirical analysis, along with their suggested approaches to disambiguation, provides useful insights for practitioners in software development striving towards a more definite requirement specification. This paper is required reading for scholars and practitioners who are interested in objectively judging language use to better understand the potential impact of linguistic considerations that may affect how software requirements are read or implemented, thereby improving productivity during implementation. In one of their surveys, Kiyavitskaya et al. [15] investigated the key areas associated with ambiguity detection and measurement. The proposed study employs a systemic approach to evaluate known methodologies for revealing the features of vagueness that are predetermined by prevailing practices for detecting and defining them accurately. By conducting a literature review and analysing various methods, several authors lay the necessary foundation for understanding the challenges encompassed inambiguity management.

By highlighting the importance of clear specifications for the achievement of effective software development, this work serves a great deal in requirements engineering. The study serves as a vital tool for scholars and professionals seeking precision and clarity in their mentioned requirement specifications. This paper uses an analysis of current methodologies to uncover the shortcomings and subsequently explores more advanced instruments or practices that can improve ambiguity perception in software development processes.

A Multi-Solution Study by Ezzini et al. [16] provides a detailed look into anaphoric ambiguity resolution in the complex world of various complicated means. The authors give a detailed description of these methodologies, highlighting their non-simplistic nature and shortcomings.

They focus their work on new methods of ambiguity and rejection. As opposed to Ezzini et al., approaches inspired by natural language processing and machine

learning using the power of sophisticated algorithms to untangle anaphoric references within requirements are in the spotlight. However, these approaches have their disadvantages. These solutions under consideration here bear a computational penalty that may serve as an obstacle to real-time deployment. More importantly, the authors bring up the potential danger of those complex methods producing false positives in terms of misidentifying some references as ambiguous and requiring meaning clarification that is unnecessary.

The work by Ezzini et al. highlights the significance of anaphoric ambiguity resolution's trade-offs between complexity and accuracy. The study contributes to the methodologies by carefully demarcating the disadvantages and complexities of these elaborate techniques, in addition to informing future researchers on how they may achieve an effective middle ground between practicality and clarity enhancement.

Shah and Jinwala [7] review the methodologies used to address ambiguity in natural language software requirements. The authors do an extensive analysis of the approaches, showing their different nature and complexity.

The research refers to linguistic approaches that focus on the syntactical and semantic structure of requirements in order to discover possible ambiguities. It also ventures into the world of formal methods that use mathematical approaches to precisely define requirements. In addition, the survey deals with automation techniques such as machine learnings and natural language processing, describing how technology can help address ambiguities.

The work of Shah and Jinwala [7] is remarkable for its detailed categorization and assessment of every methodology. Offering the strengths and concerns of these diverse approaches, the article endows researchers and practitioners with a wide arsenal that can be used to get rid of inconsistencies in specifications. This review is a helpful reference, the analysis of which helped to shed light on the problems emerging in the process of ambiguity resolution and find more effective solutions for software engineering.

Despite the fact that the survey provides quite a comprehensive overview of ambiguity resolution methodologies, there are several limitations. However, the potential limitation is that there may be no innovation in this field for quite some time. Based on the evolutionary characteristics of natural language processing and software engineering, the narration might not be representative of the latest achievements and innovative techniques for ambiguity resolution. This could lead to an insufficient characterization of the modern level of art.

Furthermore, the author who applies this methodology to his questionnaire may not do a comparative analysis of this system. While the article lumps and generalises some of the approaches, more detailed insight into their advantages and disadvantages may be provided as well. If such practices were completely compared, the readers could have an insight into what the best suitable method for a particular situation should be.

Lastly, while the "Resolving Ambiguities in Natural Language Software Requirements" survey is a good point of departure for subjecting ambiguity resolution to analysis and use, its results should be considered with limitations in mind.

Subha and Palaniswami [8] make one of the significant improvements to requirement specifications. There are diverse approaches for the authors to address readability and the quality of natural language requirements.

During the process, one of the most important parts is quality factor estimation, in which there are a range of qualitative and quantitative measures for measuring conciseness and precision that are used to measure. Through this approach, the authors state that they are planning to pinpoint any potential areas of ambiguity and hence make refinements. Secondly, the study considers approaches to text summarization in order to reduce the requirements and shorten them without losing their essential information. These methodologies are also known to enhance the quality of general requirement specifications.

The research by Subha and Palaniswami [8] reveals a proactive approach towards enhancing requirement clarity using qualitative appraisal along with text summarization to save from questioning doubts about ambiguities. Through the use

of a various approaches, authors offer a wide overview of improving requirement specifications and their contribution to the evolution of software engineering practices. This paper serves as a resource for scholars and professionals looking to enhance methodologies for clarifying requirements and quality improvement while also practicing clear requirement formulation.

TAPHSIR: Towards Anaphoric Ambiguity Detection and ReSolution in Requirements by Ezzini et al. [17] outlines a novel approach and framework for reducing anaphoric ambiguity in requirements. However, the authors outlined an uncommon method that takes into account a full circle of anaphoric reference detection and resolution.

At the core of their process is NL processing, using modern techniques for the identification of anaphoric ambiguity cases to automate this process. This intricate linguistic analysis to make use of incorporates the utilisation of contextual clues and semantic relationships. Besides, the authors propose a multi-stage resolution procedure that includes candidate targeting, ranking, and conclusive selection. This uncertainty resolution procedure offers a structured approach to uncertainty removal while increasing the level of requirement specification clarity. While the suggested methodology possesses substantial advantages for addressing anaphoric ambiguity in requirements, there are numerous disadvantages that should be addressed.

Firstly, the practical implementation of the proposed approach might prove to be rather problematic due to its very sophisticated nature. The long and complex process of language analysis and multi-level settling might lead to computational expense, which will negatively affect the software development environments when fast application is needed. Moreover, relying on advanced methods of natural language processing could have specialised training data sets and high computational power restrictions that can limit their use in some cases.

Secondly, a more detailed assessment of the performance of that given methodology can be conducted. The article does not offer a comprehensive comparison between new and existing methods; therefore, it is impossible to judge the efficacy of TAPHSIR against other solutions. A more comprehensive assessment, perhaps

based on a broader population with varied real-world considerations, would lead to better insights into the capabilities and weaknesses of the methodology.

Finally, even though TAPHSIR is a promising technique that can help to cope with anaphoric ambiguities, the issues of complexity and practical implementation, in addition to insufficient complete evaluation, should be accepted as methodological drawbacks.

TABLE 2.1: Literature review summary

Year Published	**Models Used**	**Keypoints**	**Contribution**
2022 [11]	RoBERTa, BERT, DistilBERT, ALBERT, ELECTRA, MiniLM, BART, T5	- Use of various language models for QA. - Application of IR-based Retriever models. - Application of MRC-based Reader models.	- Introduction of ReQAssis, an AI-based QA system. - Description of the ReQAssis approach in six steps. - Detailed explanation of each step in the ReQAssis approach.
2021 [6]	Supervised machine learning approach	- Different feature sets - Sampling techniques - Hyperparameters and algorithms	Successful development of an approach for assessing two quality characteristics of natural language requirements
2015 [7]	Natural Language Processing (NLP), Machine Learning (ML)	- Survey of state-of-the-art approaches for resolving ambiguities in natural language requirements. - Review and discussion of different approaches and tools for ambiguity resolution in software requirements specification. - Classification of tools into automated & semi-automated categories. - Emphasis on the use of domain-specific language, restricted syntax/grammar, and sentence patterns	The paper surveys and analyzes prevalent approaches for resolving ambiguities in natural language software requirements, presents a state-of-the-art survey of currently available tools, and aims to identify metrics for a comparative evaluation. It also points out open research issues to stimulate further developments in the field.

Table 2.1 continued from previous page

Year Published	Models Used	Keypoints	Contribution
		the readability and understandability of requirements. - Recognition that tools using knowledge-based, ontology, and machine learning approaches are efficient in identifying semantic ambiguities.	
2019 [8]	FORML	Ambiguity as an instrument to expose more subtle features in requirements analysis. Future work on exploring and exploiting the beneficial relation between ambiguity and the elicitation of tacit knowledge. Acknowledgment to Stefania Gnesi and financial support from the Centre for Human-Centred Technology Design Research at UTS and the National Science Foundation under grant CCF-1718377.	Comprehensive exploration of ambiguity phenomena in requirements specifications. Thorough examination of the relationship between ambiguity, abstraction, and absence of information. Exploration of a subtle variation of ambiguity, referred to as vagueness. Characterization of different levels at which ambiguity can be manifested in requirements specification documents. Theoretical framework for studying different forms of ambiguity.
2011 [14]	Noun-phrase coreference resolution engine	Automated identification of anaphoric ambiguity	Developing a system to detect potentially nocuous ambiguity in requirements documents, specifically focusing on anaphoric ambiguity. The system incorporates an antecedent classifier trained using heuristics

Table 2.1 continued from previous page

Year Published	Models Used	Keypoints	Contribution
2008 [13]	-	NL Processing	NL's inherent ambiguity and trade-offs with formal languages. Significance of addressing ambiguity in requirements engineering (RE). Approaches: improving writing, using restricted NL, employing tools. Taxonomy of ambiguity types in NL requirements. Emphasis on 100% recall in ambiguity detection tools. Conclusion: inevitability of ambiguity, need for early resolution.
2008 [15]	Shell scripts, NL processing (NLP) system, WordNe	Ambiguity identification in NL RSs, two-step approach, T1 and T2 tools	Proposes a two-step approach to identifying ambiguities in NL RSs. Requirements-identification experiments, Settled requirements issues, Counter indications, Tradeoffs and considerations in tool development
2013 [3]	Open Text Summarizer, Decision tree-based quality evaluator	Ambiguity analysis, Text summarization, Quality evaluation, Co-reference resolution	Proposed framework for automatic ambiguity analysis, summarization, and quality evaluation in software requirements documents. Performance evaluation comparing the proposed system with ARKref NP coreference system.

Table 2.1 continued from previous page

Year Published	Models Used	Keypoints	Contribution
2022 [17]	Ensemble ML classifier, SpanBERT	45 language features, Hybrid solution with ML-based ambiguity detection and BERT-based anaphora resolution	TAPHSIR facilitates anaphoric ambiguity detection and resolution in requirements through a hybrid solution, contributing to the improvement of precision and consistency in natural-language requirements. The tool is publicly available on Zenodo.

2.2 Research Gap

So far, the domain of automatic anaphoric ambiguity identification in natural language requirements has been little studied. Despite the significant progress in demanding natural language processing and requirement analysis, there is a glaring space left for research that follows effective anaphors within a requirements specifications.

A lot of the current research on ambiguity detection is focused on finding general ambiguities or other types of larger linguistic ambiguities. The complex meanings that anaphoric insertions bring to the table is not given much attention. Systems that must take into account the intricate interplay of the pronouns, noun phrases, and surrounding cues that cause redundancy handle the requirements for anaphoric ambiguity detection in RSs automatically. It is important to conduct thorough research on requirement specifications so that the creation of an accurate and efficient set of algorithms for the recognition and resolution of these ambiguities can be achieved.

In addition, the lack of standardised datasets and benchmarking techniques for anaphoric ambiguity with reference to this context makes it difficult to improve

research and development in such an area. However, without clearly defined evaluation criteria and metrics for comparison, it is challenging to compare the performances of any proposed approaches in a uniform manner.

As such, the state-of-the-art research gap reveals the alarming lack of dedicated methods and tools to detect, classify, and deflect natural language requirements and anaphoric ambiguities. Closing this knowledge gap could potentially greatly increase the quality and specificity of requirement specifications, which would also improve software development process management and results.

Firstly, there is a widespread weakness found in the literature on automated anaphoric ambiguity detection, as it concerns data that is used for testing. This is due to the anaphoric references in software requirements not being fully captured by several historical studies relying on relatively small datasets. Such restricted data may, in turn, affect the applicability of the solutions proposed for solving larger and more complicated necessities sets commonly found during practical software development cases.

Secondly, another study deficit stems from the intricacy of feature sets used in previous research. However, studies that focus on complex feature sets may cause overfitting or high computational costs. The balance between the size of the dataset, the complexity of features, and the performances given by models needs to be explored. By investigating the effect of dataset size on algorithmic robustness and optimal feature complexity level, one could get an idea of what is necessary for designing a more scalable and efficient solution for anaphoric ambiguity detection.

By focusing on such aspects, not only will a better understanding of anaphoric ambiguity be brought about, but the use of automated detection systems beyond limited sets of real-world scenarios can be guaranteed to advance this field.

2.3 Conclusion

Lastly, the state-of-the-art techniques existing in natural language requirements that focus on the area of anaphoric ambiguity detection have revealed a noticeable

gap in the literature. Although impressive advancements have been made towards natural language processing and requirement analysis, the rate of this problem's saturation—anaphoric ambiguity—is relatively low. The inadequate research and tools that were developed to help address this other challenging task expose an aspect of specification and clarity of requirements that is neglected yet pertinent.

This means that major research is absent in this area, which has direct economic implications for software development processes. Anaphoric ambiguity is responsible for many faulty interpretations of meanings and misalignments, and sometimes it arises that the software did not achieve its purpose. With no well-established benchmarking techniques and standardised datasets, this only makes the task more challenging. It hampers the development of various ideas and their objective assessment.

With the increasing complexity of software systems and more strongly bureaucratized organisational logics where precise requirement specifications are at a premium, anaphoric ambiguities must be addressed head-on. The revealed gap in research points to the urgent problem of establishing dedicated work aimed at creating efficient algorithms, tools, and assessment criteria to confront the issue directly. By addressing such an omission, the software engineering community can move towards stricter and thus more precise requirement specifications, resulting in an overall increase in the quality of outcomes achieved through application development.

Chapter 3

Proposed Methodology

3.1 Introduction

This chapter presents the delicate folds of our proposed methodology, a specifically developed strategic framework that aims at perfecting anaphoric ambiguity detection accuracy and efficiency in software requirements. In Fig. 3.1, we can see that our approach marches through a sequence of intertwined steps as described below: from data gathering to the evaluation of the model. As a source of light that is an anchor in the world of decoding anaphoric ambiguities, the suggested methodology appears to give structure and insights into the intricate problem of distinguishing ambiguous and unambiguous software requirements. Our methodology begins with a systematic inquiry, reflecting aspects of the discussion on the holistic approach to solving problems of language processing and ambiguity detection. This chapter details every step from the development of a comprehensive dataset to the refining of machine learning models, demonstrating our determination to enrich the functional features of anaphoric ambiguity detection tools in software engineering.

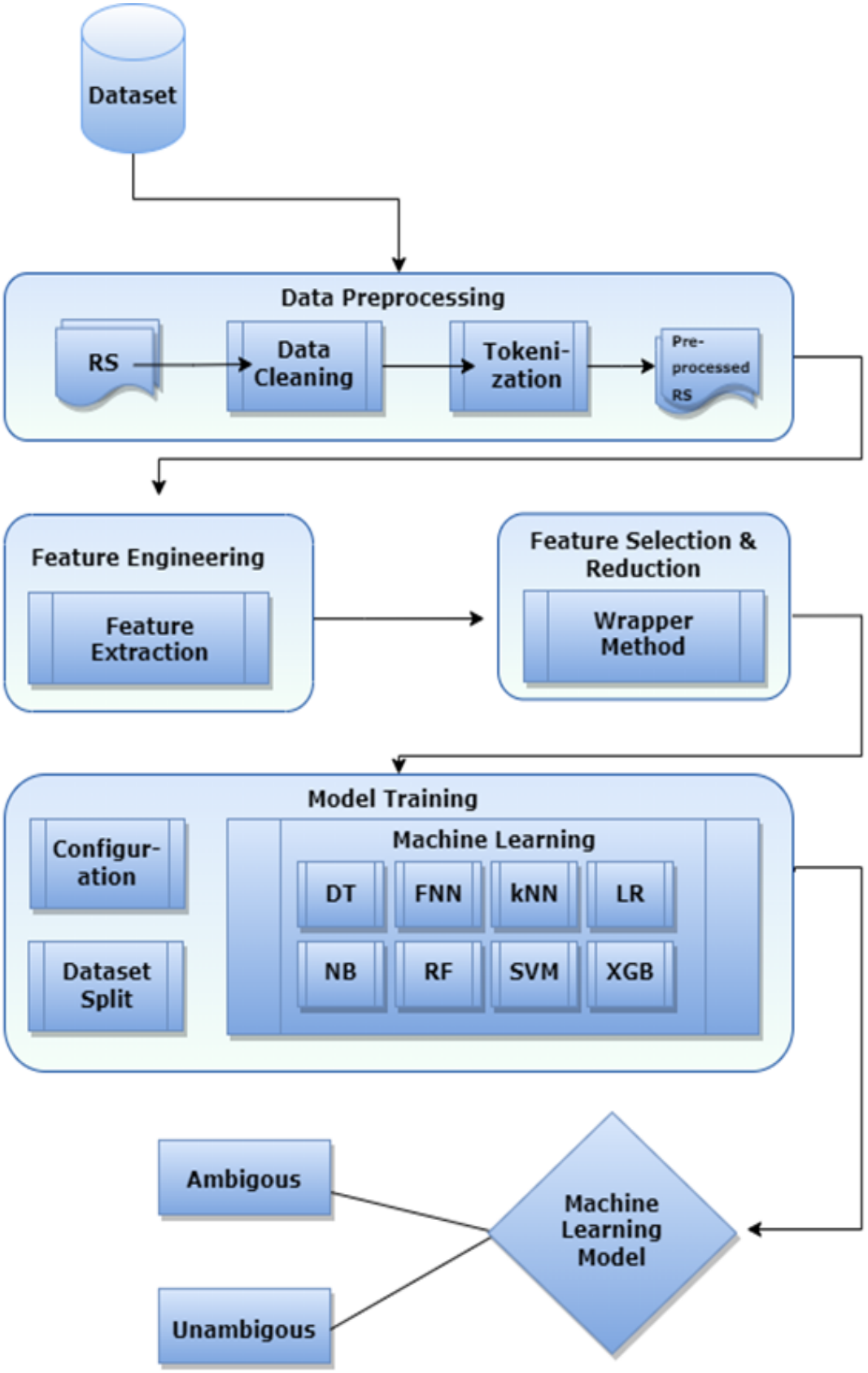

FIGURE 3.1: Proposed Methodology

3.2 Dataset Description

Data collection process plays a vital role in the proposed methodology, and DAMIR (DAtaset for MIning Requirements), which is a publicly available dataset, was carefully curated for anaphoric datifying detection. This data set is born out of rich yet vast repository of 1251 labelled software industrial needs with 738 pronouns occurrences, resulting to creation of a whopping total of 6667 contexts. The preprocessing involved in the generation of DAMIR is a series of processes that are designed to prepare the input for several options for ambiguity resolution. Catalogue of each individual document included in this dataset which employs a fine-grained annotation that classifies them either as 'Ambiguous' or "Unambiguous". This meticulous categorization is vital for subsequent supervised learning processes, since it allows the model to detect patterns associated with ambiguity insets of software requirements.

Table 3.1: Statistical Information about Dataset

Dataset (Ambiguous/ Unambiguous)	**Pronoun Occurrence**	**Class**
Train	275	ambiguous
	316	unambiguous
	591	both
Test	68	ambiguous
	79	unambiguous
	147	both
Total	343	ambiguous
	395	unambiguous
	738	both

Ambiguous Requirement: Harrows inconsistent references or interpretations, meaning there are diverse interpretations of the passage.

Unambiguous Requirement: Unambiguous, without several possible interpretations, ensuring a definite and unwavering understanding without misunderstanding.

3.3 Data Preprocessing

After the data set description, our methodology seamlessly proceeds to the pivot stage of preprocessing that occurs. Its transformative stage is aimed at converting text data into machine-learning-ready formats. A series of critical operations are carried out to achieve synchronisation and harmony at every corner of the database. In the initial stages of preprocessing, we introduce this Natural Language Processing (NLP) pipeline that subjects' data into Software Requirements Specification (SRS) through text. This pipeline can perform as a kind of macroresource planned to systematically handle and analyse the text inside the SRS:

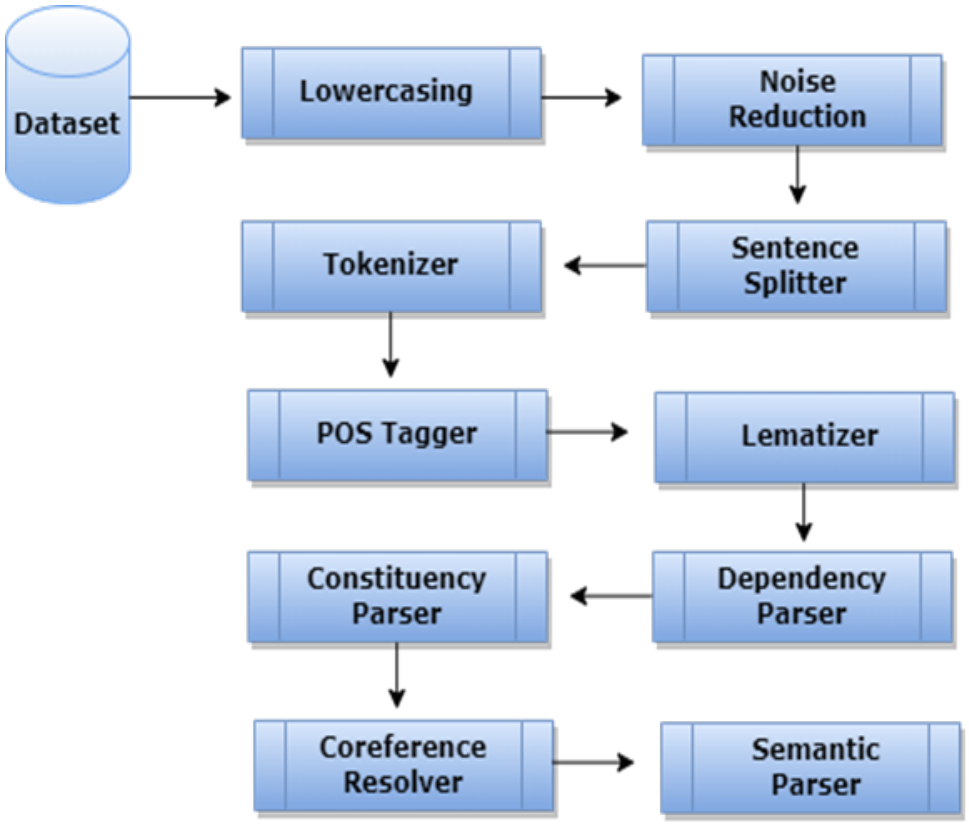

FIGURE 3.2: Data Preprocessing

3.3.1 Lowercasing for Uniformity

The provision of standardisation is the first stage whereby lowercase letters are used. This method transforms all texts to lowercase so that case variations do not affect the analysis. This is the elimination of disparities that would occur in capitalization procedures due to the standardisation of case paradigms for this data set.

3.3.2 Cleaning Processes for Noise Reduction

Finally, a number of cleaning operations are implemented to improve the dataset by eliminating irrelevant features with noise elements. These processes include the removal of any symbols, punctuation marks, and non-alphanumeric characters that are not important. In this way, removing these parameters ensures that the following analyses will not have their operations perturbed, and the model will be trained in peace.

3.3.3 Tokenizer

The tokenizer submodule functions well in tokenizing the text data by separating it into tokens or units of language that are applicable for further processing. It tokenizes the text appropriately. The most important segment is the first point of subsequent analysis and feature extraction. Each token can be used as a primitive to allow downstream processing because it is used for feature discrimination and model learning.

3.3.4 Sentence Splitter

The grammatical segmentation of our sentence splitter allows for a finer analysis of sentences, each containing its own context and meaning.

3.3.5 Part-of-Speech (POS) Tagger

We also used an advanced POS tagger that tags each word with grammatical categories (nouns, verbs, and adjectives), allowing for more in-depth linguistic analysis and greater text understanding.

3.3.6 Lemmatizer

The lemmatizer module is adept at finding and normalising words to their base or root form, leading to more standardisation and less redundancy for better analysis.

3.3.7 Constituency Parser

In our pipeline, we employ a robust and precise constituency parser to analyze the syntactic structure of sentences. A constituency parser dissects sentences to uncover their grammatical composition, breaking them down into constituent parts such as phrases and sub-phrases. This parsing reveals the hierarchical arrangement of words and phrases, showing how they combine to form larger syntactic units. For instance, in the sentence "The quick brown fox jumps over the lazy dog," the parser identifies noun phrases like "The quick brown fox" and verb phrases like "jumps over the lazy dog," illustrating the relationships between these components. This detailed syntactic information is crucial for understanding the underlying structure of sentences, which aids in tasks like anaphoric ambiguity detection by providing clear insights into the grammatical roles and relationships of different elements within the text. By integrating this powerful tool into our pipeline, we enhance our ability to accurately interpret and process complex linguistic data.

3.3.8 Dependency Parser

A dependency parses words in sentences and creates covered dependencies that represent a tree of dependent words, necessary for sentence structural analysis.

3.3.9 Coreference Resolver

One of the components that our pipeline involves is an advanced coreference resolver, which can identify and resolve references to the same entity between sentences, leading to improved uniformity and interpretation.

3.3.10 Semantic Parser

Last but not least, our semantic parser empirically covers aspects such as the meaning and intent attached to text segments for a deeper understanding of the semantics present in them.

3.3.11 Pronoun Identification and Context Determination

The first stage of the pipeline is this process of identifying pronouns in the SRS aimed at personal and possessive pronouns. Each pronoun identified is associated with its context, a requirement. The pipeline mechanically analyses the SRS content with personal and possessive pronouns recognised using linguistic patterns and grammatical rules. It precisely marks words inside the text that fall under personal or possessive pronouns.

It establishes a connection between the pronoun and its surrounding content once the pronoun is detected. This connection defines the 'context,' which refers to the requirements within which the pronoun appears. For each detected pronoun, the context includes the current requirement it belongs to and, if applicable, previous requirements where the pronoun has appeared.

3.3.12 Generating Candidate Antecedents

Additionally, a list of candidate antecedents is produced for every pronoun using preceding NPs from the context. The NPs are automatically detected using the NP parser module underlining the NLP pipeline.

3.4 Refined Dataset Creation

In the end, the careful selection of candidate predecessors results in a meticulously constructed dataset.

DAMIR is one of the most important parts because it is a well-organized and polished set of data that has been expanded on for positive anaphoric ambiguity in the case of software requirements.

In addition, through candidate antecedents inclusion, the dataset allows for careful curation that provides a sturdy basis to train the model. It helps us understand and judge the vague reasons in software requirements better, which is necessary for the anaphoric ambiguity detection model to be able to find this kind of complexity.

Utilising such a meticulous approach, this method enables the development of a detailed dataset, which allows for an accurate interpretation of ambiguity references in software requirements. DAMIR's construction represents a fundamental breakthrough, the landmark building on which subsequent analyses and model training rely upon a reliable and highly polished base dataset.

3.5 Feature Extraction and Engineering

In this segment, the focus shifts to the extraction of essential features from the dataset, a crucial step in enhancing the model's predictive capabilities. Python programming language is employed to conduct feature extraction, resulting in the identification and computation of a comprehensive set of 45 features presented in table 3.2 along with names and types. These features encapsulate diverse aspects of the dataset, providing a rich representation of the underlying patterns.

The Python code executed for feature extraction is designed to capture pertinent information from the dataset, contributing to a more robust understanding of the data's intricacies. These 45 features serve as the foundation for subsequent analyses and model development.

Following the feature extraction phase, attention is directed towards feature selection. It involves a meticulous process of identifying the most relevant features among the initially extracted set. Feature selection aims to enhance model efficiency and interpretability by retaining only those features that significantly contribute to the predictive task while discarding redundant or less informative ones.

This two-step approach, starting with feature extraction and followed by feature selection, ensures that the model is equipped with a discriminative set of features, thereby improving its overall performance and interpretability. The subsequent sections delve into the details of the feature selection process and its impact on the model's efficacy.

In this stage, we examine a 45-feature set that a reference tool used to review it. It aims at finding traits that greatly impact the performance of the tool in anaphors' ambiguity detection. Tools such as feature significance scores and correlation analysis are used to assess how relevant and important each feature is. This critical analysis streamlines the feature set to retain only those features that are informative, which are retained for further training. We have employed two measures of the importance of features, which were compared between them.

TABLE 3.2: Feature set [18]

ID	**Name**	**Type**
LF1	distance	Numeric
LF2	genderAgreement	Boolean
LF3	numberAgreement	Boolean
LF4	isAntecedentAnimate	Boolean
LF5	isAntecedentDefinite	Boolean
LF6	isAntecedentPrepositional	Boolean
LF7	isAntecedentSubject	Boolean
LF8	isAntecedentDirectObject	Boolean
LF9	isAntecedentIndirectObject	Boolean
LF10	sameSyntacticRole	Boolean
LF11	repetitivePatterns	Boolean
LF12	sameDependency	Boolean

Table 3.2 continued from previous page

ID	Name	Type
LF17	nextPOS	Enumerated
LF18	existsClause	Boolean
LF19	relativePosition	Numeric
LF20	positionOfFirstNoun	Numeric
LF21	sentencePosition	Numeric
LF22	headLemma	String
LF23	headPOS	Enumerated
LF24	precedingPOS	Enumerated
LF25	followingPOS	Enumerated
LF26	numPrecedingNPs	Numeric
LF27	numPrecedingNPsContext	Numeric
LF28	numFollowingNPs	Numeric
LF29	numNPs	Numeric
LF30	numNPsInContext	Numeric
LF31	numFollowingAdj	Numeric
LF32	nextAdj	String
LF33	precedingVerb	String
LF34	nextVerb	String
LF35	numFollowingComplementizer	Numeric
LF36	numTokensBeforeComplementizer	Numeric
LF37	isAdjBeforeNP	Boolean
LF38	numTokensBeforeInf	Numeric
LF39	numTokensBeforePrep	Numeric
LF40	numTokensBeforeGerund	Numeric
LF41	existsComplementizerNP	Boolean
LF42	existsPrecedingPrep	Boolean
LF43	precedingToken	String
LF44	nextToken	String
LF45	numPunctuations	Numeric

3.5.1 Feature Importance Scores

One way to measure the relevance of each characteristic is via the computation of feature importance scores. Each of the measures below quantifies how much any given element contributes to the predictive power of the model. Some of the most frequently used techniques are those applied in models that use decision trees, like random forests or gradient-boosted trees. It is possible to compute the feature importance scores from the average impurity decrease values that occur in multiple decision trees within an ensemble. Mathematically, the feature importance score (I) for a particular feature can be calculated as:

$$I_i = \frac{\sum_i \ impurity\ decrease_{ij}}{Number\ of\ Trees}$$

Where I_i is the importance score for feature i, impurity decrease$_{ij}$ is the impurity decrease for feature i in tree j, and the sum is taken over all trees in the ensemble.

3.5.2 Correlation Analysis

Therefore, correlation analysis is another tool used to assess the relationship between features. High linkage between features means that there is redundancy, and hence, one of them can be discarded in order to reduce dimensionality without compromising accuracy. Here, Pearson's correlation coefficient (r) is widely used, and it is given by:

$$r = \frac{\sum(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum(X_i - \overline{X})^2 \sum(Y_i - \overline{Y})^2}}$$

where X_i and Y_i are the values of the two features for the i-th observation, and $\overline{X}$ and $\overline{Y}$ their respective means.

This highly diverse analysis that combines feature importance scores and correlation analysis is among the most crucial parts of defining the significance of all

these features. High-importance scores, or those features that have unique information to be included in the model, are retained. At the same time, it is possible to discard features with high intercorrelation in order to reduce the number of features. The final product, therefore, is an optimised feature set with all the crucial features for subsequent model training. Through such a carefully selected feature extraction process, the model contributes to the robustness and interpretability of anaphoric ambiguity detection.

3.6 Feature Reduction

In the pursuit of a more concise feature set, the study employs feature reduction techniques to identify and eliminate overlapping or less informative features. The goal is to enhance model generalization and mitigate the impact of noisy elements that could adversely affect the model's performance.

Among the array of techniques utilized for feature reduction, special emphasis is placed on wrapper methods. Wrapper methods play a pivotal role in feature reduction by incorporating the model's predictive performance into the evaluation process. Unlike standalone algorithms, wrapper methods assess features' relevance based on their impact on the model's accuracy. This approach ensures that the selected features contribute significantly to the model's predictive capabilities.

The procedure for feature reduction involves the following steps:

Initial Feature Set Compilation: Begin with an extensive set of features derived from the data.

Wrapper Method Application: Implement wrapper methods to evaluate each feature subset. This involves training the model on different subsets of features and using performance metrics (such as accuracy, precision, and recall) to assess each subset's effectiveness. The steps include:

- Feature Subset Selection: Generate various subsets of the initial feature set.

- Model Training and Evaluation: Train the model on each subset and evaluate its performance.
- Subset Comparison: Compare the performance metrics to identify the most promising feature subsets.

Correlation Analysis: Conduct correlation analysis to identify and remove highly correlated features, which do not provide additional information and may introduce redundancy. This involves:

- Correlation Matrix Construction: Create a correlation matrix to quantify the relationships between features.
- Threshold Setting: Define a threshold for acceptable correlation levels (e.g., 0.8).
- Feature Elimination: Remove one of the features from each pair that exceeds the correlation threshold.

Iterative Refinement: Iteratively refine the feature set by repeating the wrapper method application and correlation analysis until the optimal set of features is identified.

By strategically reducing complexity in the sequential feature space, computational costs are minimized, and the risk of model overfitting is mitigated. The subsequent sections delve into the specifics of the wrapper method, highlighting its effectiveness in optimizing the feature set and ultimately contributing to the refinement of the model's predictive performance.

3.7 Reevaluate and Train the Model for Anaphoric Ambiguity Detection

Our strategic approach centers around the continual reassessment and enhancement of the anaphoric ambiguity detection model to ensure its adeptness in capturing intricate patterns within software requirements.

A key facet of this enhancement lies in the augmentation of the feature set, a process meticulously executed to bolster the model's discernment capabilities.

To achieve this, the dataset undergoes a rigorous division into training and validation subsets. This segregation facilitates the utilization of various machine learning algorithms, such as decision trees, random forests, support vector machines, and deep learning models, in training the model. Each algorithm is employed to harness distinct aspects of the dataset, contributing to a more holistic understanding of anaphoric ambiguity patterns.

Furthermore, the training process incorporates a robust 10-fold cross-validation methodology. This technique involves partitioning the dataset into ten subsets, with the model trained and validated iteratively on different combinations of these subsets.

The cross-validation process ensures that the model's performance is consistently evaluated across diverse data instances, promoting its adaptability to generalize learnings effectively to previously unseen data.

This phase culminates in a refined model, enriched through iterative training, validation, and cross-validation cycles, poised to exhibit enhanced performance in the detection of anaphoric ambiguities within software requirements.

3.8 Machine Learning Models

In our context, we primarily focus on supervised machine learning techniques. These algorithms require labeled training data to learn patterns and make predictions. Supervised learning involves two primary tasks: classification and regression. Classification handles categorical targets, while regression deals with continuous numerical targets. Conversely, unsupervised learning algorithms operate without labels, while semi-supervised learning methods handle partially labeled data, combining supervised and unsupervised techniques.

3.8.1 Decision Tree (DT)

Decision Trees are predictive models that use a tree-like graph to represent decisions. Each internal node means a test on a feature, each branch signifies an result of the test, and each leaf node holds a class label. These models are easy to understand and comprehend, making them valuable for visualizing complex decisions. However, they can easily overfit on the training data, especially if the tree grows too deep or lacks regularization.

3.8.2 Feed-forward Neural Network (FNN)

Feed-forward Neural Networks consist of interconnected nodes arranged in layers (input, hidden, and output). They use forward propagation to transmit data from input to output through the layers. FNNs are adept at capturing complex nonlinear relationships within data. The input layer size matches the data features, and the output layer represents the classes for prediction. Despite their ability to handle complex data, training deep networks might require substantial data and computational resources.

3.8.3 k-Nearest Neighbors (kNN)

The k-Nearest Neighbors algorithm categorizes new data points based on their similarity to existing data in the training set. It measures distances (e.g., Euclidean, Manhattan) to seek the 'k' nearest neighbors and assigns the majority class among those neighbors to the new data point. kNN's simplicity lies in its ease of implementation, particularly for multiclass classification. However, its computational cost increases with larger datasets as it requires comparing the test data with every training data point.

3.8.4 Logistic Regression (LR)

Logistic Regression is a statistical model utilized for binary classification problems. It forecasts the possibility that an instance belongs to a certain class using the logistic function (sigmoid). LR evaluates the connection between the dependent binary variable and one or more independent variables. Despite its name, logistic regression is a classification algorithm, not a regression technique, providing probabilistic values between 0 and 1.

3.8.5 Naïve Bayes (NB)

Naïve Bayes is a simple yet powerful probabilistic classifier based on Bayes' theorem. It assumes independence among features and calculates the probability of each class given the features. NB is efficient, particularly for high-dimensional datasets, and provides quick predictions. However, its strong assumption of feature independence might not hold true in some real-world scenarios.

3.8.6 Random Forest (RF)

Random Forest is an ensemble learning method that constructs multiple decision trees and amalgamates their outputs through majority voting. By reducing overfitting and variance, RF improves prediction accuracy. Each tree is trained on a different subset of the data and features, ensuring diversity. RF can handle large datasets with high dimensionality and is less prone to overfitting compared to individual decision trees.

3.8.7 Support Vector Machine (SVM)

Support Vector Machines create a hyperplane in an n-dimensional space to separate classes. SVM aims to maximize the margin between classes, enhancing generalization. It works well in spaces with many dimensions and provides different kernel functions to take care of nonlinear relationships existing in data. The

SVMs are memory-efficient since only a small part of the training data at some points during optimisation is used for decision function computation.

3.8.8 AdaBoost (ADA) and XGBoost (XGB)

AdaBoost is an ensemble boosting method in which weak learners are repeatedly combined to form a powerful classifier. It updates weights on mislabeled sample points to enhance predictions. XGBoost is an optimized algorithm for GBDT that fine-tunes the models by emphasizing on incorrectly classified instances in each iteration. However, these boosting algorithms provide good results for weaker learners since they increase the predictability and adaptability of complex datasets. The supervised ML algorithms discussed provide a range of ways to handle anaphoric ambiguity detection that come with their pros and cons. The decision trees are interpretable, and while discussing the neural networks, we have complexity management. Algorithm selection is guided by data, computer resources available to solve the model, and whether interpretability or predictive accuracy are desired.

3.9 Model Evaluation

Evaluation of the anaphoric ambiguity detection model requires in-depth analysis and accurate evaluation criteria. Evaluation measures, including accuracy, precision, recall, and F1-score, yield a detailed picture of how well the model operates. These measures are crucial performance indicators against which the practical functioning of the new instrument can be measured. To make the model perform effectively on heterogeneous datasets, cross-validation is taken into consideration, making sure that its efficiency does not rely solely on the training set alone. This is a critical stage in the tool application verification and evidence that it can be operated successfully in actual situations to resolve ambiguities apparent within the requirements software.

3.9.1 Precision

However, given that precision is a suitable measure of evaluation, it measures the accuracy with which positive examples are predicted by the model. It provides the average value of the true positive rate as the proportion of actual positive predictions to what was predicted overall by the model (TP/ (T P + FP)). In the case of anaphoric ambiguity detection, precision is a measure that indicates how many correct cases out of all instances detected as ambiguous. The low false alarm property suggests that the model can discriminate between ambiguous cases without classifying clear-cut ones as inaccurate. Precision briefly offers a picture of the degree to which the model can identify ambiguities in references, which is a valuable parameter if false positives may have grave consequences.

$$Precision = \frac{True\ Postives}{True\ Postives + False\ Postives}$$

3.9.2 Recall

One of the key KPIs, recall, measures how fully the model can capture all positive samples constituting its training set. It determines the value of true positives divided by the sum of all positive cases: (TP/ (TP + FN). In terms of anaphoric ambiguity detection, recall represents the model's ability to identify all actual cases of ambiguity in a data set. The higher recall score means a lower number of false negatives, which later corresponds to eliminating ambiguous cases and, therefore, preventing omissions in the form of oversight errors. The case where rejection is typically conventional when recalling a genuine positive exclusion, which would induce higher costs than a misidentified incorrect negative one.

$$Recall = \frac{True\ Postives}{True\ Postives + False\ Negatives}$$

3.9.3 Fβ-score

The Fβ score, which represents the harmonic mean value of these variables when their weighted sum in use is considered to apply, is also called a measure of precision and recall. The Fβ-score is just one value following the formula (1 + β^2) * (p*r)/($\pi\beta$2^** p + r), where β helps in balancing precision and recall.

It is this score that allows for some compromise between precision and recall when, depending on the given task specificities, it proves necessary. A larger Fβ score serves as a trade-off, wherein both evaluation signals call for equal levels of significance or where one feature is regarded to be more significant than the other.

$$F_\beta = (1 + \beta^2) \times \frac{Precision \ \times \ Recall}{(\beta^2 \ \times \ Precsion) + Recall}$$

Observing the selection of the most precise outcomes can be witnessed in commonly used methods such as requirements analysis tools. In the context of ambiguity analysis, this implies. The measure of performance introduced for anaphora resolution is the success rate, i.e., the fraction of correctly resolved pronoun tokens in cases with no ambiguity.

This evaluation incorporates two modes: whole or full matching, in which the antecedent needs to coincide with a ground truth entirely, and partial or proportional matching, in which interests overlap between an antecedent and ground truth. The possibility of partial evaluation is built on finesse evaluations, which are attractive to the cases under manual review discussed above where partial correctness can provide valuable information about text spans that require a closer look.

This holistic assessment method also shows that the model is able to be robust in differentiated data sets, where attention to detail for real-world applications envisions this tool as functional.

3.10 Interpret Model Results

One essential component of methodology is carried out in a full model forecasting analysis. The rationale for dissecting the results is to dig out features that have been mostly influential in decision-making. This post-modelling analysis gives important insights into requirements classification as ambiguous vs. unambiguous. The interpretability of models is important not only for verification of reliable decisions made by a model but also for understanding the underlying causes behind predictions.

Finally, the presented solution for the detection of anphoric ambiguity in software requirement documents is a more general one. Thus, our method attempts to achieve data-driven techniques, feature engineering, and state-of-the-art ML-mechanisms, sharing the focus on improving accuracy of ambiguity detection instruments. Taking part in the process of software development, we want to make our contribution efficient and successful by offering tools that can overcome ambiguity while translating such necessities.

3.11 Tools & Programming Languages

This thesis used a collection of tools and programming languages that were crucial in terms of managing, processing, and logging data. These instruments are selected meticulously to ensure efficiency, speedy execution, and accuracy at every stage of the competition.

Jupyter Notebook: Being an interactive computational environment, it enables the researcher to run code and present visualisations and information that would document the analysis pathway.

Microsoft Visio: It is used in the design of detailed diagrams, helping to visually depict system architectures and flowcharts as well as large organisational structures.

Microsoft Excel: It played the role of a thoroughness-enabling tool that enabled organised information flow and preliminary research.

Python: As a language of both computer programming and machine learning, Python is one of the most popular languages, with numerous libraries, extensive flexibility, and a strong community.

Scikit-learn (Sklearn): a library of machine learning algorithms that are used to implement different models and techniques.

PyCharm: Used as an Integrated Development Environment (IDE) for Python development that has a simple interface and debug capabilities.

LaTeX: LaTeX had been instrumental in the writing of the thesis; LaTeX brought unprecedented precision and formatting mastery, controlling a high-quality end product.

Weka: Used for resampling methods and initial experimental setups, providing opportunities for comparative analyses and preliminary descriptions about data distribution and model behaviour.

These collected tools underpin the research design as they provided a powerful preprocessing, modelling, and analysis process along with effective documentation that supported the accuracy and success of this work.

Chapter 4

Experiments, Results and Evaluation

4.1 Introduction

Following the detailed discipline of methodology that was presented in Chapter 3, this chapter describes the experiments and results presented, as well as a strict evaluation process for the anaphoric anomaly detection model. With the strategic framework illustrated in Figure 3.1 as a guide, our approach aims at improving the precision and specificity of ambiguity detection in requirements for software.

4.2 Experimental Setup

However, before moving into the results section, some background information about the experimental setting should be provided. The DAMIR dataset, curated through a rigorous data collection process, forms the foundation for our experiments. Comprising 1251 labeled software industrial requirements, the dataset undergoes meticulous preprocessing to prepare it for machine learning analysis. The experiments detailed in Chapter 4 were conducted with the following revised hardware and software specifications:

Hardware Specification

Processor: Intel(R) Core (TM) i5-3230M CPU @ 2.60GHz 2.60 GHz

RAM: 8 GB

Disk Space: 256GB SSD

Operating System and Software Specification

OS: Windows 10 Pro

Programming Language: Python (utilizing NLP)

Python Version: 3.8

Framework: PyCharm, Jupyter Notebook

Additional Software: Microsoft Excel 365, Weka 3.8

These updated specifications reflect the utilization of PyCharm as the integrated development environment (IDE) for Python programming, Python version 3.8, and Microsoft Excel 365 for result analysis. Additionally, the operating system was upgraded to Windows 10 Pro, and Weka 3.8 was employed for resampling techniques and initial experimental setups.

4.3 Implementation of NLP Pipeline

The Natural Language Processing (NLP) pipeline, a critical component of our methodology, is implemented to parse the textual content of Software Requirements Specifications (SRS). With eight intricately crafted modules, including a tokenizer, sentence splitter, part-of-speech tagger, lemmatizer, constituency parser, dependency parser, coreference resolver, and semantic parser, this pipeline ensures a systematic and thorough analysis of the linguistic nuances within the requirements.

Table 4.1: NLP Pipeline step-by-step Implementation

Steps	**Output**
Input Sentence	"The Contractor will provide disposal and demilitarization plans to DND. DND will provide oversight on the Contractor disposal management and on its execution of disposal actions."

Steps	**Output**
Tokenizer	Tokens generated: "The", "Contractor", "will", "provide", "disposal", "and", "demilitarization", "plans", "to", "DND", ".", "DND", "will", "provide", "oversight", "on", "the", "Contractor", "disposal", "management", "and", "on", "its", "execution", "of", "disposal", "actions", "."
Sentence Splitter	Sentences identified: "The Contractor will provide disposal and demilitarization plans to DND." "DND will provide oversight on the Contractor disposal management and on its execution of disposal actions."
POS Tagger	POS tagging applied: "The/DT Contractor/NN will/MD provide/VB disposal/NN and/CC demilitarization/NN plans/NNS to/TO DND/NNP ./. DND/NNP will/MD provide/VB oversight/NN on/IN the/DT Contractor/NN disposal/NN management/NN and/CC on/IN its/PRP$ execution/NN of/IN disposal/NN actions/NNS ./. "
Lemmatizer	Lemmatized text: "The", "Contractor", "will", "provide", "disposal", "and", "demilitarization", "plan", "to", "DND", ".", "DND", "will", "provide", "oversight", "on", "the", "Contractor", "disposal", "management", "and", "on", "it", "execution", "of", "disposal", "action", "."
Constituency Parser	Parsed sentence structure: (S (NP (DT The) (NN Contractor)) (VP (MD will) (VP (VB provide) (NP (NN disposal) (CC and) (NN demilitarization) (NNS plans)) (PP (TO to) (NP (NNP DND)))) (. .))) (S (NP (NNP DND)) (VP (MD will) (VP (VB provide) (NP (NN oversight) (PP (IN on) (NP (DT the) (NN Contractor) (NN disposal) (NN management) (CC and) (IN on) (NP (PRP$ its) (NN execution) (IN of) (NN disposal) (NNS actions)))))) (. .)))
Dependency Parser	Dependency relationships: "provide" ->"Contractor" (nsubj) "provide" ->"plans" (dobj) "provide" ->"DND" (prep_to) "provide" ->"DND" (nsubj) "provide" ->"oversight" (dobj) "oversight" ->"will" (aux) "oversight" ->"management" (prep_on) "management" ->"Contractor" (det) "management" ->"disposal" (compound) "management" ->"execution" (conj_and) "execution" ->"its" (poss) "execution" ->"actions" (nmod_of)
Coreference Resolver	Resolving references: The coreference resolver identifies references such as "it" referring to "DND" and "its" referring to "DND's."
Semantic Parser	Extracting semantic meaning: This step involves understanding the deeper meaning and intent behind the text segments, which can include entity extraction, relationship identification, and context comprehension.

4.4 Pronoun Identification and Context Determination

Central to the pipeline is the pronoun identification and context determination stage. Leveraging linguistic patterns and grammatical rules, this phase systematically identifies personal and possessive pronouns within the SRS. The related context is set, i.e., the nearest and previous needs are defined that serve as a background for further analysis.

4.5 Candidate Antecedent Generation & Dataset Refinement

A crucial step follows in generating candidate antecedents for each pronoun, incorporating preceding Noun Phrases (NPs) within the context. The resulting meticulously curated dataset, DAMIR, stands as a refined foundation for anaphoric ambiguity analysis within software requirements.

Such an integral step is to produce candidates for antecedents of each pronoun within the context, consisting in mentioning previous Noun Phrases (NPs). The obtained finely hand-crafted dataset, DAMIR, is a basis that has been improved in the analysis of anaphoric ambiguity in software requirements

Table 4.2: Candidate Antecedent Generation

Software Requirement	**Pronoun**	**Candidate Antecedent**
The Contractor will provide	its	The Contractor
disposal and demilitarization plans to	its	disposal and demilitarization plans
DND. DND will provide oversight on the	its	DND
Contractor disposal management and	its	oversight
on its execution of disposal actions.	its	the Contractor disposal management

4.6 Initial Experimentation

In the first phase, where different machine learning algorithms will be used to test the ability of our anaphoric ambiguity detection model, we plan on using a diverse set. The processed dataset, which has been painstakingly preprocessed to achieve uniformity and consistency, is now ready for experimentation. Here, we use eight well-known machine learning algorithms that have proven their worth over the years and accompanied with shortcomings of their own. The chosen algorithms for this trial operation are Decision Tree (DT), Feed-forward Neural Network (FNN), k-Nearest Neighbors (kNN), Logistic Regression (LR), Naïve Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM). Every algorithm is a different method of learning patterns from the data, and we expect divergent performances depending on the properties of our anaphoric ambiguity dataset. The experimentation script follows a systematic method:

4.6.1 Data Splitting

The set is grouped into training and validation sets to assist the models in construction and evaluation. This procedure helps in guaranteeing that the models are trained on the presented data and tested against unseen instances to assess their generalization performance.

4.6.2 Model Training

Thus, for every algorithm, the training set is utilized in the purpose to train a model that can identify patterns connected with ambiguous and well-defined references in software requirements. Each of the model parameters is tuned to improve their performance during this training phase.

Table 4.3: ML Algorithms performance (Initial Experiment)

Sr	Algorithm	Precision	Recall	F-Measure	Time Taken (s)
1	Naïve Bayes	0.654	0.653	0.652	0.04

Sr	Algorithm	Precision	Recall	F-Measure	Time Taken (s)
2	Random Forest	0.797	0.797	0.797	2.73
3	Decision Table	0.793	0.793	0.793	2.35
4	ADA Boost	0.681	0.680	0.680	0.69
5	Logistic Regression	0.674	0.674	0.674	0.65
6	k-Nearest Neighbors	0.790	0.790	0.790	0.01
7	SVM	0.672	0.670	0.669	6.1
8	FNN	0.784	0.784	0.784	77.22

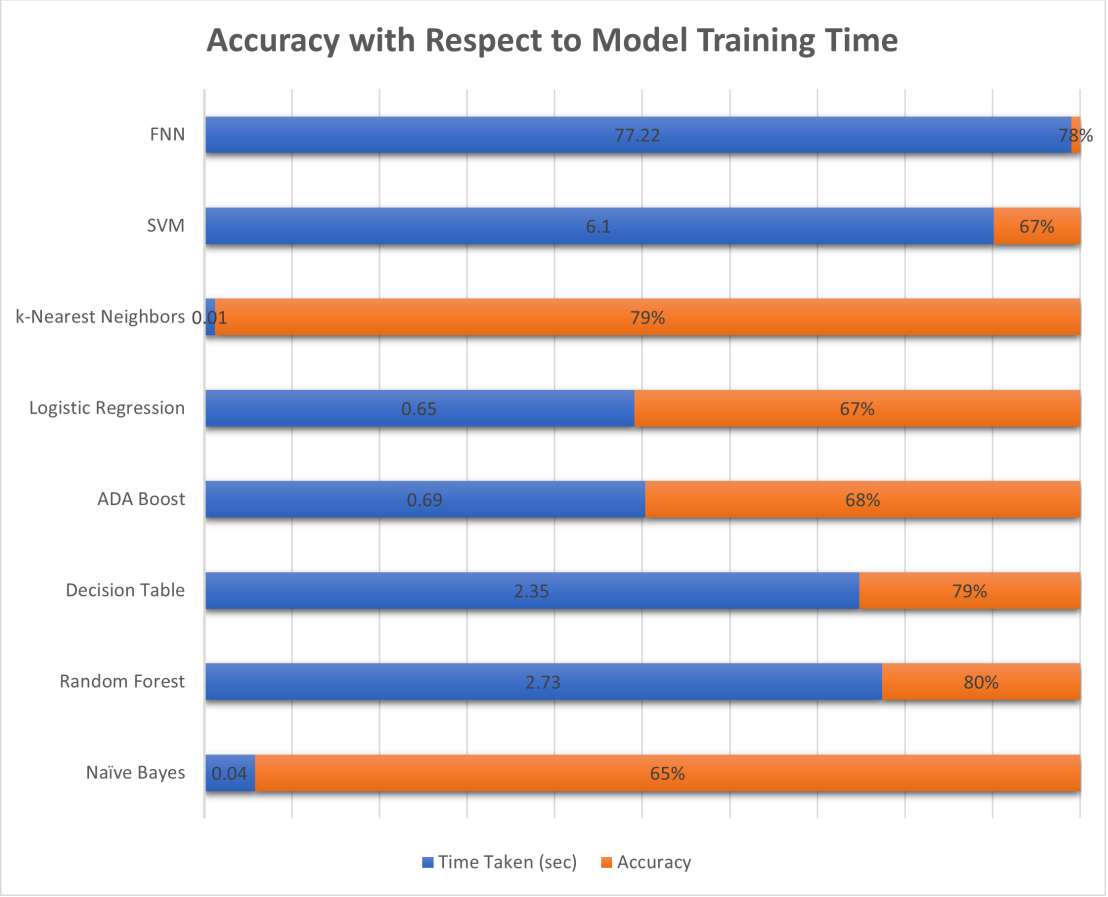

FIGURE 4.1: ML Algorithms accuracy with respect to time

4.6.3 Model Evaluation

The performance of the learned models is evaluated against the validation set by metrics sensitive to changes in data, such as precision, recall, and F-measure. These measures highlight the models' capabilities for identifying ambiguous instances correctly (precision), capturing all real ambiguous cases (recall), as well as

averaging two of the latter ones (F-measure). Offering a systematic analysis of the strengths and weaknesses of each algorithm to employ it in anaphoric ambiguity detection.

4.6.4 Results Analysis

The trade-off between precision and recall is studied in detail for every algorithm. The ability of each model to trade in precision for recall is important, enabling further decisions about the selection and reduction of feature sets. High precision, on the other hand, reduces false positives, and high recall is needed to lower false negatives. The F-measure works as a comprehensive measure that combines both elements.

Feature selection and engineering are employed to advance the performance of the model. To analyze and scrutinize a 45-feature set, we use approaches like feature importance scores and correlation analysis. This iterative procedure improves the feature set, as from this stage on, only informative features will support subsequent model training. All these supervised ML algorithms are used in the model's extensive training, which includes Decision Trees, Feed-forward Neural Networks, k-Nearest Neighbors, Logistic Regression, Naïve Bayes.
These algorithms are picked, among others, because of their individual advantages and disadvantages in adapting to various datasets. As additional steps, ensemble techniques such as voting and stacking are used to further refine and optimize the anaphoric ambiguity model. In this way, these techniques consolidate the results of several models to increase overall accuracy, generating a reliable and flexible model.

Table 4.4: ML Algorithms performance after reduced feature set– 30 features (Wrapper Method)

Sr	Algorithm	Precision	Recall	F-Measure	Time Taken
1	Naïve Bayes	0.657	0.657	0.657	0.04
2	Random Forest	0.898	0.898	0.898	2.2
3	Decision Table	0.913	0.913	0.913	1.47

Sr	Algorithm	Precision	Recall	F-Measure	Time Taken
4	ADA Boost	0.681	0.680	0.680	0.38
5	Logistic Regression	0.657	0.657	0.657	0.2
6	k-Nearest Neighbors	0.849	0.849	0.849	0.01
7	SVM	0.679	0.676	0.676	5.29
8	FNN	0.794	0.794	0.794	61.4

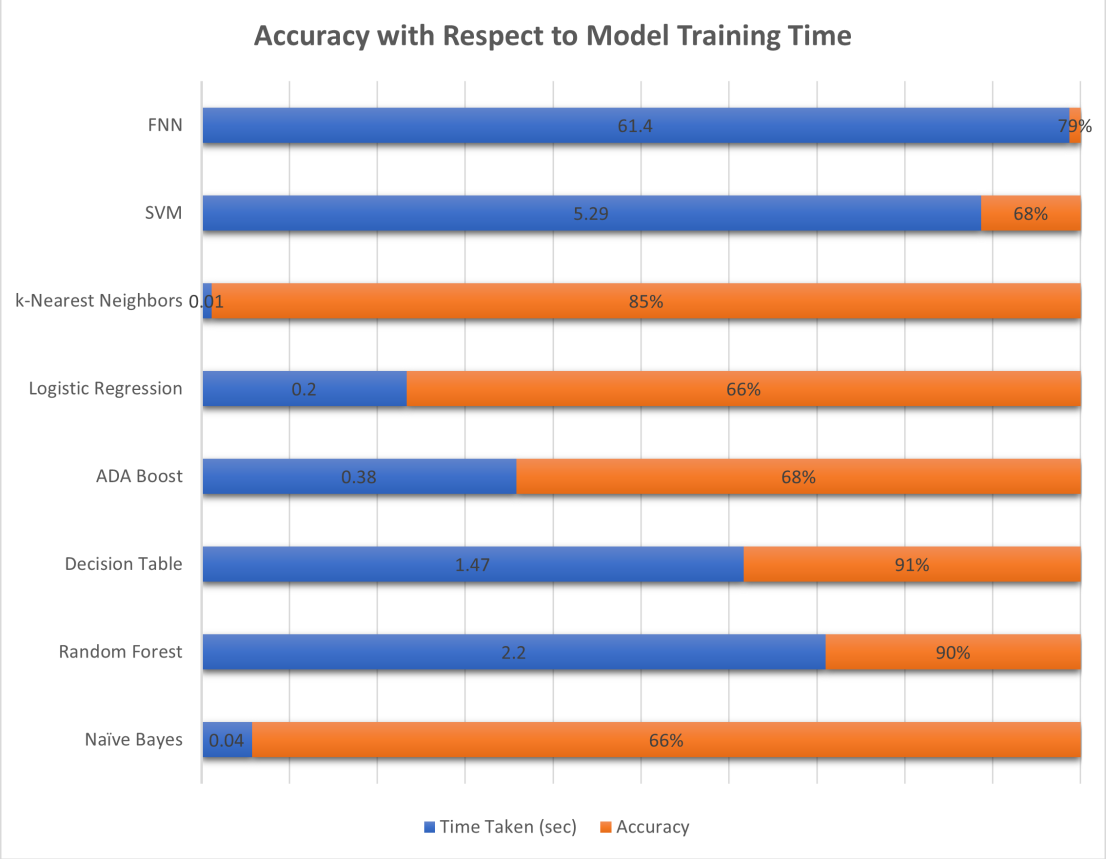

FIGURE 4.2: ML Algorithms accuracy with respect to time (reduced feature set)

TABLE 4.5: ML Algorithms performance after reduced feature set – 20 features (Wrapper Method)

Sr	Algorithm	Precision	Recall	F-Measure	Time Taken
1	Naïve Bayes	0.649	0.649	0.649	0.02
2	Random Forest	0.998	0.998	0.998	1.97
3	Decision Table	0.993	0.993	0.993	0.82
4	ADA Boost	0.681	0.680	0.680	0.19

Sr	Algorithm	Precision	Recall	F-Measure	Time Taken
5	Logistic Regression	0.658	0.658	0.658	0.1
6	k-Nearest Neighbors	0.995	0.995	0.995	0.01
7	SVM	0.673	0.672	0.671	3.18
8	FNN	0.925	0.925	0.925	16.97

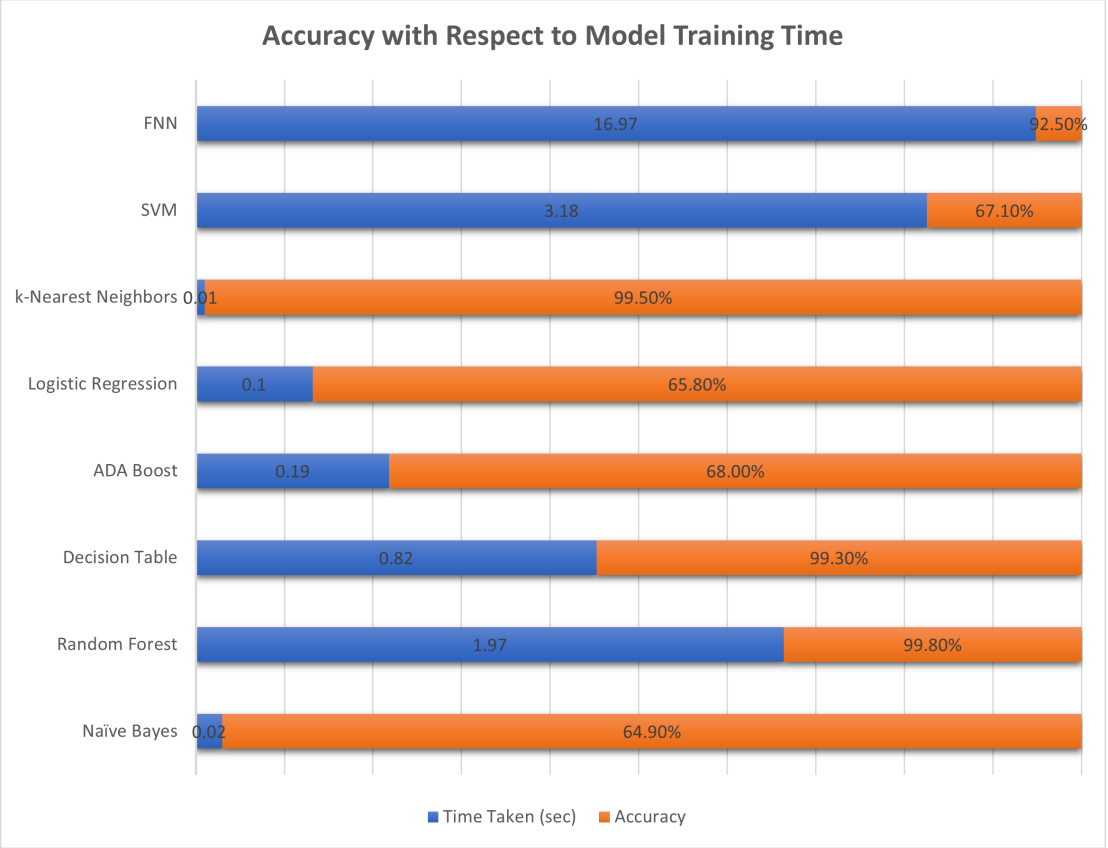

FIGURE 4.3: ML Algorithms accuracy with respect to time (reduced feature set)

Sophisticated measurement of anaphoric ambiguity detection models involves high-quality statistics such as accuracy, precision, recall, and Fβ-score. The model is cross-validated to validate the effectiveness in various datasets, particularly demonstrating its applicability and trustworthiness under real-world conditions.

This chapter provides an introduction to the experiments carried out, the results attained, and a thorough analysis of the anaphoric ambiguity detection model based on generalization expected in this chapter. The proposed methodology, which has a systematic design as well as incorporating algorithms from different

families of machine learning, is intended to support tools facilitating the professional interpretation of ambiguities in requirement statements, thereby helping develop software projects with high efficiency.

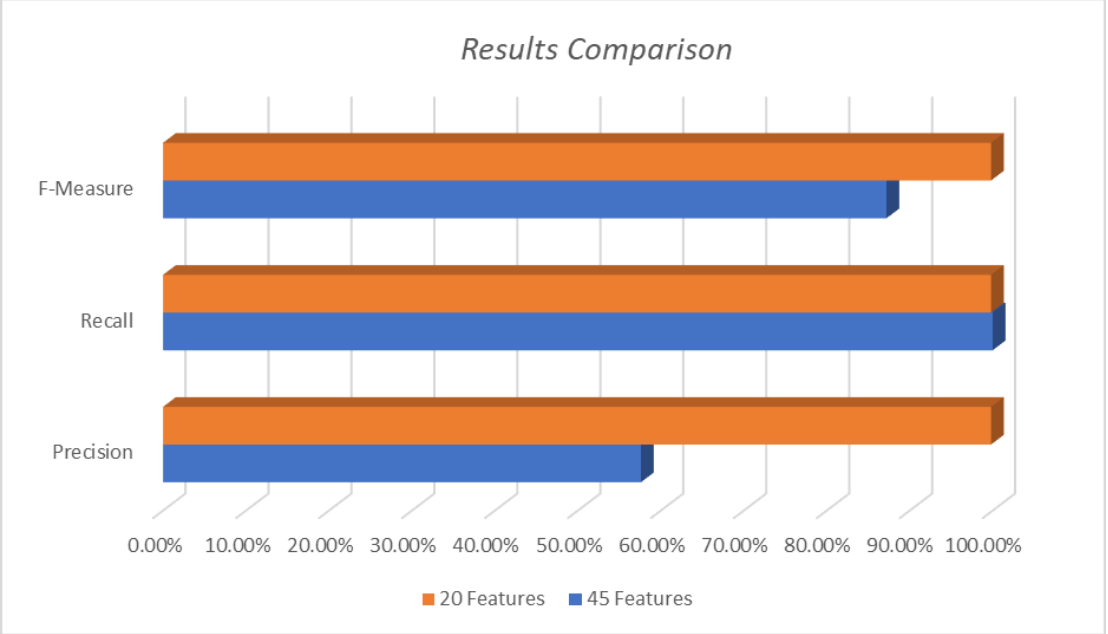

FIGURE 4.4: Results Comparison

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The writers to this research embarked on a travel that ended with building and assessment of an anaphoric ambiguity detection model for the software requirements. The precise methodology, described in Chapters 3 and 4, laid out the groundwork for a process of scientific experimentation and evaluation. In this article, you will summarize the major findings and conclusion yielded from this research.

The effectiveness of the NLP pipeline, an important component of our methodology, in parsing and identifying intricate linguistic details within software requirements was shown. The pipeline presented eight modules, deliberately constructed to guarantee a deep analysis from which pronouns identification and determination of the context would then follow.

The empirical stage, based on the DAMIR dataset, reflected the flexibility and adaptability of our model. Machine learning algorithms like Naïve Bayes, Decision Table, ADA Boost, Logistic Regression, k-Nearest Neighsours, SVM and Feed-forward Neural Network had the potential usage in detecting anaphoric ambiguity through our model.

The development of the model was further advanced through careful data feature selection and engineering. Through the example of using wrapper methods to

minimize the feature sets from 45 to 30, and then further down to 20 features, it was possible to prove that the precision, recall, and overall accuracy improved across several algorithms. In particular, the Random Forest algorithm worked well when data features were reduced to 20 with a precision of 0.998, recall of 0.998, and F-measure of 0.998 afterward.

Ensemble approaches, including voting and stacking included in the model, increase robustness. Through the output mergings conducted by dozens of algorithms, we were able to come up with an adaptive anaphoric ambiguity detection model that was more accurate in predictings and could be incorporated into different datasets.

5.2 Future Work

However, it should be noted that this study is only a part of the process related to studies dealing with anaphoric ambiguity detection; there are many fertile areas for research and development in this respect. Initially, the model can be significantly enhanced with respect to performance by exploring even more complex neural network architectures, such as transformer-based models like BERT or GPT, that have excellent capabilities of identifying complex linguistic dependencies present in natural language.

In this way, the use of linguistic features other than those used may lead to more advanced software requirements analysis when domain knowledge is added. Among other things, the fine-tuning model's ability to identify terms and contexts for specific industries could make it more suitable for domain software production tasks.

Furthermore, increasing the size and variability of the dataset by including other requirements for software could also make the model more generalized. A much larger data set ensures a greater variety of contacts, therefore enriching the model's potential for real-world exploitation.

Last but not least, anaphoric ambiguity detection bestows a valuable solution to the SRAA problem. However, that is why it becomes vitally important to make sure that requirements are interpreted correctly in an understandable way due to the fact that software development projects grow more complicated by the day. This model, which will feature a systematic approach and functional natural language processing for software engineering, will have content that is specifically tailored.

Bibliography

[1] R. R. Nelson *et al.*, "It project management: Lessons learned from project retrospectives 1999–2020," *Foundations and Trends® in Information Systems*, vol. 4, no. 4, pp. 275–381, 2021.

[2] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural language processing for requirements engineering: A systematic mapping study," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–41, 2021.

[3] R. Subha and S. Palaniswami, "Quality factor assessment and text summarization of unambiguous natural language requirements," in *International Conference on Advances in Computing, Communication and Control*, pp. 131–146, Springer, 2013.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] S. Abualhaija, C. Arora, A. Sleimi, and L. C. Briand, "Automated question answering for improved understanding of compliance requirements: A multi-document study," in *2022 IEEE 30th International Requirements Engineering Conference (RE)*, pp. 39–50, IEEE, 2022.

[6] P. Kummler, "Automated quality assessment of natural language requirements,"

[7] U. S. Shah and D. C. Jinwala, "Resolving ambiguities in natural language software requirements: a comprehensive survey," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 5, pp. 1–7, 2015.

[8] V. Gervasi, A. Ferrari, D. Zowghi, and P. Spoletini, "Ambiguity in requirements engineering: towards a unifying framework," *From Software Engineering to Formal Methods and Tools, and Back: Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday*, pp. 191–210, 2019.

[9] C. Manning and H. Schutze, *Foundations of statistical natural language processing*. MIT press, 1999.

[10] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[11] S. Ezzini, "Artificial intelligence-enabled automation for ambiguity handling and question answering in natural-language requirements," 2022.

[12] A. Yadav, A. Patel, and M. Shah, "A comprehensive review on resolving ambiguities in natural language processing," *AI Open*, vol. 2, pp. 85–92, 2021.

[13] D. M. Berry, "Ambiguity in natural language requirements documents," in *Monterey Workshop*, pp. 1–7, Springer, 2007.

[14] H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Analysing anaphoric ambiguity in natural language requirements," *Requirements engineering*, vol. 16, pp. 163–189, 2011.

[15] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requirements engineering*, vol. 13, pp. 207–239, 2008.

[16] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, "Automated handling of anaphoric ambiguity in requirements: a multi-solution study," in *Proceedings of the 44th International Conference on Software Engineering*, pp. 187–199, 2022.

[17] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, "Taphsir: towards anaphoric ambiguity detection and resolution in requirements," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1677–1681, 2022.

[18] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, "Automated handling of anaphoric ambiguity in requirements: a multi-solution study," in *Proceedings of the 44th International Conference on Software Engineering*, pp. 187–199, 2022.