Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.052218

ARTICLE

Classification and Comprehension of Software Requirements Using Ensemble Learning

Jalil Abbas[1,*], Arshad Ahmad[2,*], Syed Muqsit Shaheed[3], Rubia Fatima[4], Sajid Shah[5], Mohammad Elaffendi[5] and Gauhar Ali[5]

[1]School of Computer Science and Technology, Anhui University, Hefei, 230039, China

[2]School of Computing Sciences, Pak Austria Fachhochschule, Institute of Applied Sciences and Technology, Haripur, 22620, Pakistan

[3]Department of Computer Science and IT, University of Lahore, Lahore, 55150, Pakistan

[4]Department of Computer Science, Emerson University, Punjab, Multan, 60000, Pakistan

[5]EIAS (Emerging Intelligent Autonomous Systems) Data Science Lab, Prince Sultan University, Riyad, 12435, Saudi Arabia

*Corresponding Authors: Jalil Abbas. Email: jalil02085@stu.ahu.edu.cn; Arshad Ahmad. Email: yaarshad@gmail.com

Received: 26 March 2024 Accepted: 18 June 2024 Published: 15 August 2024

ABSTRACT

The software development process mostly depends on accurately identifying both essential and optional features. Initially, user needs are typically expressed in free-form language, requiring significant time and human resources to translate these into clear functional and non-functional requirements. To address this challenge, various machine learning (ML) methods have been explored to automate the understanding of these requirements, aiming to reduce time and human effort. However, existing techniques often struggle with complex instructions and large-scale projects. In our study, we introduce an innovative approach known as the Functional and Non-functional Requirements Classifier (FNRC). By combining the traditional random forest algorithm with the Accuracy Sliding Window (ASW) technique, we develop optimal sub-ensembles that surpass the initial classifier's accuracy while using fewer trees. Experimental results demonstrate that our FNRC methodology performs robustly across different datasets, achieving a balanced Precision of 75% on the PROMISE dataset and an impressive Recall of 85% on the CCHIT dataset. Both datasets consistently maintain an F-measure around 64%, highlighting FNRC's ability to effectively balance precision and recall in diverse scenarios. These findings contribute to more accurate and efficient software development processes, increasing the probability of achieving successful project outcomes.

KEYWORDS

Ensemble learning; machine learning; non-functional requirements; requirement engineering; accuracy sliding window

1 Introduction

Requirement engineering techniques, both traditional and advanced, play a vital role in organizing software requirements, aligning essential project needs with innovative strategies to address the complexities of modern software development [1]. In this context, understanding functional requirements (FRs) and non-functional requirements (NFRs) is fundamental to the software development process. FRs encompass all the essential features and services a software system must provide to its users, including user authentication processes, efficient data archiving systems, advanced search functionalities, and secure payment processing mechanisms. Conversely, NFRs define the software's inherent constraints and desired quality attributes, such as scalability, user-friendliness, robust security measures, performance efficiency, and accessibility [2].

A high-quality software system is built upon both FRs and NFRs, adapting to meet user demands while ensuring exceptional performance and efficiency. Accurately identifying user requirements from their informal expressions presents a significant challenge in the early stages of software development. This crucial step shapes the direction of the entire project, demanding considerable investment in terms of time and effort. Addressing communication and coordination challenges during requirements change management is essential for effective requirement engineering in software development [3]. Moreover, manually sifting through textual documents to uncover these requirements is impractical due to the associated time, cost, and potential for error [4]. Therefore, automatic requirement classification is essential for managing large volumes of textual documents. By accurately comprehending and interpreting user-expressed requirements into tangible FRs and conceptual NFRs, software solutions can be created that not only meet but exceed user expectations in terms of functionality and effectiveness. Extensive research has been conducted on NFRs, with various studies employing different methods and techniques to extract and organize NFRs from diverse types of documents.

Understanding user requirements expressed in natural language has long been a challenge that demands considerable human effort. To alleviate this workload and streamline the process, a wave of innovation has introduced several machine learning (ML) based methods [5,6]. By harnessing the potential of ML, developers can unlock new possibilities, paving the way for more efficient and user-centric software solutions. The use of ML techniques for extracting NFRs from software requirement documents or other textual sources has gained popularity due to their ability to process and analyze large datasets efficiently. Commonly used supervised methods include Random Forest, Support Vector Machines (SVM), and Naive Bayes. However, several limitations and challenges remain associated with this approach.

Random Forest algorithms, known for their versatility and robustness, are increasingly utilized for extracting FRs and NFRs from textual data. This ensemble learning technique, which constructs multiple decision trees (DTs) and combines their outputs for more accurate and stable predictions, is particularly favored for its effectiveness in handling diverse data types and its resistance to over-fitting [7]. However, when applied to the specific task of FR and NFR extraction, Random Forest algorithms encounter certain limitations. Their traditional strength in handling structured, numerical datasets does not always translate well to the processing of high-dimensional, sparse textual data typical in requirement documents. This can result in challenges in accurately distinguishing between different types of requirements.

This study delves into the detailed aspects of FRs and NFRs, exploring methods for identifying user needs and their impact on new product development. It emphasizes the significant role of FRs and NFRs in creating innovative and efficient software systems. To address the above-said limitations, we propose an enhanced version of the Random Forest algorithm, known as the Functional and Non-functional Requirements Classifier (FNRC). This novel approach incorporates an innovative Accuracy Sliding Window (ASW) mechanism, designed to prune inefficient DTs and identify an optimal sub-ensemble, thereby improving accuracy and reducing time complexity. By evaluating our proposed FNRC on a dataset of FRs and NFRs, we demonstrate its superiority over other state-of-the-art algorithms in terms of classification accuracy, time efficiency, and memory consumption.

The rest of the paper is structured as follows: Section 2 provides details of previously introduced methods for classifying FRs and NFRs. Section 3 describes the proposed methodology, followed by experiments and results in Section 4. Section 5 addresses threats to validity, and Section 6 concludes the paper along with suggestions for future work.

2 Related Work

Software requirement specifications (SRS) documents are typically written in natural language, containing a mixture of functional requirements (FRs) and non-functional requirements (NFRs). Data analytics has revolutionized requirement engineering by enabling the efficient prediction and management of NFRs. Despite their importance, NFRs often receive less attention than FRs in the hierarchy of system requirements. The study [8] aimed to develop a reliable technique for predicting NFRs in software development. By utilizing data analytics techniques, the system analyzes user behavior, system performance, and environmental factors to predict NFRs. This systematic approach employs advanced machine learning (ML) and statistical methods to process and analyze data, identifying patterns and trends related to specific NFRs. The goal is to provide software developers with a reliable tool to proactively address potential NFR issues and optimize software systems. This research holds the potential to enhance software quality and performance by giving due attention to both FRs and NFRs.

Software failures often result from neglecting non-functional requirements (NFRs), emphasizing the need for automated prediction and management systems. Software Requirement Specifications (SRS) documents cover all stakeholder requirements and technical specifications. Despite ML and NLP advancements suggesting semi-automated methods for optimizing requirements, their complexity limits adoption. Recent research uses word embeddings like Word2Vec and FastText to build CNNs for categorizing functional requirements (FRs) [9,10]. Study [11] focused on managing NFRs in agile projects, while study [12] highlights the need for standardized approaches due to diverse expert opinions on NFRs. Currently, there is no standardized stage for evaluating NFRs in requirements engineering.

Authors of [13] proposed automating the identification of NFRs in user app reviews to help developers meet user needs. Study [14] investigated categorizing requirements based on app review change logs, which enhances classification accuracy for both FRs and NFRs. Study [15] emphasized the importance of early identification of NFRs in software development, noting the labor-intensive nature of this task and advocating for automated methods. Study [16] introduced an automated method using information retrieval techniques to identify and categorize NFRs, improving efficiency and accuracy, and aiding software teams in addressing critical requirements early in development.

The field of security requirements classification is increasingly important in software engineering [17]. However, the implementation of advanced ML methods is limited by the lack of extensive datasets. Study [18] focused on classifying software requirements with an emphasis on maintainability, while Mahmoud et al. [19] proposed a method for extracting NFRs to enhance software quality. Study [20] developed Urdu benchmark corpora for OCR training. Lu et al. [21] introduced an automated approach to categorize app user reviews into functional and non-functional requirements. Li et al. [22] classified NFRs into behavioral and visual categories. Studies [23] and [24] presented ML methodologies for NFR classification. Rahman et al. [25] used Word2Vec and RNNs for advanced requirements management. Study [26] applied NLP and ML methods in software contexts.

Win et al. [27] tested the Analytic Hierarchy Process on the RALIC dataset. Gnanasekaran et al. [28] explored ML for automatic NFR categorization, and Jha et al. [29] mined NFRs from app store reviews.

Study by Younas et al. [30] assessed differences between frequently used indicator terms and requirement statements to improve NFR classification accuracy. Handa et al. [31] provided a thorough analysis of recent methods developed for predicting NFRs. In [32], the authors proposed a refined approach to resolve conflicts, particularly when two NFRs are contradictory, by distinguishing between external and internal issues. Study [33] analyzed the structural aspects of a Shallow Artificial Neural Network (ANN) for classifying NFRs. Shah et al. [34] also used a Shallow ANN to categorize NFRs by examining its structural characteristics. Shankar et al. [35] explored the elicitation, formulation, and verification of NFRs in the early phases of engineering design projects. Another study [36] developed a model to extract and evaluate security aspects from scenarios based on predefined security goals and requirements. Cho et al. [37] used NLP and ML to differentiate FRs from NFRs. Researchers in [38] utilized ML to represent text data from SRS and classify requirements into FRs and NFRs. Horkoff [39] presented an ML-based method using massive data to train algorithms.

Researchers have developed various methods to enhance software requirement classification, addressing both FRs and NFRs. Numerous techniques have been proposed to improve the functionality, interpretability, and effectiveness of these classifications. For instance, study [40] conducted a comparative analysis of random forest and gradient boosting algorithms, concluding that random forest offers superior classification accuracy for FRs. Research [41] highlighted that random forest is faster to train and effective with large datasets, making it popular in industries for its ability to handle complex problems and maintain accuracy in imbalanced datasets. Despite its robustness and versatility, random forest has drawbacks in complexity [42], as each tree generates separate rules, creating numerous decision paths that complicate the model's interpretability. To improve the classification process, we propose several innovative strategies, including a modification of the random forest algorithm. This enhanced algorithm, named FNRC (Functional and Non-Functional Requirements Classifier), is meticulously designed to accurately classify both FRs and NFRs. Our approach focuses on refining the existing algorithm to more effectively distinguish and categorize diverse software development needs. The ultimate goal is to streamline the development process and significantly improve the quality of the final product.

3 Proposed Methodology

This section describes about the overall methodology of the proposed technique. It is based on the preprocessing of the data and detailed description of FNRC. The architecture of the proposed methodology is shown in Fig. 1.

3.1 Data Pre-Processing

We obtained datasets from PROMISE, a repository for software engineering research data, and CCHIT, which focused on health information technology [43,44]. The PROMISE dataset is a well-established collection of software engineering requirement documents, including project proposals and software specifications. We employed this dataset to evaluate the effectiveness of NFR extraction solution in the context of free-text documents, as they provide a realistic representation of requirements in their natural form. On the other hand, the CCHIT dataset consists of requirement statements specifically related to healthcare IT systems, presenting a challenge as each requirement statement contains multiple NFR types.

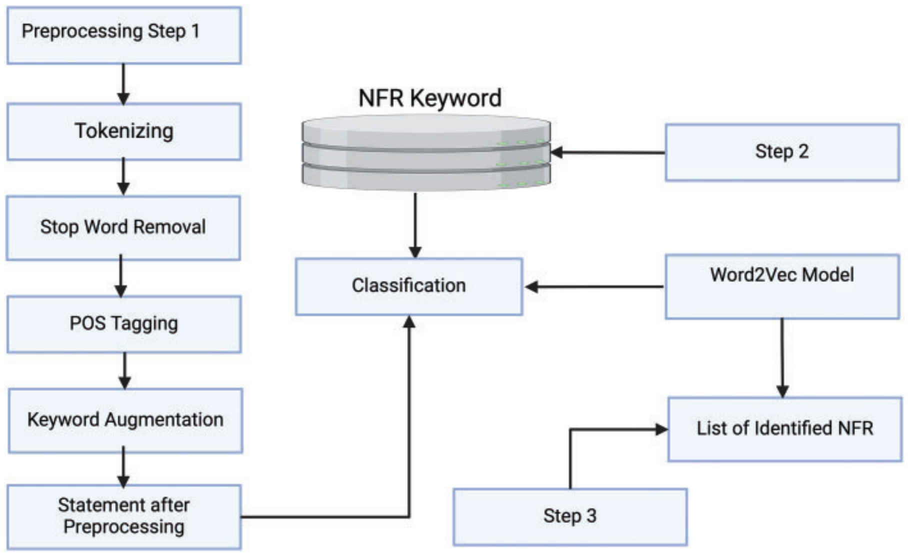

Figure 1: Proposed methodology for extraction of NFRS

For both datasets (PROMISE and CCHIT), we have used 70% of the data for training, 10% for pruning, and 20% for testing. Our experiments were conducted on a 40-core Intel machine with 4 × 2.4 GHz Intel 10-core E7-8870 Xeon processors and 256 GB of RAM, using R version 4.3.2 for modifications of the Random Forest algorithm and implementation of the Accuracy Sliding Window (ASW) mechanism. After the selection of datasets, we initiated the data pre-processing phase. This started with data tokenization, where input data was separated into individual tokens (words). Subsequently, we applied several pre-processing techniques to the dataset. During tokenization, we performed case folding to standardize all letters to uppercase, ensuring uniformity. Next, we removed stop words—such as pronouns, conjunctions, and prepositions—using a predefined list of irrelevant words for the analysis. This step involved scanning the text and filtering out these irrelevant words. Finally, we applied stemming and transformation, extracting the root forms of words through suffix stripping and lemmatization.

The transformation step is performed using TF-IDF, where we calculated the Term Frequency (TF)-Inverse Document Frequency (IDF) score. This TF-IDF-based information is then utilized to apply our proposed Functional and Non-functional Requirements Classifier (FNRC). The FNRC addresses previous shortcomings by introducing an Accuracy Sliding Window (ASW) mechanism for pruning trees to identify an optimal sub-ensemble. These optimal sub-ensembles require significantly fewer trees to achieve accuracy equal to or greater than that of the initial ensemble classifier. The detailed methodology of the proposed FNRC is explained in Section 3.

3.2 Functional and Non-Functional Requirements Classifier 'FNRC'

After pre-processing data, we split dataset as 70% of the dataset as the training dataset, 10% as the pruning dataset and 20% of the dataset as testing dataset. Before implementing FNRC, we train a baseline Random Forest model with the preprocessed data.

3.2.1 Random Forest

Random forest is a famous ML-based algorithm widely used by researchers for both regression and classification problems [45]. It gives output in the form of decision trees on different data samples as shown in Fig. 2. In the case of classification, it opts for the majority vote while in the case of regression, it takes the average value. It combines the tree to offer more accurate consolidated results. One of the famous features of the random forest is handling continuous variable data for classification and categorical variable data for regression. The major strength of this algorithm is handling complex data and eliminating the problem of over fitting.

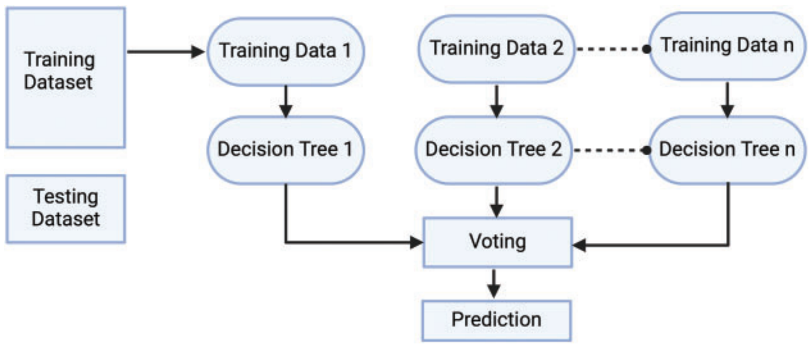

Figure 2: Working of random forest algorithm

The random forest algorithm is based on four steps as follows:

Step 1: In this step, a decision tree is built by using subsets of features and data points. n number of random data points and m features are selected from the dataset of k records. The typical order of magnitude for "n", "m", and "k" in the context of a Random Forest algorithm is $k \geq n > m$, with "k" being the largest as it represents the total size of the dataset, "n" being potentially equal to but often less than "k", and "m" being the smallest, signifying a subset of the total features.

Step 2: In this step, each sample is used to construct the individual tree.

Step 3: Here, the output is generated by using each decision tree.

Step 4: In the last step, the final output is obtained for classification by using majority voting and for regression by using average values.

The Random Forest algorithm, while powerful and versatile, does face limitations due to its inherent complexity, particularly in terms of interpretability and the generation of complex instructions. Each tree in a Random Forest generates its own set of rules based on the training data, and when combined into an ensemble, the multitude of decision paths can be difficult to trace and understand. This complexity can pose challenges in situations where explaining the model's decisions is crucial.

3.2.2 ASW Approach to Prune an Ensemble

To overcome the problem of complex instructions, a modified random forest algorithm is proposed named FNRC (Functional and Non-Functional Requirements Classifier) by introducing an Accuracy Sliding Window (ASW) for tree pruning to identify an optimal sub-ensemble. Tree pruning in ML involves trimming parts of the tree to simplify the model. In the context of Random Forest, it refers to reducing the number of trees in the ensemble. We introduced ASW to identify the most effective subset of trees that contribute to the overall accuracy, without unnecessarily complicating it. We performed the following steps:

Step-1: We started by training Random Forest algorithm, using our preprocessed dataset.

Step-2: To visualize and document the results, we used an algorithm titled "Classifier Ensemble Filtering" to prune a classifier ensemble. This algorithm creates a subset of effective classifiers based on the ensemble's performance on the pruning set (P_{rd}) and its ability to accurately classify both minority and majority classes.

Algorithm 1: Classifier Ensemble Filtering

```
Input: $C_{ens} = cf_{1...}cf_{\alpha}$ where $C_{ens}$ is a classifier ensemble that has been trained on $T_{rd}$, $P_{rd}$, $T_{sd}$
Data: There are three sets: the Training Set ($T_{rd}$), the Testing Set ($T_{sd}$), and the Pruning Set ($P_{rd}$)
Pruning:
1. Initialize:
2.  Let Cens be the initial ensemble of classifiers. // Initializing classifiers
3.  For each pj in pruning data set $P_{rd}$: // Checking the Pruning set
4.    Extract ci(xj), // ci's prediction on pj.
5.  For each classifier ci in Cens:
6.    Initialize Aij, Bij, Cij to 0 for each j
7.    For each data point xj in $P_{rd}$:
8.      If ci(xj) == yj and ci(xj) is in the minority group:
9.        Aij = 1   // ci correctly predicts yj, minority group
10.     Else if ci(xj) == yj and ci(xj) is in the majority group:
11.       Bij = 1   // ci correctly predicts yj, majority group
12.     Else if ci(xj) == yj:
13.       Cij = 1   // ci correctly predicts yj
14. // Calculate Composite Influence Score for classifier ci
15. ICi = ∑_(j=1)^N (Aij ∗ (2v(j)max - v(j)ci(xj)) + Bij ∗ v(j)sec + Cij ∗ (v(j)correct - v(j)ci(xj) -
    v(j)max));
16. Output: Sub Ensemble Classifier
```

Step-3: Evaluated the initial ensemble model's performance on the dataset to establish baseline accuracy.

Step-4: Begin the iterative pruning process:

- Temporarily removed the tree from the ensemble.
- Re-evaluated the ensemble's accuracy on the validation dataset without the removed tree.
- Determine if the accuracy of the ensemble with the tree removed falls within the predefined accuracy window.
- If it does, keep the tree removed; otherwise, put it back into the ensemble.

Step-5: Iterated through all the trees in the ensemble using the process described above. This resulted in a subset of trees that contribute to the desired accuracy.

Step-6: The subset of trees that remained after the iterative pruning process constitutes the optimal sub-ensemble. This subset contains trees that collectively contribute to the overall accuracy within the defined accuracy window.

Step-7: Assessed the performance of the optimal sub-ensemble on a separate testing dataset to ensure it maintains good generalization to unseen data.

We designed an algorithm titled "The Accuracy Sliding Window," which dynamically adjusts the number of trees in a Random Forest classifier to maximize accuracy. It iteratively evaluates the performance of the classifier with different numbers of trees and updates the ensemble based on accuracy improvements, ultimately resulting in an optimized ensemble of pruned trees. The detailed Algorithm 2 is discussed below:

Algorithm 2: The Accuracy Sliding Windows

```
1.  Classifier, nTree, and nCount
Input: Classifier, nTree, nCount  // Initializing input variables.
2. Data: $T_{rd}$, $P_{rd}$, $T_{sd}$  // Defining Training, Pruning, and Testing sets.
3. Initialize:
4.  Accuracy = calculateAccuracy(); // Function to calculate initial accuracy.
5. Pruning Process:
6.    While nTree  > 0 do: // Loop for processing Random Forest.
7.      RandomForestClassifier  <- generateRandomForest(nTree);  //Generating Forest.
8.     currentAccuracy  <- calculateAccuracy(RandomForestClassifier); // Calculating accuracy.
9.      If currentAccuracy  > Accuracy then:
10.       nTree  += 1; // Increment tree count.
11.       Accuracy = currentAccuracy; // Update accuracy.
12.     Else If (currentAccuracy  == Accuracy) and (nCount  <= 2) then:
13.       nTree  += 1; // Increment tree count if conditions are met.
14.       nCount  += 1; // Increment nCount  within threshold.
15. Output: Ensemble of Pruned Trees // Resultant pruned tree ensemble.
```

In Fig. 3, we illustrated our innovative method, which focuses on the elimination of underperforming DTs within the classifiers, resulting in a subsequent decrease in errors. The core concept of our proposed FNRC technique is to consistently and effectively reduce errors. In this process, the best DTs are retained, ultimately forming our optimal sub-ensemble. Notably, the graph depicted in the figure highlights a critical point where error rates begin to increase. It is at this juncture that we apply an accuracy sliding window, a mechanism in our proposed algorithm. This step is designed to maintain the accuracy of our proposed method, preventing it from deteriorating.

4 Results and Discussion

This section explains the experimental setup and results for the proposed technique.

4.1 Results on PROMISE Dataset

The PROMISE dataset, containing extensive potential NFR data, is used to gauge the applicability of the method outlined in Fig. 1. The Table 1 presents a breakdown of NFRs by category and an evaluation of our proposed method's accuracy. This evaluation is based on the detailed and labor-intensive task of manually categorizing each FR to identify it as an NFR. The implemented method assumes a one-to-one correspondence between each FR and NFR type, determined by evaluating the similarity between indicator keywords and the FR statement using a specific similarity metric. An NFR is considered necessary if its similarity score is the highest and exceeds a certain threshold value. The complete pseudo-code of the suggested method is detailed in Algorithm 1 and Algorithm 2. The classification performance achieved an accuracy of 75% on the dataset, with an F-measure of 64%. Precision and recall measures were calculated to provide further insight into these outcomes. Requirements with similarity values below the threshold were classified as FRs, while those above were categorized as NFRs. Applying this threshold, the system identified 41 feature representations as NFRs, specifically derived from the highest accuracy (True Positive) predictions in the context of usability. In our assessment, categories such as legality and portability demonstrated outstanding precision, achieving up to 100%. The maximum and minimum recall scores were 86% and 29%, respectively, with availability being the only NFR category where recall exceeded precision. This higher recall but lower precision indicates more false positives in this category. The type availability category showed the poorest performance in terms of recall, suggesting a need to refine the indicator keywords for certain NFR types. Fig. 4 provides a graphical representation of precision and recall for further clarity.

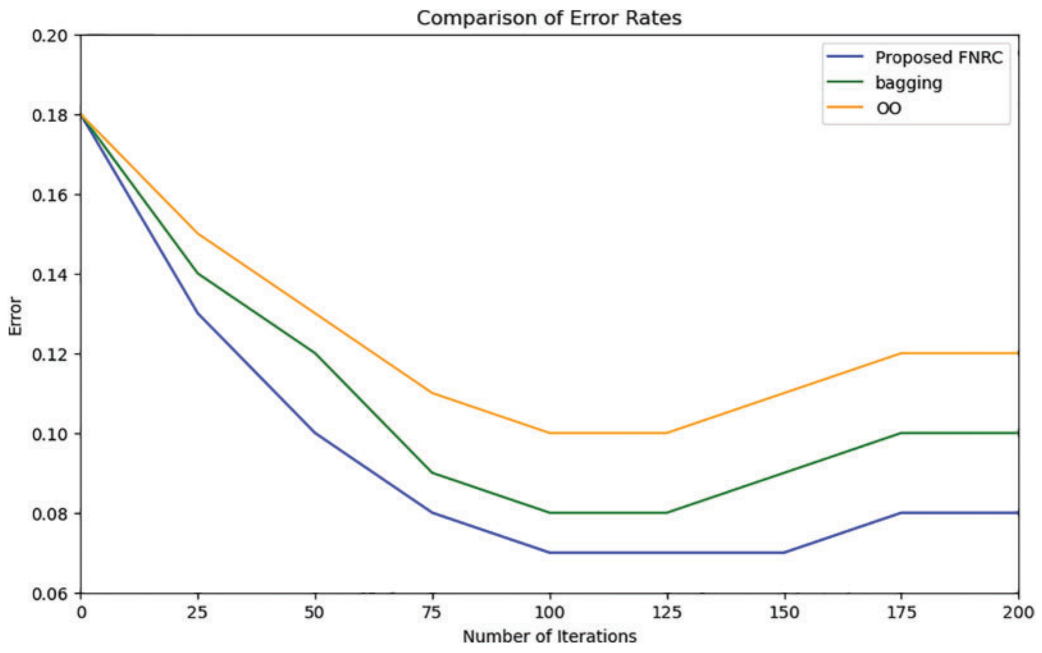

Figure 3: Comparison of error rates

Table 1: Performance evaluation of FNRC methodology at a criterion of 0.61

NFR	Actual	TP	FP	FN	P	R	F-measure
Availability	21	13	21	8	0.38	0.62	0.47
Legal	13	8	0	5	1.00	0.62	0.76
Look and feel	28	21	6	17	0.78	0.55	0.65
Maintainability	17	7	1	10	0.88	0.41	0.56
Operational	62	32	7	30	0.82	052	0.63
Performance	54	30	10	24	0.75	0.29	0.41

(Continued)

Table 1 (continued)

NFR	Actual	TP	FP	FN	P	R	F-measure
Scalability	21	6	2	15	0.75	0.53	0.63
Security	66	35	11	31	0.76	0.61	0.69
Usability	67	41	11	26	0.79	0.50	0.50
Fault Tolerance	10	5	5	5	0.50	1.00	1.00
Portability	1	1	0	0	1.00	0.86	0.72
Functional	255	219	133	36	0.62	0.59	0.64
Average					0.75	0.59	0.64

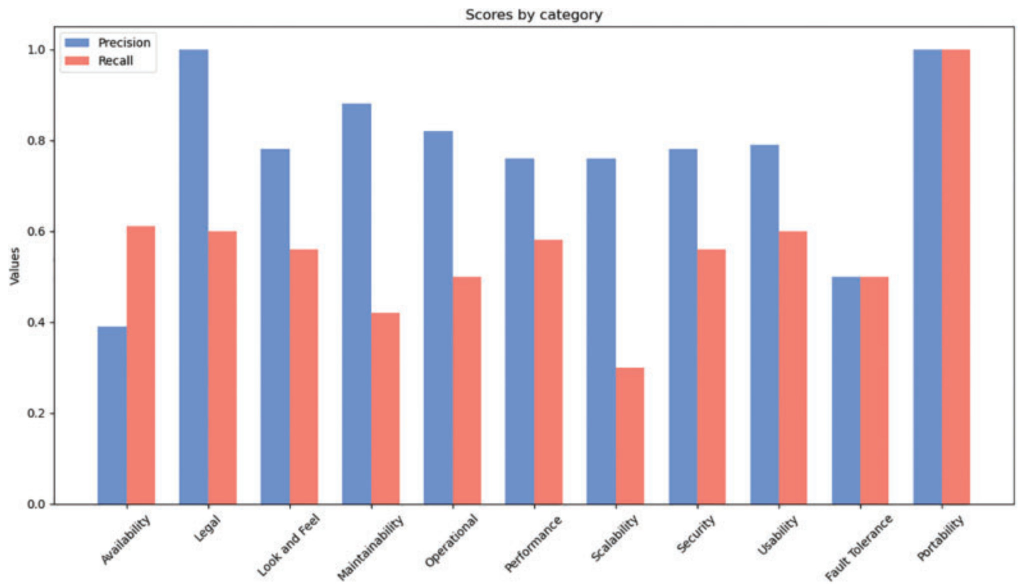

Figure 4: Precision and recall of different types of NFR

To address shortcomings in the current model's performance and align with the scholarly focus on the preliminary processing phase within the domain of Information Retrieval (IR), we employed a range of pre-processing methods to achieve effective solutions. In addition to our standard pre-processing techniques, we implemented a semantic similarity strategy to enhance results. Precision, recall, and F-measure metrics were systematically computed for numerical values ranging from 0.0 to 0.99. Table 2 provides a detailed comparative analysis of three different data processing methods: Keyword Augmentation, Part-Of-Speech (POS) Tagging, and Traditional Methods.

Table 2: Extraction of requirements by using different strategies at different threshold values

Threshold (λ)	Keyword augmentation			POS tagging			Traditional methods		
	Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
0	0.4250	0.6222	0.4683	0.4393	0.6235	0.4830	0.3372	0.4327	0.3484

(Continued)

Table 2 (continued)

Threshold (λ)	Keyword augmentation			POS tagging			Traditional methods		
	Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
0.1	0.4250	0.6222	0.4683	0.4393	0.6235	0.4830	0.3372	0.4327	0.3484
0.2	0.5091	0.6225	0.4695	0.5234	0.6238	0.4841	0.3372	0.4327	0.3484
0.3	0.5106	0.6238	0.4870	0.5435	0.6251	0.5160	0.3375	0.4327	0.3486
0.4	0.5524	0.6376	0.5528	0.5913	0.6376	0.5805	0.4531	0.4449	0.4151
0.5	0.6916	0.6452	0.6535	0.6904	0.6426	0.6513	0.5065	0.4111	0.4228
0.55	0.7280	0.6391	0.6680	0.7264	0.6353	0.6649	0.5030	0.3786	0.3988
0.6	0.7512	0.5980	0.6463	0.7489	0.5930	0.6420	0.5039	0.3448	0.3694
0.61	0.7523	0.5881	0.6385	0.7501	0.5831	0.6342	0.5073	0.3442	0.3695
065	0.7454	0.5546	0.6103	0.7383	0.5422	0.5977	0.5064	0.3210	0.3522
0.7	0.6747	0.4123	0.4834	0.6677	0.3999	0.4699	0.5379	0.2932	0.3356
0.8	0.6717	0.3273	0.3956	0.6671	0.3125	0.3784	0.5520	0.2194	0.2563
0.9	0.7184	0.1857	0.2255	0.7203	0.5660	0.6100	0.6225	0.1413	0.1546
0.99	0.4808	0.1149	0.1093	0.4808	0.1149	0.1093	0.0340	0.0833	0.0612

Table 2 provides insights into the performance of various pre-processing methods, evaluated using precision, recall, and F-measure across a range of threshold values from 0 to 0.99. In the traditional method, the highest recall of 45% occurs at a threshold of 0.4, an F-measure of 42% at 0.5, and the highest precision of 62% at 0.9. However, the 62% precision comes with a recall of only 14%. At a threshold of 0.5, the average performance is 50% precision, 41% recall, and 42% F-measure.

In the Word2Vec method, part of our pre-processing toolkit, words are selected based on their POS tags, such as NN for common nouns, JJ for adjectives, NP for proper nouns, and DT for determiners. According to source [35], nouns and adjectives are often prospective keywords. For recall, the highest value obtained is 64% at a threshold of 0.5, an F-measure of 66.5% at 0.55, and a precision of 75% at 0.61. The improvements in precision, recall, and F-measure are 0.23%, 0.26%, and 0.30%, respectively. At a threshold of 0.55, the average performance is 73% precision, 64% recall, and 66% F-measure.

In the third phase of our pre-processing approach, we focused on keyword augmentation. After applying Word2Vec, some requirement statements became too brief, with only three or fewer words, adversely affecting semantic similarity due to insufficient content. This issue can be mitigated by repeating the statement, which broadens the dataset, increases keyword possibilities, and improves threshold achievement. This method of statement expansion proves effective across various types of NFRs, demonstrating its robustness regardless of the specific NFR category. According to the results shown in Table 2, the highest recall is 64.5%, the highest F-measure is 66.8%, and the highest precision is 75.23%. The improvements are 25% in precision, 20% in recall, and 24% in F-measure. Although only a small fraction of requirement statements undergoes word augmentation processing, this pre-processing step is efficient. If there are many requirement statements with fewer than four words, then pre-processing becomes even more beneficial.

In our research, we conducted experiments on the PROMISE dataset to compare the performance of three methodologies: Random Forest with Word2Vec, Random Forest with Classical Feature Extraction, and our proposed FNRC, which enhances Random Forest with an Accuracy Sliding Window. The results, shown in Fig. 5, illustrate the performance metrics Precision, Recall, and F-measure across these approaches. FNRC achieved the highest Precision, Recall, and F-measure, demonstrating its superior effectiveness. This comparison underscores the advantages of FNRC over state-of-the-art methods and standard Random Forest techniques.

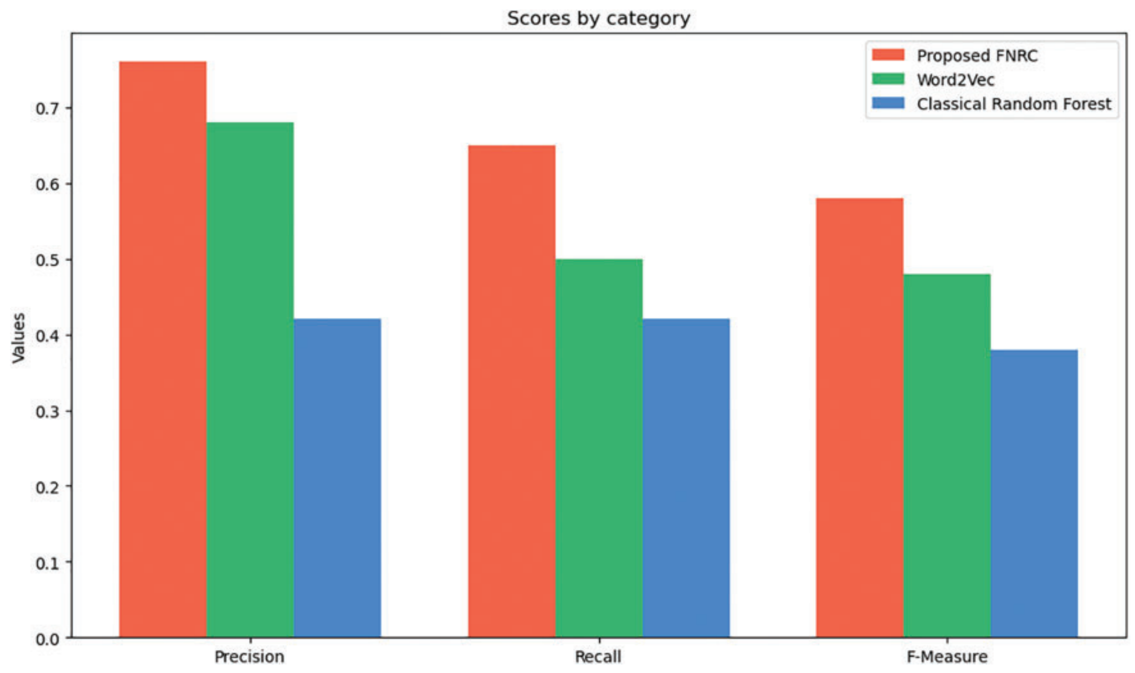

Figure 5: The effectiveness of classical, Word2Vec, and proposed FNRC extraction

4.2 Results on CCHIT Dataset

We evaluated our method's reliability using the CCHIT Ambulatory dataset, which includes data related to ambulatory care. Table 3 presents the results of our proposed method for identifying various NFR types within this dataset, using a threshold of 0.60 after applying Word2Vec and the FNRC pre-processing. The method utilizes the top 20 keywords identified from research [11] as indicators for categorizing NFRs. In analyzing the CCHIT dataset, we observed that a single entry can relate to several NFR types. This complexity requires applying different criteria to determine the relevance of each NFR type, considering them significant if they score above the set threshold. Due to the multiple NFR types in CCHIT requirement statements, the recall value is higher than that achieved with the PROMISE dataset, which has less complexity and fewer NFR types per statement. However, the CCHIT dataset has a lower precision value than the PROMISE dataset, as the complexity of having multiple NFR types per statement in CCHIT leads to more false positives.

In our paper, we identified specific quality Non-Functional Requirements (NFRs) that align with the ISO 9126 model, including operability, usability, fault tolerance, portability, efficiency, reliability, and maintainability. These attributes conform to the standards set by ISO 9126. Additionally, we explored quality NFR attributes not covered by ISO 9126 but present in the ISO 25010 standard, such as legal, look and feel, performance, access control, security, an expanded view of maintainability, and scalability.

Table 3: Performance evaluation of FNRC methodology at a criterion of 0.60

Requirements	P	R	F-measure
Access control	0.60	0.75	0.67
Audit	0.67	0.81	0.73
Availability	0.33	1.00	0.50
Legal	0.50	1.00	0.67
Maintenance	0.58	0.78	0.67
Recoverability	0.50	0.60	0.55
Performance	0.33	1.00	0.50
Reliability	0.40	1.00	0.57
Security	0.70	0.82	0.75
Usability	0.50.	0.75	0.60
Functional	0.90	0.84	0.87

5 Threats to Validity

In this section, we outlined potential threats that may impact the validity of our research. While we aim to comprehensively address these threats, we also provided information on the steps taken to mitigate their impact.

The applicability of the feature extraction model to other datasets or domains is a concern of this research. Threats to external validity arise when the model's performance on PROMISE and CCHIT datasets does not generalize to different contexts. To address this, we tested the model on diverse datasets representative of various domains. If the datasets used are not representative of the broader context, external validity is compromised. We mitigate this by carefully selecting datasets that align with the research's domain of interest.

Data quality and preprocessing methodologies can impact the results. Inadequate data cleaning or inconsistent preprocessing methods may introduce noise. To enhance internal validity standardized preprocessing methods were established to ensure uniformity in data transformation steps, such as normalization, tokenization, and stemming. Additionally, data quality assurance measures were implemented, encompassing data validation checks and quality control processes that addressed issues during data collection and pre-processing.

Internal validity threats pertain to the accuracy of the research findings. Threats can arise if the FNRC model's implementation or parameter choices are not optimal. Any errors or suboptimal parameter settings can compromise the internal validity of the results. When introducing the accuracy sliding window, we ensured that the FNRC model's implementation is error-free and that parameters such as window size and sliding frequency were chosen appropriately.

6 Conclusion and Future Work

In this paper, the FNRC algorithm, an enhanced version of the random forest approach designed to address challenges related to high time complexity and low accuracy. By incorporating the Accuracy Sliding Window (ASW), the FNRC algorithm effectively prunes trees to create optimal sub-ensembles. These sub-ensembles achieve accuracy levels that meet or exceed those of the initial ensemble classifier while using significantly fewer trees. Experimental findings demonstrate the effectiveness of the FNRC algorithm, which utilizes classical pre-processing techniques such as tokenization, stop word removal, and lemma generation in combination with a semantic similarity strategy. Through a detailed analysis of precision, recall, and F-measure values across various threshold levels (ranging from 0.0 to 0.99), the study identified optimal points for these metrics. Specifically, a recall of 45% was achieved at a threshold of 0.4, an F-measure of 42% at 0.5, and a precision of 62% at 0.9, although this high precision came with a low recall of 14%. At a significance level of 0.5, the average performance showed 50% precision, 41% recall, and 42% F-measure. The FNRC algorithm competes with state-of-the-art algorithms across various metrics, including classification accuracy, time complexity, and memory consumption, confirming its potential to transform classification paradigms and tackle complex data while maintaining computational efficiency.

Future work should focus on refining the FNRC algorithm by exploring parameter optimization and fine-tuning the ASW-based pruning technique for improved accuracy and efficiency. Efforts to extend the algorithm's applicability to larger and more diverse datasets, along with investigations into hybrid models, could lead to broader practical implementations. Enhancing the interpretability of the FNRC algorithm's decisions, integrating it into real-world software development processes, and seeking user feedback for iterative improvements are essential steps. Additionally, staying current with evolving techniques and adapting the algorithm to specific industries will ensure its continued relevance and effectiveness in addressing NFR classification challenges.

Acknowledgement: The authors would like to acknowledge the support provided by EIAS (Emerging Intelligent Autonomous Systems) Data Science Lab, Prince Sultan University, Kingdom of Saudi Arabia.

Funding Statement: This work is supported by EIAS (Emerging Intelligent Autonomous Systems) Data Science Lab, Prince Sultan University, Kingdom of Saudi Arabia, by paying the APC.

Author Contributions: The authors confirm contribution to the paper as follows: Study conception and design: Jalil Abbas, Arshad Ahmad. Analysis and interpretation of results: Jalil Abbas, Arshad Ahmad, Syed Muqsit Shaheed, Draft manuscript preparation: All authors. Funding and support: Sajid Shah, Mohammad ELAffendi, Gauhar Ali. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are publicly available on: PROMISE Dataset: https://github.com/AleksandarMitrevski/se-requirements-classification (accessed on 15 January 2024). Certification Commission for Health Information Technology: http://www.cchit.org (accessed on 15 January 2024).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] J. Abbas, "Quintessence of traditional and agile requirement engineering," *J. Softw. Eng. Appl.*, vol. 9, no. 3, pp. 63–70, Mar. 2016. doi: 10.4236/jsea.2016.93005.

[2] I. Khurshid *et al.*, "Classification of non-functional requirements from IoT oriented healthcare requirement document," *Front Public Health*, vol. 10, pp. 860536, 2022. doi: 10.3389/fpubh.2022.860536.

[3] S. Qureshi, S. U. R. Khan, I. Ur-Rehman, Y. Javed, S. Saleem and A. Iqbal, “A conceptual model to address the communication and coordination challenges during requirements change management in global software development,” *IEEE Access*, vol. 9, pp. 123456–123470, Jun. 2021. doi: 10.1109/ACCESS.2021.3091603.

[4] L. Kumar, S. Baldwa, S. M. Jambavalikar, L. B. Murthy, and A. Krishna, “Software functional and non-functional requirement classification using word-embedding,” *Proc. Int. Conf. Adv. Inf. Netw. Appl.*, vol. 450, Mar. 2022, pp. 167–179.

[5] R. Jindal, R. Malhotra, A. Jain, and A. Bansal, “Mining non-functional requirements using machine learning techniques,” *e-Informatica Softw. Eng. J.*, vol. 15, no. 1, pp. 85–114, 2021.

[6] K. Sheha deh, N. Arman, and F. Khamayseh, “Semi-automated classification of Arabic user requirements into functional and non-functional requirements using NLP tools,” in *Proc. Int. Conf. Inf. Technol. (ICIT)*, Jul. 2021, pp. 527–532.

[7] A. Smola and S. V. N. Vishwanathan, *Introduction to Machine Learning*, 1st ed., Cambridge, UK: Cambridge University Press, 2008, vol. 32, no. 34.

[8] N. Handa, A. Sharma, and A. Gupta, “Framework for prediction and classification of non-functional requirements: A novel vision,” *Cluster Comput.*, vol. 25, no. 2, pp. 1155–1173, Apr. 2022. doi: 10.1007/s10586-021-03484-0.

[9] S. Jp, V. K. Menon, K. P. Soman, and A. K. Ojha, “A non-exclusive multi-class convolutional neural network for the classification of functional requirements in AUTOSAR software requirement specification text,” *IEEE Access*, vol. 10, pp. 117707–117714, 2022. doi: 10.1109/ACCESS.2022.3217752.

[10] N. Rahimi, F. Eassa, and L. Elrefaei, “An ensemble machine learning technique for functional requirement classification,” *Symmetry*, vol. 12, no. 10, pp. 1601, Oct. 2020. doi: 10.3390/sym12101601.

[11] M. Alenezi, H. A. Basit, M. A. Beg, and M. S. Shaukat, “Synthesizing secure software development activities for linear and agile lifecycle models,” *Softw. Pract. Exp.*, vol. 52, no. 6, pp. 1426–1453, Jun. 2022. doi: 10.1002/spe.3072.

[12] M. Binkhonain and L. Zhao, “A review of machine learning algorithms for identification and classification of non-functional requirements,” *Expert Syst. Appl. X*, vol. 1, no. 2, pp. 100001, Jan. 2019. doi: 10.1016/j.eswax.2019.100001.

[13] D. Dave and V. Anu, “Identifying functional and non-functional software requirements from user app reviews,” in *Proc. IEEE Int. IOT, Electron. Mechatron. Conf. (IEMTRONICS)*, Jun. 2022, pp. 1–6.

[14] C. Wang, F. Zhang, P. Liang, M. Daneva, and M. Van Sinderen, “Can app changelogs improve requirements classification from app reviews? An exploratory study,” in *ESEM ’18: Proc. 12th ACM/IEEE Inter. Symp. Empiri. Software Engin. Measure.*, pp. 1–4. doi: 10.1145/3239235.32674.

[15] P. Leelaprute and S. Amasaki, “A comparative study on vectorization methods for non-functional requirements classification,” *Inf. Softw. Technol.*, vol. 150, no. 11–12, pp. 106991, Oct. 2022. doi: 10.1016/j.infsof.2022.106991.

[16] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, “The detection and classification of non-functional requirements with application to early aspects,” in *Proc. 14th IEEE Int. Requir. Eng. Conf. (RE’06)*, 2006, pp. 39–48. doi: 10.1109/RE.2006.65.

[17] P. Kadebu, S. Sikka, R. K. Tyagi, and P. Chiurunge, “A classification approach for software requirements towards maintainable security,” *Sci. Afr.*, vol. 19, no. 1, pp. e01496, Jun. 2023. doi: 10.1016/j.sciaf.2022.e01496.

[18] M. Ahmed, S. U. R. Khan, and K. A. Alam, “An NLP-based quality attributes extraction and prioritization framework in Agile-driven software development,” *Autom. Softw. Eng.*, vol. 30, no. 1, pp. 7, Jan. 2023. doi: 10.1007/s10515-022-00371-9.

[19] A. Mahmoud and G. Williams, “Detecting, classifying, and tracing non-functional software requirements,” *Requir. Eng.*, vol. 21, no. 3, pp. 357–381, Oct. 2016. doi: 10.1007/s00766-016-0252-8.

[20] T. Nasir, M. K. Malik, and K. Shahzad, “MMU-OCR-21: Towards end-to-end Urdu text recognition using deep learning,” *IEEE Access*, vol. 9, pp. 124945–124962, Oct. 2021. doi: 10.1109/ACCESS.2021.3110787.

[21] M. Lu and P. Liang, "Automatic classification of non-functional requirements from augmented app user reviews," in *Proc. 21st Int. Conf. Eval. Assess. Softw. Eng.*, Jun. 2017, pp. 344–353.

[22] X. Li, T. Wang, P. Liang, and C. Wang, "Automatic classification of non-functional requirements in app user reviews based on system model," *Acta Electron. Sin.*, vol. 50, no. 9, pp. 2079–2089, Sep. 2022. doi: 10.12263/DZXB.20210454.

[23] M. A. Haque, M. A. Rahman, and M. S. Siddik, "Non-functional requirements classification with feature extraction and machine learning: An empirical study," in *Proc. 1st Int. Conf. Adv. Sci., Eng. Robot. Technol. (ICASERT)*, May 2019, pp. 1–5. doi: 10.47489/p000s331z702-1-5mc.

[24] E. Dias Canedo and B. Cordeiro Mendes, "Software requirements classification using machine learning algorithms," *Entropy*, vol. 22, no. 9, pp. 1057, Sep. 2020. doi: 10.3390/e22091057.

[25] M. A. Rahman, M. A. Haque, M. N. A. Tawhid, and M. S. Siddik, "Classifying non-functional requirements using RNN variants for quality software development," in *3rd ACM SIGSOFT Inter. Workshop,* ACM, pp. 25–30, Aug. 2019. doi: 10.1145/3340482.334274.

[26] L. Tóth and L. Vidács, "Comparative study of the performance of various classifiers in labeling non-functional requirements," *Inf. Technol. Control*, vol. 48, no. 3, pp. 432–445, Sep. 2019. doi: 10.5755/j01.itc.48.3.21973.

[27] T. Z. Win, R. Mohamed, and J. Sallim, "Requirement prioritization based on non-functional requirement classification using hierarchy AHP," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 769, no. 1, pp. 012060, Feb. 2020. doi: 10.1088/1757-899X/769/1/012060.

[28] R. K. Gnanasekaran, S. Chakraborty, J. Dehlinger, and L. Deng, "Using recurrent neural networks for classification of natural language-based non-functional requirements," in *REFSQ Workshops*, vol. 2857, Oct. 2021.

[29] N. Jha and A. Mahmoud, "Mining non-functional requirements from app store reviews," *Empir Softw. Eng.*, vol. 24, no. 6, pp. 3659–3695, Oct. 2019. doi: 10.1007/s10664-019-09716-7.

[30] M. Younas, D. N. Jawawi, I. Ghani, and M. A. Shah, "Extraction of non-functional requirement using semantic similarity distance," *Neural Comput. Appl.*, vol. 32, no. 11, pp. 7383–7397, Aug. 2020. doi: 10.1007/s00521-019-04226-5.

[31] N. Handa, A. Sharma, and A. Gupta, "An inclusive study of several machine learning based non-functional requirements prediction techniques," in *Proc. 2nd Int. Conf. Futuristic Trends Netw. Comput. Technol. (FTNCT),* Chandigarh, India, 2020, pp. 482–493.

[32] U. Shah, S. Patel, and D. C. Jinwala, "Detecting intra-conflicts in non-functional requirements," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 29, no. 3, pp. 435–461, Jun. 2021. doi: 10.1142/S0218488521500197.

[33] D. A. López-Hernández, E. Mezura-Montes, J. O. Ocharán-Hernández, and A. J. Sánchez-García, "Non-functional requirements classification using artificial neural networks," in *2021 IEEE Inter. Autumn Meet. Power, Electr. Comput. (ROPEC)*, vol. 2021, no. 1, pp. 69–85, Oct. 2021. doi: 10.1109/ROPEC53248.2021.9668132.

[34] U. Shah, S. Patel, and D. Jinwala, "An ontological approach to specify conflicts among non-functional requirements," in *Proc. 2nd Int. Conf. Geoinformatics Data Anal.*, Mar. 2019, pp. 145–149.

[35] P. Shankar, B. Morkos, D. Yadav, and J. D. Summers, "Towards the formalization of non-functional requirements in conceptual design," *Res. Eng. Des.*, vol. 31, no. 4, pp. 449–469, Oct. 2020. doi: 10.1007/s00163-020-00345-6.

[36] A. Abdel Qader, "A novel intelligent model for classifying and evaluating non-functional security requirements from scenarios," *Indones J. Electr. Eng. Comput. Sci.*, vol. 15, no. 3, pp. 1578–1585, Mar. 2019. doi: 10.11591/ijeecs.v15.i3.pp1578-1585.

[37] B. -S. Cho and S. -W. Lee, "A comparative study on requirements analysis techniques using natural language processing and machine learning," *J. Korea Soc. Comput. Inf.*, vol. 25, no. 7, pp. 27–37, Jul. 2020. doi: 10.9708/jksci.2020.25.7.027.

[38] G. Y. Quba, H. Al Qaisi, A. Althunibat, and S. AlZu'bi, "Software requirements classification using machine learning algorithm's," in *2021 Int. Conf. Inf. Technol. (ICIT)*, IEEE, 2021.

[39] J. Horkoff, "Non-functional requirements for machine learning: Challenges and new directions," in *2019 IEEE 27th Int. Requir. Eng. Conf. (RE)*, IEEE, 2019.
[40] L. F. Li, N. C. Jin-An, Z. M. Kasirun, and Y. P. Chua, "An empirical comparison of machine learning algorithms for classification of software requirements," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 5, pp. 69–85, Oct. 2019. doi: 10.14569/IJACSA.2019.0100510.
[41] R. Raymond and M. A. Savarimuthu, "Retrieval of interactive requirements for data intensive applications using random forest classifier," *Informatica*, vol. 47, no. 9, pp. 69–85, Oct. 2023. doi: 10.31449/inf.v47i9.12345.
[42] C. Tang, D. Garreau, and U. von Luxburg, "When do random forests fail?" *Adv. Neural Inf. Process. Syst.*, vol. 31, pp. 69–85, Oct. 2018.
[43] A. Mitrevski, "SE requirements classification," *GitHub*, 2024. Accessed: Jun. 05, 2024. [Online]. Available: https://github.com/AleksandarMitrevski/se-requirements-classification
[44] US Certification Authority for Electronic Health Records (EHR) and Networks, Certification Commission for Health Information Technology (CCHIT). Accessed: Jun. 05, 2024. [Online]. Available: http://www.cchit.org
[45] Random Forest Algorithm, "Machine learning random forest algorithm," Accessed: Jun. 05, 2024. [Online]. Available: https://www.javatpoint.com/machine-learning-random-forest-algorithm