EasyChair Preprint

№ 10531

Intelligent Requirements Engineering: Applying Machine Learning for Requirements Classification

Mo'Ath Shatnawi, Ahmad Audat and Marah Saraireh

July 10, 2023

Intelligent Requirements Engineering: Applying Machine Learning for Requirements Classification

1st Mo'ath Shatnawi
Philadelphia University
Amman, Jordan
msshatnawi5@gmail.com

2nd Ahmad Audat
Philadelphia University
Amman, Jordan
audat.ahmad@gmail.com

3rd Marah Saraireh
Philadelphia University
Amman, Jordan
marahyahia160@gmail.com

***Abstract*—The classification of requirements plays a crucial role in requirements engineering, enabling the differentiation between legally relevant requirements and auxiliary content. However, the manual labeling of each content element in a requirements specification as a "functional requirement" or "non-functional requirement" or "information" is a time-consuming and error-prone task. In this paper, we propose an approach that automates the classification of content elements in a natural language requirements specification as either "functional requirement" or "non-functional requirement" or "information" Our approach leverages a combination of Convolutional Neural Networks (CNN) and Support Vector Machines (SVM) for the classification task. The CNN model is responsible for extracting meaningful features from the textual content, while the SVM classifier is employed to make the final classification decision. To train and validate our model, we utilized online datasets specifically designed for requirements classification. Additionally, we augmented these datasets by incorporating data from other projects. The performance of our model was measured using various evaluation metrics, including accuracy, F1 score, precision, recall, and confusion matrix analysis. The experimental results demonstrate promising performance with a precision of 80%, accuracy of 85%, F1 score of 88%, and recall of 90%. These results indicate that our approach successfully automates the classification process and significantly reduces the need for manual labeling, thereby saving time and reducing the potential for errors in requirements classification.**

***Index Terms*—Requirements classification, natural language processing, Convolutional Neural Networks, Support Vector Machines, accuracy, F1 score, precision, recall, Machine Learning, Datasets**

I. Introduction

Artificial Intelligence (AI) is what defines machine intelligence or computer intelligence because of the intelligence provided by computers and machines that simulate the intelligence and capabilities of a person, such as prediction, learning, or conclusion, as well as a term concerned with the manufacture of computers, prediction, or the detection of something [1]. Machine Learning is one of the most important types of artificial intelligence, through which computers are programmed through specific algorithms taught to them through experience and training so that computers can implement new orders based on the data they have been trained on [2]. Requirements engineering plays a crucial role in machine learning by defining and operationalizing the goals and constraints that need to be fulfilled in a system as required by stakeholders. It involves a systematic and structured process that begins with domain analysis, followed by requirement elicitation, specification, evaluation, negotiation, and documentation. The ultimate objective is to establish clear responsibilities for all stakeholders and achieve high-quality requirements, despite the inherent challenges associated with this process [3]. In requirements engineering, the accurate classification of requirements plays a vital role in ensuring the successful development and implementation of software systems [4]. The ability to differentiate between legally relevant requirements and auxiliary content is crucial for effective decision-making, compliance, and overall project success. However, manually labeling each content element in a requirements specification as a "functional requirement" or "non-functional requirement" can be a labor-intensive and error-prone task [5]. As a result, there is a growing interest in leveraging machine learning techniques to automate the requirements classification process.

In this research, our objective is to develop a requirements classification model utilizing artificial intelligence techniques, particularly machine learning. We aim to leverage the power of machine learning algorithms to effectively classify requirements based on their nature and characteristics. We will employ a dataset that is readily available in kaggle. Additionally, we will augment this dataset by incorporating new requirements gathered from various other projects. To structure our research, we will begin by reviewing previous studies that have explored similar works in the field of requirements classification. Next, we will focus on the collecting the dataset. We will gather the initial dataset from kaggle,and we will collect additional requirements from different projects, ensuring a comprehensive representation of real-world requirements.The next section we will proceed to develop our requirements classification model. by Convolutional Neural Networks (CNN) and Support Vector Machines (SVM) and k-nearest neighbor (KNN). We will measure various evaluation metrics, including accuracy, precision, recall, F1 score, and analyze the confusion matrix.

Finally, we will conclude our research by summarizing the findings and discussing the implications of our developed requirements classification model. We will highlight the contributions of our work in leveraging machine learning techniques

This work is supported by the Deanship of Scientific Research in Philadelphia University, Jordan.

for more accurate and efficient requirements classification. The methodology of our research, which encompasses the following steps:
- Studying Previous Intelligent Models for Requirements Classification.
- Acquiring an initial dataset by leveraging datasets available in Kaggle and GitHub.
- Oversampling by Collecting new datasets of requirements from various projects and applications. - Conducting data preprocessing to prepare the textual data for analysis.
- Implementing several classification models, including CNN, SVM, and KNN.
- Evaluating the performance of the models using accuracy, precision, recall, and F1-score metrics.

II. Previous Intelligent Models for requirements Classification

In this section, we provide an overview of the strengths and limitations of the research on of Requirements Classification model using machine learning.

The research titled "Automatic Classification of Requirements Based on Convolutional Neural Networks" conducted by Jonas Winkler and Andreas Vogelsang [6], focused on the categorization of requirements and information without explicitly distinguishing between functional and non-functional requirements. By employing Convolutional Neural Networks (CNN), the study aimed to automate the classification process, improving the efficiency and accuracy of requirements categorization.

One notable strength of the research lies in its utilization of CNNs to automatically classify requirements and information. CNNs, known for their effectiveness in pattern recognition tasks, were adapted to handle textual representations of requirements. This innovative application of CNNs showcases the researchers' ability to leverage deep learning techniques for automating the classification process, enabling faster and more reliable categorization.

However, it is important to note that the research had a limitation in terms of the specific categorization of requirements as functional or non-functional. While the study successfully classified requirements and information, it did not explicitly differentiate between different types of requirements. This omission restricts the ability to analyze and distinguish between functional and non-functional requirements, which are crucial distinctions in requirements engineering. Incorporating such categorization could provide a more comprehensive understanding of the requirements and their characteristics.

The secend research paper "Software Requirements Classification using Machine Learning algorithm's" by Gaith Y Quba et al. [7], addresses the challenge of automating the classification of software requirements. Previous attempts to automate this process have been insufficient, leading to the need for an improved approach. This study aims to fill this gap by proposing a technique that utilizes machine learning to automatically classify software requirements into two classes: Functional Requirements and Non-Functional Requirements.

In the methodology, the study utilizes the PROMISE_exp dataset, which contains labeled software requirements. The dataset undergoes preprocessing steps such as normalization, extractions, and the selection of suitable techniques. The text data from software requirements specifications is represented using the Bag-of-Words (BoW) technique. To perform the classification, two machine learning algorithms, Support Vector Machine (SVM) and K-Nearest Neighbors (KNN), are employed. The proposed technique's effectiveness is evaluated using metrics such as Precision, Recall, and F-measure.

One strength of this research lies in its utilization of machine learning algorithms to automate the classification of software requirements. By leveraging machine learning techniques, the study addresses the challenges associated with manual classification, including the effort, time, and cost involved. Automating the classification process not only improves efficiency but also enhances the accuracy by reducing potential human errors.

However, a potential weakness of the study is its reliance on the PROMISE_exp dataset for evaluation. The extent to which this dataset accurately represents real-world software requirements may limit the generalizability of the proposed technique. Additionally, the research lacks a comprehensive analysis of the performance and limitations of the SVM and KNN algorithms used. Further exploration and comparison with alternative algorithms would provide a more comprehensive understanding of the most effective machine learning approach for requirements classification.

Where we also expanded the study to include more research which used different methods and algorithms to classify requirements, it summarizes their used approaches, proposed algorithms, used dataset, and the result for each study.
The paper [10] highlights the crucial role of security in software development, emphasizing the need for clear security requirements in the Software Requirement Specification (SRS). It identifies a common issue where these requirements are often inadequately defined, leading to potential security vulnerabilities. To address this, the authors propose a method to mine security-related requirements from the SRS and classify them into specific categories. The method uses text mining and the J48 decision tree algorithm to develop prediction models for each security type. This approach aims to improve future software development by facilitating early security requirement identification and thus enhancing the reliability of software. The authors present a three-step methodology for classifying security requirements based on their descriptions. They first gather security requirements from a public dataset and select descriptions related to security issues. Then, they apply text mining techniques, including preprocessing, feature selection, and TF-IDF weighting to convert the text data into a structured format. In the final step, they develop four binary prediction models using the J48 decision tree method, each focused on a specific type of security requirement: authentication-authorization, access control, cryptography-encryption, and data integrity. The performance of these models is assessed using Receiver Operating Characteristic (ROC) analysis. More-

over, The study [11] proposed a novel approach to address misclassification issues in Non-Functional Requirements (NFRs) using a Convolutional Neural Network (CNN) based multi-label classifier. The goal was to automate the process of classifying stakeholder requirements into five classes of NFRs: reliability, usability, portability, maintainability, and efficiency. The implementation of this approach occurs in two steps: corpus construction and annotation, followed by feature extraction and CNN training. In step 1, the procedure starts by identifying critical NFR attributes—reliability, efficiency, portability, usability, and maintainability—based on various software quality models. Software requirement specification (SRS) documents are then collected, and researchers manually extract requirements related to the selected NFR attributes to create a dataset. This dataset creation process follows the MATTER-MAMA framework to ensure a representative and balanced corpus. Subsequently, each requirement related to selected categories is reassessed by a crowd to minimize the chance of mislabeling, and three annotators perform specifications to the dataset based on guidelines and instructions. The outcome of step 1 is a multi-labeled corpus. In step 2, a CNN is designed for feature extraction and classification. The labeled corpus from step 1 is used to train a word-level CNN for multi-label text classification using TensorFlow and Keras. Feature extraction is performed on word2vec, which is used to determine class labels based on the semantic dependency and relatedness of words in a requirement. The CNN generates dense vectors for word representation, predicting labels with improved fit criteria for classification.

Furthermore, the authors [12] applied a systematic approach that involved filtering and selecting papers discussing NFR catalogs. Inclusion and exclusion criteria were established, and a systematic 'lightweight' mapping study was conducted on 20 selected papers representing diverse NFRs related to security, performance, and usability. Keywords were collected from 31 catalogs to create a dataset, ensuring duplication avoidance and comprehensive coverage. The final dataset included 77 words across three categories: usability, security, and performance. The methodology involved four steps: visualizing a "lightweight" methodological process, converting the sig parser to CSV format, keyword embedding, and QA. The parser was developed to convert SIG catalogs into CSV files, and the quality assurance process ensured search efficiency and classification accuracy. The Tera-PROMISE Open Science data set of 625 specifications was used for evaluation. This specification, obtained from 15 master's degree projects, covered 11 types of NFRs. The authors focused on four specific types: security, performance, operational, and usability, resulting in 187 related statements.

This research paper [13] addresses the automated extraction and classification of functional requirements (FRs) and non-functional requirements (NFRs) in software development, leveraging supervised machine learning. The central concern arises from the fact that NFRs are often overlooked and identified late in the software development process, pointing towards the need for enhanced tools for early identification and management of NFRs.

The researchers developed and evaluated a supervised machine learning approach that uses metadata, lexical, and syntactical features. This approach was based on the Support Vector Machine (SVM) classifier algorithm. To deal with class imbalances in their dataset, the researchers employed under- and over-sampling strategies. They also augmented the dataset using user comments from software reviews, particularly for the binary classification cases.

For classification, they focused on two classes: non-functional requirements (NFR) and functional requirements (FR). These two classes were balanced by random under-sampling of the majority class. For individual NFR types, binary classifiers were used, while a multi-class classifier was employed for predicting the four major classes.

The performance of the classifiers was evaluated through cross-validation techniques, and they achieved impressive precision and recall rates. For the identification of FRs and NFRs, precision and recall reached up to approximately 92%. For specific NFRs like security and performance, they attained around 92% precision and 90% recall.

III. Collecting the New Dataset

In this section, we describe the process of compiling the dataset used in the research for software requirements classification using machine learning algorithms. The dataset plays a crucial role in training and evaluating the proposed classification model. We aimed to create a comprehensive and diverse dataset that accurately represents software requirements. To achieve this, we employed a multi-step approach, starting with an initial dataset and augmenting it with additional requirements from other sources.

A. Collecting dataset

In the initial step, we identified the initial dataset on GitHub, which was curated by Mitrevski (2021) [8] specifically for the task of requirements classification. This dataset encompassed a total of 626 requirements, which were subsequently classified into two distinct categories: functional requirements and non-functional requirements.

After that, We discovered a second dataset on Kaggle called "Software Requirements Dataset," curated by Iam Vaibhav [9]. This dataset aimed to classify requirements into multiple classes and consisted of approximately 976 requirements. We integrated this dataset with our primary dataset, focusing on classifying the requirements as either functional or non-functional. After conducting data re-processing, including the removal of duplicate requirements, our combined dataset now comprises around 1180 requirements.

Additionally, we conducted a thorough study of various projects and applications to gather a substantial number of requirements, striving to include the largest possible variety in our dataset. The analysis encompassed various research papers, resulting in the extraction of 70 non-functional requirements and 361 functional requirements Specifically, the

TABLE I
COMPARISON PREVIOUS INTELLIGENT MODELS FOR REQUIREMENTS CLASSIFICATION

Auther Name	Titel	Algorithm	Dataset	Result
Winkler, J.	Automatic classification of requirements based on convolutional neural networks.	Convolutional Neural Networks (CNN)	A dataset was created from the database of The DOORS document through the knowledge gained	A precision of 0.73 and a recall of 0.89.
Quba, G.	Software requirements classification using machine learning algorithm's	Support Vector Machine (SVM) and K-Nearest Neighbors (KNN)	The PROMISE_exp dataset	The use of Words (BoW) with the Support Vector Machine (SVM) algorithm outperformed BoW with KNN, achieving an F1 score of 90% in binary classification (FR or NFR), 66% across 11 NFR subcategories, and 72% across all 12 subcategories.
Jindal, R.	Automated Classification of Security Requirements	J48 decision tree algorithm	A publicly available dataset from the PROMISE software engineering repository	Model 4 performance (AUC=0.83, sensitivity=80%). Models 1 and 2 performance (AUC=0.72 and 0.77 respectively). Model 3 performance (AUC=0.69).
Sabir, M.	Multi-label classifier to deal with misclassification in non-functional requirements.	a Convolutional Neural Network (CNN) algorithm	Private SRS documents and manually extracted	There isn't a specific about the results of the proposed approach
Kumar, M.	Extraction and classification of non-functional requirements from text files: a supervised learning approach.	The methodology comprised four main steps: Systematic "lightweight" process visualization, Converting the SIG parser into CSV format, Merging keywords, and Quality assurance.	Utilized an Open Science Tera-PROMISE dataset	187 phrases related to security, performance, and usability.
Kurtanović, Z.	Automatically classifying functional and non-functional requirements using supervised machine learning.	Support Vector Machine (SVM)	"Quality attributes (NFR)" dataset	Precision and Recall 92

study proposed by [24] yielded 12 non-functional requirements and 28 functional requirements. Additionally, the study proposed by [23] yielded 12 non-functional requirements and 51 functional requirements. Moreover, [22] provided 10 non-functional requirements and 57 functional requirements, while [20], [21] contributed 15 non-functional requirements and 64 functional requirements. Furthermore, the works of [16]–[19] resulted in 21 non-functional requirements and 161 functional requirements.

B. The Pre-processing of Textual Data

Textual data is a valuable source of information in many domains, but its raw form is not directly compatible with machine learning algorithms. Therefore, pre-processing techniques play a crucial role in converting text into a suitable format for analysis. In our study, we focus on the pre-processing steps of tokenization, sequence padding, and TF-IDF vectorization to prepare the textual data for classification using the CNN, SVM, and KNN models [14].

1) Tokenization: Tokenization is the process of converting text into a sequence of tokens, where each token represents a word or a subword unit [14]. In our CNN model code, the Tokenizer class from the tensorflow.keras.preprocessing.text module is used for tokenization. It is initialized and then fitted on the input text data (X) using the fit_on_texts method. This step creates a vocabulary of unique words in the dataset and assigns a unique integer index to each word Figure 1.

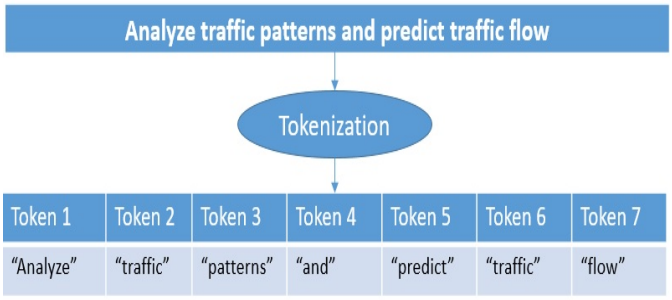
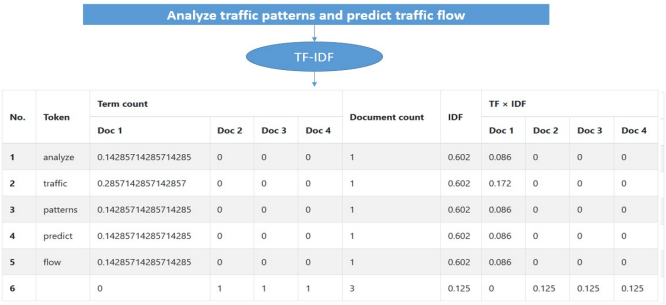


Fig. 1. Example for Tokenization from our requirements

2) Sequence Padding: Neural networks require input data of equal length, but the input text data may have varying

lengths [14]. To overcome this, sequence padding is applied to make all sequences of equal length. In our CNN model code, the pad_sequences function from the tensorflow.keras.preprocessing.sequence module is used for padding. The sequences obtained from tokenization are padded to a specified max_seq_length (which is set to 100 in this code) using the pad_sequences function. The padding is done by adding zeros to the end of each sequence to match the maximum sequence length.

3) The TF-IDF: The TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer is a commonly used technique for converting textual data into numerical features that can be used for machine learning algorithms [15]. It assigns a numerical value to each word or term in the text based on its frequency and importance in the document and across the entire dataset.We employ the TF-IDF vectorizer in both the SVM and KNN models to convert the textual data into numerical features2.



No.	Token	Term count				Document count	IDF	TF × IDF			
		Doc 1	Doc 2	Doc 3	Doc 4			Doc 1	Doc 2	Doc 3	Doc 4
1	analyze	0.14285714285714285	0	0	0	1	0.602	0.086	0	0	0
2	traffic	0.2857142857142857	0	0	0	1	0.602	0.172	0	0	0
3	patterns	0.14285714285714285	0	0	0	1	0.602	0.086	0	0	0
4	predict	0.14285714285714285	0	0	0	1	0.602	0.086	0	0	0
5	flow	0.14285714285714285	0	0	0	1	0.602	0.086	0	0	0
6		0	1	1	1	3	0.125	0	0.125	0.125	0.125

Fig. 2. Example for TF-IDF from our requirements

IV. Implementation and Results

In the implementation phase, we utilized Python to conduct the classification process employing machine learning techniques. We experimented with multiple algorithms, specifically Convolutional Neural Network (CNN), Support Vector Machine (SVM), and another instance of CNN, to achieve optimal results for the task at hand.

In evaluating the experiment, we employed various performance metrics, namely accuracy, F1 score, recall, and precision. These metrics enable a comprehensive analysis of the model's performance, capturing different aspects such as overall correctness, balance between precision and recall, and more. By utilizing these methodologies and metrics, we aimed to establish a reliable and effective classification process for software requirements.

A. CNN

During the Convolutional Neural Network (CNN), were experimented with to achieve the best possible outcomes. The model underwent training on 70% of the dataset, validation on 20%, and testing on 10%. To ensure uniformity, the dataset underwent preprocessing through tokenization and sequence padding. The CNN model was comprised of an embedding layer, a convolutional layer, a global max pooling layer, and dense layers. It was compiled using the Adam optimizer and trained for 10 epochs with a batch size of 32. The classification results are as follows:

TABLE II
The Results of CNN Model

Eval. Metrics	Trainaing	Validations	Testing
Accuracy	98%	88%	92%
Precision	98%	92%	94%
Recall	97%	92%	94%
F1- score	98%	92%	94%

From the Table II ,the CNN model achieved high accuracy, precision, recall, and F1-score across all evaluation metrics, demonstrating its effectiveness in classifying the data.

The Confusion matrices were generated for the training, validation, and testing datasets, which are shown in the Tables IIIIVV.

TABLE III
The Confusion Matrix of CNN training datasets

Trainaing Confusion Matrix		Actual	
		Functional	Non-functional
Predicted	Functional	758	19
	Non-functional	1	285

TABLE IV
The Confusion Matrix of CNN validation datasets

Validation Confusion Matrix		Actual	
		Functional	Non-functional
Predicted	Functional	194	18
	Non-functional	18	75

TABLE V
The Confusion Matrix of CNN testing datasets

Testing Confusion Matrix		Actual	
		Functional	Non-functional
Predicted	Functional	103	6
	Non-functional	6	36

Figure 3 depicts the performance of the CNN algorithm in terms of accuracy across different evaluation sets. The results indicate high accuracy in the training set (98%), followed by slightly lower accuracy in the validation set (88%) and the testing set (92%). Similarly, precision, recall, and F1-score exhibit consistent and favorable values across the three evaluation sets.

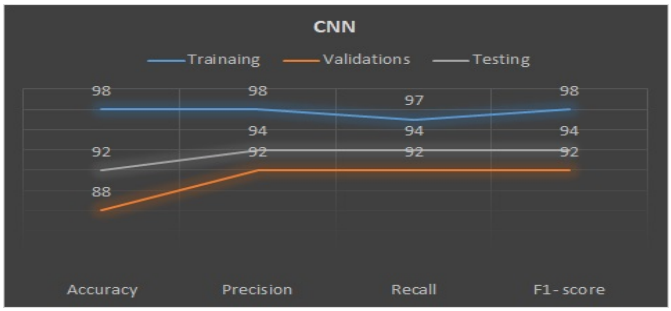


Fig. 3. The performance of the CNN algorithm

B. SVM

The Support Vector Machine (SVM) algorithm was employed for classification in this implementation. To convert the textual data into numerical features, the TF-IDF vectorizer was utilized. The dataset was then split into training, validation, and testing sets.

The SVM model, with a linear kernel, was trained on the training set. The trained model and vectorizer were saved for future use. Predictions were made on the training, validation, and testing sets. Evaluation metrics such as accuracy, precision, recall, and F1-score were calculated for each set. Additionally, a confusion matrix was generated for the test set.The classification results are as follows:

TABLE VI
THE RESULTS OF SVM MODEL

Eval. Metrics	Trainaing	Validations	Testing
Accuracy	96%	88%	88%
Precision	95%	88%	89%
Recall	99%	96%	95%
F1- score	97%	92%	92%

The SVM model, as shown in Table VI, achieved good accuracy, precision, recall, and F1-score across all evaluation metrics, indicating its effectiveness in classifying the data.

The Confusion matrices were generated for the training, validation, and testing datasets, which are shown in the Tables VIIVIIIIX.

Figure 4 depicts the performance of the SVM algorithm in terms of accuracy across different evaluation sets.The results indicate relatively high accuracy in the training set (96%),

TABLE VII
THE CONFUSION MATRIX OF SVM TRAINING DATASETS

Trainaing Confusion Matrix		Actual	
		Functional	Non-functional
Predicted	Functional	772	5
	Non-functional	42	244

TABLE VIII
THE CONFUSION MATRIX OF SVM VALIDATION DATASETS

Validation Confusion Matrix		Actual	
		Functional	Non-functional
Predicted	Functional	203	9
	Non-functional	24	66

TABLE IX
THE CONFUSION MATRIX OF SVM TESTING DATASETS

Testing Confusion Matrix		Actual	
		Functional	Non-functional
Predicted	Functional	104	5
	Non-functional	13	29

followed by slightly lower accuracy in the validation set (88%) and the testing set (88%). Precision, recall, and F1-score also exhibit consistent and favorable values across the three evaluation sets.

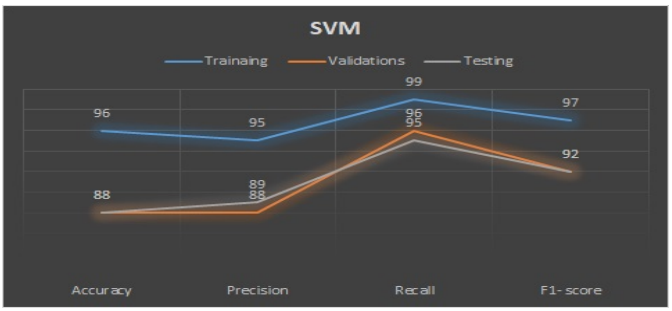


Fig. 4. The performance of the SVM algorithm

C. KNN

The K-Nearest Neighbors (KNN) algorithm is applied for classification. The textual data was transformed into numerical features using the TF-IDF vectorizer. Subsequently, the dataset was divided into training, validation, and testing sets. The

KNN model was trained on the training set, with the number of neighbors (k) set to 5.The trained model was then saved, along with the TF-IDF vectorizer, for future use. Predictions were made on the training, validation, and testing sets using the trained KNN model.The accuracy, precision, recall, and F1-score exhibited favorable values across the training, validation, and testing sets.The classification results are as follows:

TABLE X
THE RESULTS OF KNN MODEL

Eval. Metrics	Trainaing	Validations	Testing
Accuracy	90%	86%	85%
Precision	91%	87%	90%
Recall	95%	93%	90%
F1- score	93%	90%	90%

The KNN model performance was evaluated using various metrics, including accuracy, precision, recall, and F1-score. The results are summarized in Table X.

The Confusion matrices were generated for the training, validation, and testing datasets, which are shown in the Tables XIXIIXIII.

TABLE XI
THE CONFUSION MATRIX OF KNN TRAINING DATASETS

Trainaing Confusion Matrix		Actual	
		Functional	Non-functional
Predicted	Functional	742	35
	Non-functional	27	219

TABLE XII
THE CONFUSION MATRIX OF KNN VALIDATION DATASETS

Validation Confusion Matrix		Actual	
		Functional	Non-functional
Predicted	Functional	199	13
	Non-functional	29	64

Figure 6 presents a visual representation of the KNN algorithm's accuracy performance across different evaluation sets. The training set achieved the highest accuracy at 90%, followed by the validation set at 86% and the testing set at 85%. Similarly, precision, recall, and F1-score exhibit consistent and favorable values across the three evaluation sets.

TABLE XIII
THE CONFUSION MATRIX OF KNN TESTING DATASETS

Testing Confusion Matrix		Actual	
		Functional	Non-functional
Predicted	Functional	98	11
	Non-functional	11	31

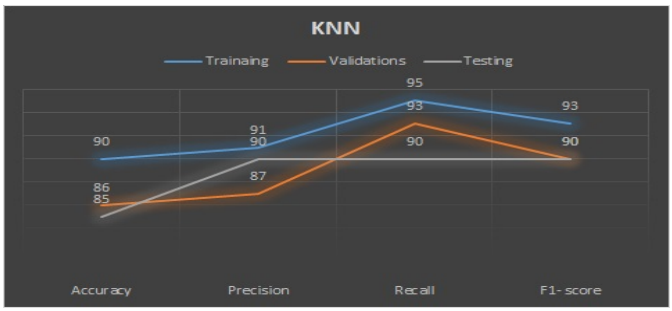


Fig. 5. The performance of the KNN algorithm

TABLE XIV
THE SUMMARY CNN, SVM, AND KNN RESULTS

Algorithem Model	Accuracy	Precision	Recall	F1- score
CNN	98%	98%	97%	98%
SVM	96%	95%	99%	97%
KNN	90%	91%	95%	93%

D. Summery

Table XIV presents a summary of the results obtained from the three models: CNN, SVM, and KNN. The CNN model achieved the highest accuracy of 98%, indicating its ability to classify the data with a high level of correctness. It also exhibited excellent precision and recall scores of 98% and 97% respectively, demonstrating its capability to accurately identify positive instances and minimize false positives. The F1-score, which combines precision and recall, was also impressive at 98% for the CNN model.

The SVM model also performed well, with an accuracy of 96%. It showed a slightly lower precision of 95% but compensated with an outstanding recall of 99%, indicating its effectiveness in correctly identifying positive instances. The F1-score for the SVM model was 97%, reflecting a good balance between precision and recall.

The KNN model achieved an accuracy of 90%, which is lower compared to the other two models. However, it still displayed respectable precision and recall scores of 91% and 95% respectively. The F1-score for the KNN model was 93%, indicating a relatively balanced performance between precision

and recall.

Overall, the CNN model demonstrated the highest accuracy and performed consistently well across all evaluation metrics. The SVM model excelled in recall, making it highly reliable for correctly identifying positive instances. The KNN model showed good performance but had slightly lower accuracy compared to the other two models. Figure **??** can provide a visual representation or diagram to further analyze and compare the performance of these models.

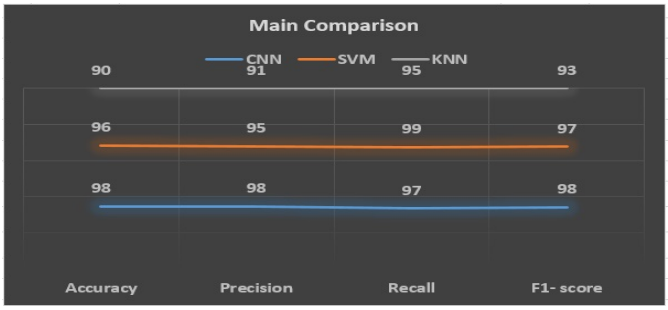


Fig. 6. The performance of CNN, SVM, and KNN algorithms

V. Conclusion

In conclusion, our study aimed to leverage artificial intelligence to enhance requirements engineering. We employed machine learning techniques to develop multiple classification models for distinguishing between functional and non-functional requirements. To accomplish this, we constructed a novel dataset by amalgamating existing datasets from sources such as Kaggle and GitHub. Additionally, we collected new requirement datasets from various projects and applications.

Throughout our research, we implemented three different models: CNN, SVM, and KNN. These models underwent rigorous evaluation and testing, yielding promising results. These results highlight the effectiveness of the CNN model in achieving remarkable accuracy (98%), along with balanced performance across precision (98%), recall (97%), and F1-score (98%) metrics. The SVM model also exhibits strong overall performance, particularly in terms of recall (99%), indicating its ability to correctly identify positive instances. The KNN model demonstrates a respectable level of accuracy (90%) and precision (91%), although its recall (95%) and F1-score (93%) are slightly lower compared to the other models.

By utilizing these classification models, we have made significant progress in automating the classification of requirements, facilitating more efficient and accurate requirement engineering processes. Our work demonstrates the potential of artificial intelligence and machine learning in improving the field of requirements engineering and opens avenues for further research and development in this domain.

References

[1] Winston, P. H. (1984). Artificial intelligence. Addison-Wesley Longman Publishing Co., Inc..

[2] Mitchell, T. M. (2007). Machine learning (Vol. 1). New York: McGraw-hill.

[3] Van Lamsweerde, A. (2000, June). Requirements engineering in the year 00: A research perspective. In Proceedings of the 22nd international conference on Software engineering (pp. 5-19).

[4] Nuseibeh, B., Easterbrook, S. (2000, May). Requirements engineering: a roadmap. In Proceedings of the Conference on the Future of Software Engineering (pp. 35-46).

[5] Ninaus, G., Reinfrank, F., Stettinger, M., Felfernig, A. (2014, August). Content-based recommendation techniques for requirements engineering. In 2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE) (pp. 27-34). IEEE.

[6] Winkler, J., Vogelsang, A. (2016, September). Automatic classification of requirements based on convolutional neural networks. In 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW) (pp. 3945). IEEE.

[7] Quba, G. Y., Al Qaisi, H., Althunibat, A., AlZu'bi, S. (2021, July). Software requirements classification using machine learning algorithm's. In 2021 International Conference on Information Technology (ICIT) (pp. 685-690). IEEE.

[8] Mitrevski, A. "SE Requirements Classification PROMISE_exp Dataset.", 2021, https://github.com/AleksandarMitrevski/se-requirements-classification.

[9] Vaibhav, Iam. "Software Requirements Dataset." Accessed May 16, 2023. https://www.kaggle.com/datasets/iamvaibhav100/software-requirements-dataset.

[10] Jindal, R., Malhotra, R., Jain, A. (2016, September). *Automated classification of security requirements.* In 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI) (pp. 2027-2033). IEEE.

[11] Sabir, M., Chrysoulas, C., Banissi, E. (2020). *Multi-label classifier to deal with misclassification in non-functional requirements.* In Trends and Innovations in Information Systems and Technologies: Volume 1 8 (pp. 486-493). Springer International Publishing.

[12] Kumar, M. S., Harika, A. (2020). *Extraction and classification of non-functional requirements from text files: a supervised learning approach.* Psychology and Education, 57(9), 4120-4123.

[13] Kurtanović, Z., Maalej, W. (2017, September). *Automatically classifying functional and non-functional requirements using supervised machine learning.* In 2017 IEEE 25th International Requirements Engineering Conference (RE) (pp. 490-495). Ieee.

[14] Chowdhury, R. R., Hossain, M. S., Hossain, S., Andersson, K. (2019, September). Analyzing sentiment of movie reviews in bangla by applying machine learning techniques. In 2019 International Conference on Bangla Speech and Language Processing (ICBSLP) (pp. 1-6). IEEE.

[15] Christian, H., Agus, M. P., Suhartono, D. (2016). Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF). ComTech: Computer, Mathematics and Engineering Applications, 7(4), 285-294.

[16] Aleko, D. R., and Djahel, S. (2020). *An efficient adaptive traffic light control system for urban road traffic congestion reduction in smart cities.* Information, 11(2), 119.

[17] Khalid, T., Khan, A. N., Ali, M., Adeel, A., ur Rehman Khan, A., & Shuja, J. (2019). *A fog-based security framework for intelligent traffic light control system.* Multimedia Tools and Applications, 78, 24595-24615.[utf8]inputenc

[18] Liu, J., Li, J., Zhang, L., Dai, F., Zhang, Y., Meng, X., Shen, J. (2018). *Secure intelligent traffic light control using fog computing*. Future Generation Computer Systems, 78, 817-824.

[19] Elsagheer Mohamed, S. A., & AlShalfan, K. A. (2021). *Intelligent traffic management system based on the internet of vehicles (IoV).* Journal of advanced transportation, 2021, 1-23.

[20] Habib, M. A., Ahmad, M., Jabbar, S., Khalid, S., Chaudhry, J., Saleem, K., ... Khalil, M. S. (2019). *Security and privacy based access control model for internet of connected vehicles.* Future Generation Computer Systems, 97, 687-696.

[21] Feng, Y., Huang, S., Chen, Q. A., Liu, H. X., Mao, Z. M. (2018). *Vulnerability of traffic control system under cyberattacks with falsified data.* Transportation research record, 2672(1), 1-11.

[22] M. Bani Younes and A. Boukerche, *Intelligent Traffic Light Controlling Algorithms Using Vehicular Networks*," IEEE Trans. Veh. Technol., vol. 65, no. 8, pp. 5887–5899, 2016, doi: 10.1109/TVT.2015.2472367.

[23] Hossan, S., & Nower, N. (2020). *Fog-based dynamic traffic light control system for improving public transport.*Public Transport, 12, 431-454.

[24] Younes, M. B., Boukerche, A. (2018). *An efficient dynamic traffic light scheduling algorithm considering emergency vehicles for intelligent transportation systems.* Wireless Networks, 24, 2451-2463.