Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/170620

Automated Classification of User Requirements written in Arabic using machine learning algorithms

Ahmad Althunibat[1], *Reem Amro*[1], *Belal Hawashin*[1], *Hend AlNuhait*[2], *Sally Almanasra*[2] *and Haneen A. Al-Khawaja*[3,4,*]

[1]Faculty of Sciences and Information Technology, Al-Zaytoonah University of Jordan, Amman, Jordan
[2]Faculty of Computer Studies, Arab Open University, Riyadh, Saudi Arabia
[3]Faculty of Business, Amman Arab University, Amman, Jordan
[4]Swiss FinTech Innovation Lab, University of Zurich, Switzerland

Received: 31 Aug. 2023, Revised: 18 Sep. 2023, Accepted: 7 Oct. 2023
Published online: 1 Nov. 2023

Abstract: Requirement engineering is a critical step in software engineering, influencing software development outcomes. The manual classification of software requirements into Functional Requirements and Non-Functional Requirements is a laborious and costly process with varying accuracy. Errors in classification can lead to misunderstandings and incomplete products. Arabic requirements pose additional challenges due to their structural and semantic characteristics, contributing to inherent ambiguity. In addition to the lack of Arabic Studies as well as the public datasets for requirements written in Arabic.This study proposes combining machine learning and deep learning algorithms, including K-nearest neighbors (KNN), Support Vector Machines (SVMs), Random Forest, XGBoost, and the Arabert model, with optimization techniques to streamline the categorization of Arabic requirements. Optimal configurations for several classifiers are identified and examined by stemming techniques. In addition, an Arabic data set for requirements was collected. The results demonstrate the effectiveness of the proposed approach, enhancing productivity and mitigating risks. The SVM classifier achieves an F1-Score of 0.93, while combining it with ISRIStemmer improves the score to 0.95. The Arabert model achieves the highest F1-Score of 0.97, highlighting its performance in classifying Arabic requirements accurately.

Keywords: Deep Learning, Functional Requirement, Machine Learning, Non-Functional Requirement, Arabic Requirements Classification.

1 Introduction

Software engineering is a systematic process comprising multiple phases, including requirements gathering, design, development, testing, and maintenance. This structured approach aims to create software that is dependable, of high quality, and easy to maintain [1, 2]. Central to this process are software requirements, a critical phase involving the identification, description, and categorization of two fundamental types of requirements: Functional Requirements (FRs) and Non-Functional Requirements (NFRs) [3, 4]. Functional Requirements (FRs) encompass the core services and functions that a software system must provide. They define what the system should do, specifying its intended behavior, operations, and features. FRs serve as the building blocks that outline the system's primary functionalities, such as user interactions, data processing, and system responses [5]. Non-Functional Requirements (NFRs), on the other hand, focus on attributes that shape how the system performs its functions. These attributes include aspects like performance, security, usability, and reliability. NFRs complement FRs by outlining the quality criteria that the system must meet to ensure its effectiveness, safety, and user satisfaction[4,5]. In the realm of software requirements engineering, classifying these FRs and NFRs can be a labor-intensive and costly task. Manual classification can introduce errors and inconsistencies, potentially leading to misunderstandings and incomplete product development [6]. While efforts have been made to classify English-written requirements [7] and [8], there remains a significant research gap in the classification of Arabic software requirements using supervised learning techniques. The Arabic language presents unique challenges for classification due to its structural, syntactic, and semantic complexities, making it inherently

* Corresponding author e-mail: h.alkhawaja@aau.edu.jo

more ambiguous than other languages [10]. Furthermore, the scarcity of resources and publicly available datasets for Arabic requirements hampers the development of accurate models capable of handling the language's nuances and variations. This study aims to address these challenges by introducing an automated classification approach based on supervised learning for Arabic-written software requirements. The approach seeks to categorize these requirements into the two vital categories of FRs and NFRs. To the best of current knowledge, no prior research has undertaken the classification of Arabic software requirements using supervised learning techniques. The primary hurdle faced in this endeavor is the lack of publicly available Arabic language datasets for software requirements classification. To overcome this challenge, the approach intends to construct a dataset by sourcing software requirements documents from diverse Arabic-speaking sources, including academic publications, industry standards, and online repositories. The dataset will be meticulously labeled by domain experts to ensure accuracy and consistency. The approach will then employ a range of supervised learning classifiers, such as K-nearest neighbors (KNN), Support Vector Machines (SVMs), Random Forest, XGBoost, and deep learning algorithms like the Arabert model, to perform the classification. The objective is to evaluate the performance of these classifiers and feature selection techniques, ultimately identifying the most efficient approach for classifying Arabic language requirements. This proposed method holds the potential to significantly enhance the precision and effectiveness of software requirements categorization in Arabic-speaking regions, ultimately leading to improved software development outcomes. The remainder of this paper is organized as follows: Section Two reviews the relevant literature, Section Three outlines the method, Section Four presents the experimental data and findings, Section Five discusses the results, and Section Six concludes the paper with recommendations for further research.

2 LITERATURE REVIEW

This section provides an overview and summary of past research on the classification of software requirements through the application of machine learning and deep learning algorithms. Specifically, this research has focused on distinguishing between Functional Requirements and Non-Functional Requirements.

English Requirements Classification

Dave and Anu [7] utilized machine learning algorithms to categorize software requirements into Functional Requirements, Non-Functional Requirements, or Non-Requirements. This was done by analyzing both Mobile App Reviews and formal Software Requirements Specification documents. By using Stochastic Gradient Descent (SGD), The Support Vector Machine (SVM), and Random Forest (RF) ML algorithms, combined with the term frequency-inverse document frequency (TF_IDF) natural language processing technique, developers can improve their application without wasting time and better respond to customer needs. Based on the results, SGD had the highest accuracy rate at 83% for identifying FRs, NFRs, and NRs. A recent study [11], created a method to classify software requirements using the BoW technique in conjunction with machine learning algorithms. The data used for this method was obtained from PROMISE_exp, and KNN and SVM were utilized to categorize requirements into two types: functional and non-functional. In addition, the NFR requirements were further classified into 11 sub-categories to address the issue of manual classification, which can be both costly and time-consuming. The results indicate that the SVM with BoW produces better outcomes than KNN algorithms, with an overall F-measure average of 0.74. Dave, et al [12] proposed three data models that utilize different machine learning algorithms, namely SVM, SGD, and RF, combined with the TF_IDF NLP technique. The aim was to classify FRs, NFRs, and their corresponding sub-categories from SRS documents containing formal software requirements in order to improve the requirements classification process and reduce manual effort. The results indicated that SVM with TF_IDF had the highest F1 score of 0.88 when identifying FRs. Meanwhile, SGD had the highest F1 score of 0.92 when identifying NFRs. Additionally, SVM with TF_IDF proved to be the best method for identifying NFR types related to Availability, Look & Feel, Maintainability, Operational, and Scalability. On the other hand, SGD with TF_IDF produced better results for NFR-types Security, Legal, and Usability. Khatian, et al [13] created a method to classify non-functional requirements using supervised machine learning algorithms. This was followed by a study comparing five different ML algorithms: random forest classifier (RFC), k-nearest neighbor (KNN), decision tree, and logistic regression (LR), as well as naïve Bayes. The performance of each algorithm was evaluated using four different metrics: recall, precision, confusion matrix, and accuracy. The results indicate that the logistic regression algorithm has the highest prediction rates and a 75% accuracy, making it the best algorithm among the five studied. Talele and Phalnikar [8] conducted a systematic review to examine the ML algorithms utilized in classifying software requirements as functional or non-functional. The goal was to identify the most effective algorithm for prioritizing and classifying requirements, as well as how to evaluate them. The results reveal that SVM and DT algorithms outperform other ML algorithms in terms of accuracy, with a 2% improvement. Additionally, the study found that Drank is the best algorithm for prioritizing requirements. Shah, et al [14] suggested a new method for

automatically categorizing software requirements using machine learning techniques such as Naïve Bayesian, Support Vector Machine, and Recurrent Neural Networks. This approach is combined with natural language processing using TF_IDF to distinguish functional requirements from non-functional requirements. The goal is to prevent the misclassification of software requirements, which can cause ambiguities and ultimately lead to bugs and failures. The algorithms employed in this study produced results with an accuracy rate of 91%, 90%, and 92%, respectively. Chatterjee, et al [15] conducted research to pinpoint and categorize the requirements that hold the most weight when it comes to designing the architecture of software systems. The research team utilized a Bidirectional Long Short-Term Memory Network (Bi-LSTM) to extract pertinent information from the Software Requirement Specification (SRS) document. Their process included gathering information from individual words within the SRS document, which was then combined for final classification. The outcome of the research revealed that the identification of Architecturally Significant Functional Requirements (ASFRs) had an f-score of 0.86, while classification had an f-score of 0.83. Haque, et al [16] conducted experimental studies aimed at identifying the optimal pair for Non-functional Requirements classification. We used seven machine learning algorithms (GNB, BNB, KNN, SVM, SGD SVM, and DTree) and four feature extraction methods (BoW, TF_IDF (3)) to assist developers in creating high-quality software. Our findings indicate that the SGD, SVM classifier achieved the highest accuracy score of 0.76. Additionally, the TF_IDF feature extraction technique outperformed the other methods. Li, et al. [17] conducted an experiment to determine the accuracy of classifying requirements into functional and non-functional categories, the machine learning algorithms of gradient boosting and random forest were compared. The results indicated that the gradient boosting algorithm was more effective in accurately classifying non-functional requirements. Binkhonain and Zhao [18] conducted an investigation to classify NFR using several machine-learning algorithms and four feature selection techniques. The findings indicated that the combination of TF_IDF for feature extraction and the SGD or SVM algorithm resulted in the most precise NFR classification. Baker, et al [19] conducted a study using a combination of fully connected artificial neural networks (ANNs) and convolutional neural networks (CNNs), to classify non-functional requirements (NFRs). The researchers utilized random vectors to represent requirement sentences as input for CNN, focusing on five NFR categories: operability, performance, security, maintainability, and usability. They conducted their experiments using the PROMISE dataset, which included 1165 NFRs that spanned over ten categories. The experiment was divided into five stages, including data pre-processing, ANN model construction, CNN model construction, and evaluation. The study's evaluation results revealed a precision level ranging from 82% to 90% and recall ranging from 78% to 85% in the ANN model. In contrast, the CNN model achieved a precision level ranging from 82% to 94% and recall ranging from 76% to 97%, resulting in a high F-score of 92%. Wang, et al [20] conducted a study to analyze how app change logs impact the automatic categorization of requirements from app reviews using supervised machine learning algorithms, including Naïve Bayes, Bagging, J48, and KNN. The accuracy of requirements classification was compared across the four algorithms, and it was found that Naïve Bayes performed the best for categorizing app reviews. Lu and Liang [21] developed an approach using machine learning algorithms such as Naive Bayes, J48, and Bagging, combined with four classification techniques including BoW, TF_IDF, CHI2, and AUR-BoW, to automatically categorize user reviews into four types of NFRs (reliability, usability, portability, and performance), FRs, and Others. This approach helps developers gain a better understanding of the user's needs and wants. The results of this approach have shown an F-measure of 71.8% through the combination of AUR-BoW with Bagging.

Arabic Requirements Classification

Alzanin, et al [22] presented a system for classifying Arabic tweets into five separate categories based on their linguistic traits and content. The authors looked at two different textual representations: stemmed text using the term frequency-inverse document frequency (TF_IDF) and word embedding using Word2vec. Support Vector Machine (SVM), Gaussian Naive Bayes (GNB), and Random Forest (RF) were the three classifiers investigated in the study, each with its own set of hyperparameters. The scientists manually annotated a dataset of about 35,600 Arabic tweets for the experiments. With stemming and TF_IDF, the performance of the RF and SVM classifiers with radial basis function (RBF) kernel was comparable, resulting in macro-scores between 98.09% and 98.14%. However, the performance of GNB with word embedding was disappointingly poor. Notably, the outcomes outperformed the current benchmark score of 92.95% obtained using a deep learning algorithm specifically RNN-GRU (recurrent neural network-gated recurrent unit). Al-Hagree and Al-Gaphari [23] analyzed user feedback on mobile banking services applications and used that data to inform future improvements and fixes. To achieve this goal, a dataset consisting of comments made by Google Play Store users about financial mobile apps was compiled. The researchers annotated the reviews by hand, assigning ratings of favorable, negative, and neutral to each. Naive Bayes (NB), K-nearest neighbor (KNN), Decision Tree (DT), and Support Vector Machine (SVM) models were used for Arabic sentiment analysis, all of which are machine learning approaches.

The results of the evaluation showed that the NB model was superior to the DT, KNN, and SVM algorithms. The NB model outperformed all others in terms of sentiment analysis for mobile banking services with respect to accuracy (89.65%), recall (88.08%), precision (88.25%), and F-score (88.25%). This study demonstrates how machine learning methods may be applied successfully to the problem of mobile banking service user sentiment analysis. Atwan, et al [24] evaluated the effectiveness, of three widely used classification algorithms - K-nearest neighbor, Naive Bayes, and decision tree - were employed to classify Arabic text, both with and without the use of a light stemmer in the preprocessing phase. The experiment involved classifying a dataset from Agency France Persse (AFP) Arabic Newswire 2001, which contained 800 files categorized into four groups. The results revealed that the decision tree, in conjunction with the light stemmer, achieved the highest accuracy rate of 93% among the three classifiers. The study's primary objective was to simplify the process and reduce costs for users and developers seeking and utilizing such data. Ibrahim, et al [25] proposed standard scraping techniques, they collected a dataset from various sources and classified Arabic short texts into specific categories based on their titles. The texts were evaluated using three common Naive Bayes classifiers: Multinomial Naive Bayes, Complemented Naive Bayes, and Gaussian Naive Bayes. The data underwent several preprocessing steps, such as removing punctuation and stop words, as well as space vectorization. During the testing phase, the Complemented Naive Bayes Classifier demonstrated the highest accuracy of 0.84 for feature extraction. These results suggest that this method could potentially be applied to various brief text classification applications. Al-Tamimi, et al [26] made a demonstration of how active learning can enhance an AI agent's proficiency in text classification of Arabic news articles. The findings indicate that active learning outperforms passive learning by achieving the desired classification accuracy with minimal data processing. The study employed three different iterations of the NADA dataset. The importance of using augmentation techniques to increase the sample size of under-represented categories in unbalanced datasets was emphasized to improve the system's overall accuracy. Furthermore, active learning enables the integration of machine learning into more complex Arabic language challenges by reducing the time and cost involved in training AI agents on complicated datasets. Shehadeh, et al [9] created Natural language processing (NLP) technologies to provide a semi-automated approach to categorizing Arabic software requirements as either functional or non-functional. The proposed method involved extracting data from the software requirements and utilizing a set of heuristics that relied on fundamental Arabic language constructions to carry out the classification task. The researchers utilized their findings to assist software engineers in manually classifying software needs with less effort and cost. the results show that the average recall of functional is equal to 83.33% while the average recall of non-functional is equal to 93.33%. Al-Smadi, et al [27] presented two advanced neural networks that utilize long short-term memory (LSTM) to introduce aspect-based sentiment analysis of reviews for hotels in the Arabic language. The first model, a bidirectional LSTM with a conditional random field classifier (Bi-LSTM-CRF), was used to extract aspect opinion target expressions (OTEs), while the second model, an aspect-based LSTM, was used to determine aspect sentiment polarity. The aspect-OTEs were treated as attention expressions to aid in identifying sentiment polarity. The models were assessed using a benchmark dataset of reviews for Arabic hotels, with the first model achieving an F1 Score of 69.98% and the second model achieving an accuracy of 82.6%. Dahou, et al [28] developed an innovative technique for categorizing Arabic sentences utilizing the differential evolution algorithm and the convolutional neural network. This novel approach incorporates the CNN architecture and network parameters to automatically search for the optimal configuration using the DE algorithm. The proposed method, known as DE-CNN, takes into account five CNN parameters and was tested on five Arabic sentiment datasets. The performance of DE-CNN was evaluated and compared to other advanced algorithms. The experimental results indicate that DE-CNN achieved a high accuracy rate of 93.28% and was faster and more precise than the state-of-the-art algorithms. Hmeidi, et al [29] performed a study to compare the classification of Arabic text. The researchers utilized a dataset consisting of 2700 Arabic articles, each corresponding to a distinct category. To prepare the texts for analysis, the authors implemented stemming and cleaning techniques and then employed five common text classification methods. The findings of the study revealed that the SVM classifier outperformed the other classifiers that were evaluated. As far as we know, there is a notable absence in the literature regarding the classification of Arabic requirements. The literature on the classification of Arabic requirements has a few drawbacks that have hindered a satisfactory resolution to the problem. To begin with, there is not much research that concentrates particularly on Arabic needs, which limits our knowledge of the specific challenges and characteristics of Arabic language requirements. The shortage of research has restricted the creation of effective and specialized systems to accurately categorize them. Additionally, the present research often employs techniques and strategies developed for languages other than Arabic, neglecting to consider the language's cultural and linguistic nuances. As a result of these oversights, inadequate performance, and inaccurate classification results ensue. Due to the absence of comprehensive and standardized datasets required for Arabic, it is challenging to evaluate and compare different classification algorithms effectively. These shortcomings underscore the need for more concentrated research initiatives aimed at tackling the distinct challenges of

Arabic requirement categorization and developing dependable models tailored to the Arabic language.

3 METHODOLOGY

In this study, we propose an automatic classification method for software requirements written in Arabic to solve the limitations highlighted in the literature review section. Our method divides requirements into functional and non-functional categories using supervised learning techniques. To achieve this, we formulated a comprehensive methodology consisting of five steps for both machine learning and deep learning to attain the goal. The proposed automated classification method for software requirements in Arabic combines the strength of supervised learning techniques with the richness of Arabic language processing. We want to use machine learning methods to create a robust and accurate model that can successfully classify requirements into functional and non-functional categories. This method is especially useful in the software development process because understanding the nature of requirements is critical for developing and implementing software systems. We want to increase the efficiency and accuracy of requirement analysis in Arabic-speaking situations by using this method. Moreover, by combining deep learning techniques into our methodology, we hope to capitalize on neural networks' innate capacity to capture complex patterns and correlations within the Arabic text. AraBERT, a deep learning model, has shown excellent performance in natural language processing challenges [30]. We intend to attain even greater accuracy and generalization skills in categorizing functional and non-functional requirements by constructing a suitable architecture on a large-scale dataset of Arabic software requirements. The use of deep learning techniques in our methodology offers another level of sophistication and the possibility of increased performance in automated requirement classification. In this study, the following algorithms were employed: Algorithm 1: Training classifier for Arabic Functional and Non-Functional Requirements and Algorithm 2: Classification of Arabic Requirements using classifier.

Table 1 Algorithm 1 Training classifier for Arabic Functional and Non-Functional Requirements

	Algorithm 1
01	Input: Dataset, Classifier
02	Output: Trained Model
03	Data pre-processing to begin cleaning the requirements.
04	Tokenize text using 'word_tokenize function from the Python nltk library.
05	Stopwords removal using 'stopwords.words('*arabic*')' function
06	Normalization to transform the text into a standardized format.
07	Apply count vectorizer and TF_IDF transformer
08	Apply feature selection using the chi-square
09	Apply Scaling using Min-Max scaling method
10	Train Classifiers using the preprocessed dataset and different parameters.

The algorithm starts by preprocessing the Dataset, which involves tokenizing the text in the Dataset, removing stopwords, and applying normalization techniques. After that, the data is transformed using a count vectorizer and a TF_IDF transformer to convert the tokenized text into numerical features. Relevant features are then selected through feature selection, and scaling is applied to ensure consistent feature values. The model training phase begins by specifying the classifier and different parameters. The algorithm outputs the Trained Model, which is now ready for further evaluation and testing.

Table 2 Algorithm 2 Classification of Arabic Requirements using classifier

	Algorithm 2
01	Input: Trained model and Testing Arabic Requirements.
02	Output: Classification of Arabic Requirements.
03	Pre-processing the testing Arabic requirements.
04	Tokenize text using 'word_tokenize' function from the Python nltk library.
05	Stopwords removal using 'stopwords.words('*arabic*')' function.
06	Normalization to transform the text into a standardized format.
07	Apply count vectorizer and TF_IDF transformer.
08	Apply feature selection using the chi-square.
09	Apply Scaling using Min-Max scaling method.
10	Apply evaluation metrics such as precision, recall, F1-score.

Upon completion of the training process, the algorithm produces a trained model that is ready for further evaluation and testing. The algorithm for the classification of Arabic requirements using a classifier begins with preprocessing the testing of Arabic requirements. The Arabic requirements designated for testing are subjected to tokenization, which involves breaking them down into individual words. This is followed by the removal of stopwords, or common words that do not carry significant meaning, and normalization to ensure consistent representation. Once the preprocessing is completed, the data undergoes transformation using a count vectorizer and TF_IDF transformer. These techniques capture the importance and frequency of words within the requirements. Furthermore, feature selection and scaling techniques are applied to choose relevant features and maintain consistent values. The trained model is then employed to predict the preprocessed testing Arabic requirements. Lastly, the algorithm assesses the accuracy of its predictions by comparing them with the actual labels and utilizing evaluation metrics such as precision, recall, and F1-score to measure the model's performance. The methodology used to achieve this goal is described below. Figure 1 shows the steps

Step1: Data Gathering

The initial phase of our methodology involved an extensive and meticulous effort to gather a diverse and comprehensive dataset of Arabic software requirements.

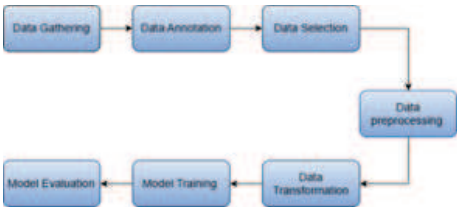

Fig. 1 The proposed Methodology.

Our approach was designed to encompass a broad spectrum of requirements from various sources and domains, ensuring the dataset's representativeness and relevance to real-world scenarios. To achieve this, we adopted a multifaceted strategy for data collection:

- –Industry Standards: We explored industry-specific standards and guidelines, a valuable source of requirements documentation. These standards provide detailed insights into domain-specific software requirements, contributing to the dataset's richness and applicability.
- –Academic Publications: A thorough review of academic papers and journals in the fields of software engineering and requirements engineering was conducted. These publications often contain well-documented software requirements, making them an invaluable source of data.
- –Online Repositories: Recognizing the prevalence of software projects hosted on online platforms, we systematically searched repositories such as GitHub and SourceForge. These platforms host a multitude of projects with publicly available requirements documents, offering a wealth of real-world data. Our overarching objective throughout the data collection process was to capture the heterogeneity and complexity inherent in Arabic-written software requirements. By sourcing requirements from diverse domains and a variety of reputable platforms, we aimed to construct a dataset that reflects the multifaceted nature of software requirements in Arabic-speaking contexts. Following the compilation of the dataset, a stringent validation process was initiated. This critical step involved comprehensive checks to ensure the accuracy and reliability of the collected data. Our validation procedures were designed to identify and rectify any discrepancies, errors, or inconsistencies within the dataset.

This validation phase served as a safeguard to maintain data integrity and quality. It underscored our commitment to delivering a meticulously curated dataset that serves as the cornerstone of our research. The data collection process encountered several challenges and limitations. These included the limited availability of Arabic software requirements documents, potential variations in quality and consistency among collected requirements, concerns about domain representation balance, the inherent linguistic complexity of Arabic, potential privacy and copyright restrictions on certain sources, the possibility of data bias favoring specific domains or regions, the need for domain expertise in the annotation process, and the resource-intensive nature of data collection and validation. Despite these constraints, our methodology aimed to address these challenges to create a robust and representative dataset for our research, recognizing the importance of data quality and diversity in training effective machine learning models for Arabic software requirements classification. In summary, Step 1 was marked by a meticulous and exhaustive effort to amass a diverse and representative dataset of Arabic software requirements. This dataset, drawn from a variety of trusted sources, is not only extensive but also subject to rigorous validation to uphold the highest standards of data quality and reliability.

Step2: Data Annotation

The dataset underwent a meticulous annotation process carried out by three domain experts. These specialists have a thorough understanding of software engineering principles, which enables them to distinguish between functional and non-functional aspects of the requirements and to classify them. The domain specialists carefully examined each requirement, examining its elements and labeling them according to their knowledge and experience. The participation of domain specialists in a certain field during the annotation procedure is of utmost importance in guaranteeing the precision and dependability of the dataset. Their extensive comprehension of software necessities and familiarity with the Arabic dialect empower them to make knowledgeable judgments while classifying the requirements. Their proficiency assists in surmounting obstacles, such as uncertainty or slight variations that might occur during the annotation process.

Step3: Data Selection

A subset of the classified data was selected for further processing based on the availability of labeled data.

Step4: Data Preprocessing

This study uploaded the dataset in Text format to Python language, followed by a series of steps to begin cleaning the requirements. To prepare the data for machine learning algorithms, a set of preprocessing steps were performed. These steps encompassed tokenization, stop word elimination, and normalization, all of which were vital in converting the raw data into a more sophisticated and organized structure.

–Tokenization

Tokenization is an essential initial step in text analysis and natural language processing. Its main objective is to break down the textual data into individual words or tokens, which is critical for a more precise analysis of the requirements [31]. Through the segmentation of the text into tokens, we can scrutinize the connections between words and capture the specific patterns that contribute to the meaning and context of the requirements. The text undergoes tokenization by following predefined rules or algorithms, which typically involve dividing the text using whitespace or punctuation marks. A common illustration of this process is breaking down a sentence such as يجب أن يكون للنظام واجهة سهلة الاستخدام would be tokenized into the following individual tokens: يجب, أن, يكون, للنظام, واجهة, سهلة, الاستخدام. Each token is a distinct unit of meaning that may be analyzed and processed further. Tokenizing the phrase allows us to examine the words' distinct meanings. For example, يجب translates to "should," أن translates to "that," يكون translates to "be," للنظام translates to "for the system," واجهة translates to "interface," سهلة translates to "easy," and الاستخدام translates to "to use." The Arabic software requirements were tokenized in this study by utilizing the 'word_tokenize()' function from the Python nltk library.

–Stopword removal

Stopword removal is an essential step in the preprocessing phase of the Arabic language. Stop words are common words that have no important meaning and do not contribute to the categorization process. Examples of stop words in Arabic include في (in), على (on), من (from), إلى (to), and ثم (then). These words are common in the language but give no useful information for classification tasks. Eliminating stop words from the data reduces text dimensionality and highlights informative content words. This method enhances the accuracy and efficiency of subsequent analysis and machine learning models. Removing these irrelevant words enables the models to capture vital features and patterns that differentiate between various requirements [32]. For example, consider the sentence يجب أن يكون للنظام واجهة سهلة الاستخدام (The system should have a user-friendly interface). After stopword removal, the sentence would be simplified to يجب يكون نظام واجهة سهلة الاستخدام (system should have user-friendly interface). The removal of stop words eliminates words like أن (that), لـل (for the), and ال (the), It does not contribute significantly to the meaning of the text. Eliminating stop words allows us to concentrate on the words that hold greater semantic significance and are essential for classification purposes. This simplified version of the text amplifies the efficiency of subsequent analysis and empowers the machine learning models to capture the relevant information for correct classification more accurately. In this study, the Arabic stopwords list provided by the nltk library was utilized. Specifically, the 'stopwords.words('arabic')' function was used to obtain a set of Arabic Stopwords.

–Normalization

Normalization is critical in transforming the text into a standardized format. Several procedures are performed in this stage to improve the efficacy of the machine learning algorithms in the classification task. Firstly, diacritics like accents and vowel signs are removed from the text. Diacritics are employed in Arabic to signify short vowels and pronunciation, although they do not contain important information for text classification. Removing diacritics simplifies the text and minimizes the data's dimensionality.

Secondly, Punctuation signs such as commas, periods, and question marks are removed. These symbols are useless for classifying software requirements and can create noise or discrepancies in the data. We guarantee that the attention is only on relevant content words by deleting punctuation marks. Furthermore, deleted are non-alphanumeric characters like emoticons or other symbols. These characters might affect learning since they do not contribute to the text's semantic meaning. By getting rid of them, we provide a clearer and more accurate representation of the requirements [33].

Additionally, normalization includes translating letters to their basic forms while accounting for changes in spelling and morphology. Words in Arabic can have several forms due to prefixes, suffixes, and other grammatical structures. We unify words with comparable meanings and minimize the vocabulary size by reducing letters to their basic forms, which

increases the learning process for machine learning algorithms [34]. For example, the word سهلة (easy) may appear in different forms such as سهلة (feminine singular) and سهل (masculine singular). Through normalization, these variations are reduced to their base form, سهل (easy), allowing the algorithms to consistently handle them.

Normalization techniques are applied to standardize and maintain consistency in text data for efficient machine-learning algorithm processing. This results in improved accuracy and effectiveness of subsequent classification procedures. Treating similar words and forms equivalently through normalization leads to more reliable and robust outcomes.

–Stemming

Stemming plays a critical role in information retrieval and natural language processing. Applying stemming to Arabic text requires the removal of prefixes, suffixes, and other affixes from words to extract their core or stem. This consolidation of related words with comparable meanings through the process reduces data complexity, enabling machine learning algorithms to easily classify and process the text. Arabic has multiple stemming algorithms at its disposal, including ISRIStemmer and ArabicLightStemmer. These algorithms are crafted to manage Arabic's distinct features and deliver efficient stemming abilities [35]. We want to minimize the variances in Arabic words and return them to their stem forms by using these stemming techniques. This approach assists in the simplification of text data and the capture of the basic semantic meaning of words while ignoring prefixes, suffixes, and other word variants. These stemming techniques were chosen because of their demonstrated performance and established reputation in the field of Arabic text processing. We guarantee that our preparation pipeline effectively handles the linguistic intricacies of the Arabic language by adding these algorithms into our technique, allowing for enhanced text categorization and analysis jobs. Here are some examples of how stemming is performed in Arabic text using the ISRI Stemmer: Original Text: الكتب، كتب، كتاب، الكتابة. Stemmed Text: كتب، كتاب، الكتاب. In this example, the original words الكتب (books), الكتاب (book), and الكتابة (writing) are stemmed from their root form كتاب (book). The stemming process removes the definite article ال as well as other word variants, yielding the common root word.

Step5: Data Transformation

Once the preprocessing phase is finished, the textual data that has been processed goes through several steps of data transformation to get ready for text classification. The succeeding sections detail the implementation of techniques such as TF_IDF vectorization, feature selection, and scaling to achieve this.

–TF_IDF Vectorization

We use TF_IDF vectorization to convert processed textual data into a numerical representation. Each requirement in the dataset is transformed into a vector, where the elements correspond to the TF_IDF values of specific words [36]. For instance, the requirement sentence يجب أن يحتوي النظام على واجهة سهلة الاستخدام للمستخدم (The system should have a user-friendly interface) would be represented as a vector containing TF_IDF values for the words النظام (system), سهلة الاستخدام (user-friendly), and واجهة (interface). This process enables us to capture the relative importance of words in the entire dataset and encode textual information in a suitable format for further analysis. The TF_IDF score is calculated by multiplying the TF value by the IDF value. The TF_IDF equation (1) [37]

$$TF\_IDF = TF(term, document) \times IDF(term) \quad (1)$$

where: TF(term, document) refers to the number of times a specific term appears in a document. This metric measures the term frequency within the document, and IDF(term) represents the inverse document frequency of the term. It assesses the rarity and significance of the term across the entire collection of documents. The purpose of this score is

to identify terms that are both frequent within a document and uncommon in the overall collection, indicating their importance in describing the content of the document.

–Feature Selection

Once the TF_IDF vectorization is complete, we proceed with feature selection to decrease the data's dimensionality and concentrate on the most valuable features. This stage entails selecting the leading characteristics based on their relevance to the classification task. By assessing the significance of each feature, we can prioritize the most distinguishing words that substantially contribute to differentiating between functional and non-functional requirements. This feature selection procedure guarantees that the following classification models are trained on the most pertinent and useful features. The objective of this study is to identify the most important features from the TF_IDF matrix through the process of feature selection. This is achieved by using the chi-square χ^2 test as the scoring function in Python. The chi-square test measures the statistical dependence between each feature and the target variable [38], allowing for the identification of the attributes that have the strongest relationship with the classification task.

–Scaling

To complete the data transformation process, it is crucial to scale the features. This ensures that they all have an equal impact on the machine-learning algorithms. During this step, we use the scaling method (the Min-Max scaling method). This scaling method, implemented through the MinMaxScaler function, ensures that the features are transformed to a common scale, typically ranging from 0 to 1, to bring all the features to a common range. By doing so, any potential bias caused by differences in magnitude or units of the features is eliminated. This process enables fair and accurate comparisons between the features and enhances the performance of the machine learning models when classifying text. Applying Min-Max scaling enables the normalization of feature values, thereby maintaining their relative relationships and placing them within a consistent range. This process is especially crucial when dealing with features that possess varying scales or ranges. By doing so, it guarantees that all features contribute proportionally to both the analysis and subsequent modeling tasks [39].

Step6: Model Training

–Training using machine learning

This step involves the training of machine learning models for text classification using preprocessed datasets. To ensure dependable performance evaluation and the selection of the best classifier for our task, cross-validation with k-folds was used. The dataset is equally divided into five parts, with each part serving as the testing set while the remaining four parts function as the training set. The succeeding paragraphs will present an overview of the classifiers utilized, each with distinct characteristics and advantages that enable them to capture varying patterns and relationships in the text datasets.

–Support Vector Machines (SVM)

Text classification tasks often use SVM, a potent algorithm. SVM maps input data onto a higher-dimensional feature space and then finds the optimal hyperplane that separates the various classes [40]. SVM is renowned for its ability to manage high-dimensional data and handle complex decision boundaries with effectiveness.

–Stochastic Gradient Descent (SGD)

SGD has been effectively utilized for addressing machine learning problems that are both large-scale and sparse, as frequently encountered in natural language processing and text classification. Due to the sparsity of the data, SGD is renowned for its efficiency and rapid convergence [41].

–Random Forest (RF)

Random Forest is a method used in ensemble learning, which combines several decision trees. These trees are trained on different subsets of data, and the final classification is based on the combined predictions of all the trees. Random Forest has earned a reputation for being robust, able to handle high-dimensional data, and preventing overfitting [42].

–K-Nearest Neighbors (KNN)

KNN is a classification algorithm that doesn't rely on fixed parameters. Instead, it classifies data points by comparing their distance to other data points within the feature space. When a new sample is added, KNN assigns it a label based on the majority class among its k closest neighbors [43]. This approach is particularly useful when dealing with non-linear boundaries, and the algorithm is both straightforward to implement and effective.

–Extreme Gradient Boosting (XGBoost)

The XGBoost algorithm has become popular due to its excellent performance in a variety of machine learning tasks. It achieves high predictive accuracy by utilizing both decision trees and gradient boosting techniques. XGBoost is highly regarded for its scalability, speed, and proficiency in handling intricate feature interactions [44].

–Logistic Regression (LR)

Logistic Regression is a popular binary classification approach. It stimulates the link between the input features and the likelihood of belonging to a particular class. Logistic Regression is well-known for its ease of use, interpretability, and resilience when dealing with noise and outliers [45].

–Decision Trees (DT)
Decision Trees are hierarchical structures that divide data into multiple feature values to make sequential judgments define core nodes as decisions and leaf nodes as class labels. Decision trees are simple to learn and analyze, and they can handle numerical as well as categorical data [46].
–Training using Deep learning
Our focus in this stage is to utilize a pre-existing deep learning model called Arabert for text classification. Arabert is a model that was specifically designed to handle Arabic text and has already undergone extensive training on a large corpus. This allows it to effectively capture the unique nuances and characteristics of the Arabic language. By harnessing the power of Arabert, we can achieve accurate text classification without investing significant resources into training from scratch.
–AraBERT model
Utilizing the Transformer architecture, the AraBERT model has shown great success in numerous natural language processing endeavors. By implementing self-attention mechanisms, it can effectively comprehend the dependencies and contextual nuances present in the input text. With its advanced understanding of Arabic language syntax, semantics, and subtle intricacies, AraBERT stands as a prime option for Arabic text classification. Utilizing the AraBERT model in Arabic text classification presents numerous benefits. This includes the ability to capture specific linguistic features unique to the Arabic language, manage contextual dependencies, and achieve high accuracy in classification [30]. By incorporating AraBERT into our classification task, we can harness its pre-trained knowledge and proficiency in Arabic language processing, ultimately enhancing the precision and effectiveness of our classification methodology.

Step7: Model Evaluation

Once the machine learning models and the Arabert model have been trained using preprocessed data, evaluating their performance becomes crucial in determining their effectiveness in Arabic text classification. This involves calculating significant evaluation metrics such as macro-recall, macro-precision, and macro-F1 scores to accurately measure the model's accuracy and overall effectiveness.

4 EXPERIMENTS AND RESULTS

This section outlines the experiments that were performed to measure the effectiveness of our automated classification method for software requirements in Arabic. Our main aim was to evaluate the accuracy of supervised learning techniques in categorizing requirements as either functional or non-functional.

Dataset

The study utilized a dataset comprising 400 requirements, which were evenly divided between functional and non-functional categories. Each group contained 200 requirements, ensuring a balanced distribution of data. This balance was crucial to prevent negative impacts on machine learning models caused by imbalanced data. The distribution of Arabic requirements into the functional and non-functional categories is outlined in TABLE 3 where we show the distribution of ARABIC requirements across the functional and non-functional categories, highlighting the equal split.

Table 3 ARABIC Requirements distribution

Algorithm 2	
Category	Count
Functional Requirements (FRs)	200
Non-Functional Requirements (NFRs)	200

Evaluation measurements

In this study, the efficacy of the suggested approach for the classification of software requirements was evaluated using three metrics that are quite common in the field of evaluation: recall, precision, and the F1 score. These indicators offer extremely helpful insights into the effectiveness and precision of the classification models [47]. While implementing k-fold cross-validation, we employed the macro-average of Recall, Precision, and F1 score as the metric for evaluation. This technique computes the metric for each fold individually and then calculates the mean value. Adopting the macro-average method assists in addressing any possible imbalance in classes and offers an extensive evaluation of the model's efficiency across diverse folds of the dataset. We utilized macro Recall to evaluate the ability of the model to accurately identify positive instances (true positives) in all categories. This approach gave us a comprehensive evaluation of the model's capacity to capture pertinent instances from every category, regardless of their sizes or frequencies. Macro Recall is computed as the mean of Recall values across all

classes. Macro Recall equation (2).

$$MacroRecall = (Recall\_class1 + Recall\_class2 + \cdots + Recall\_classN)/N \tag{2}$$

Similarly, we utilized macro Precision to evaluate the accuracy of the model in detecting positive instances across all categories. This approach enabled us to analyze how well the model reduced false positive predictions while giving equal consideration to all classes. Macro Precision is obtained by averaging the Precision values for all classes. Macro Precision equation (3).

$$MacroPrecision = (Precision\_class1 + Precision\_class2 + \cdots + Precision_classN)/N \tag{3}$$

Lastly, the model's performance was assessed using the macro F1 score, which provided a fair evaluation by factoring in both Recall and Precision. This score determined the harmonic mean of Recall and Precision for every class, thus delivering a comprehensive overview of the model's ability to accurately classify instances across all categories. The calculation of the Macro F1 score entailed averaging the F1 scores across all classes. Macro F1 score equation (4).

$$MacroF1score = (F1\_score\_class1 + F1\_score\_class2 + \cdots + F1\_score\_classN)/N \tag{4}$$

In the formulas presented above, "N" stands for the total number of classes involved in the classification issue, while "Recall_class," "Precision_class," and "F1_score_class" are the individual values assigned to each class [48].

Experimental settings

The experimental settings used in this study consisted of the software and hardware configurations. The experiments were conducted using Google Colab, a website that offers unrestricted use of GPU and TPU computing resources for the purpose of conducting machine learning experiments [49]. The experiments were implemented using Python programming language, and several libraries were used, including scikit-learn, pandas, numpy, and matplotlib. The hardware used for running the experiments was a Dell laptop with a Core i7 processor and 16 GB of RAM.

D. Experimental Results

–Using machine learning

During this stage of the research, our primary focus was on fine-tuning the classifiers' parameters in order to achieve the best possible results. In order to determine the hyperparameters that should be used for each classifier, the study investigated a wide variety of possible combinations. For instance, change the kernel type and regularization parameter in SVM, as well as the maximum depth and number of trees in Random Forest. In the KNN algorithm, we altered the number of neighbors. In addition, we made adjustments to XGBoost's learning rate and the maximum depth, and we took into account SGD's loss function, penalty type, learning rate, and maximum iterations. In the field of logistic regression, we conducted research and experiments including different forms of penalties, varying degrees of regularization, and various solver methods. The study investigated the criterion for dividing nodes, the maximum depth that could be reached, the minimum number of samples needed for splitting a node, and the minimum number of samples needed for a leaf node while working on the Decision Tree Classifier. Our goal was to improve the performance of the classifiers by accurately classifying the preprocessed Arabic requirements into functional and non-functional classes. To achieve this, we systematically varied these parameters and evaluated the results of our experiments. 5-fold cross-validation to assess the performance of each classifier was utilized. This method includes dividing the preprocessed dataset into five equal parts, where four parts are utilized to train the classifiers and the remaining part for testing as mentioned by [50]. This procedure was repeated five times, using each part as the testing set once. The utilization of cross-validation provided us with a reliable evaluation of the performance of each classifier on the training data. The study utilized the macro-Recall score, the macro-Precision score, and the macro-F1 score metrics for the purposes of evaluation. After conducting extensive experimentation and optimizing parameters, we discovered that certain classifiers displayed the best performance in categorizing preprocessed Arabic requirements into functional and non-functional categories. Among the tested classifiers, the KNN classifier with k=3, the SVM classifier with an RBF kernel and C=1, and the Random Forest (RF) classifier with a maximum depth of 20 achieved remarkable results. Additionally, the XGBoost classifier exhibited superior performance with a maximum depth of 3. The Decision Tree (DT) classifier when used without any specific parameters, as well as the SGD classifier with a modified Huber loss function, elastic net penalty, alpha=0.01, and a maximum of 150 iterations, and the Logistic Regression classifier with C=10, L2 penalty, and the liblinear solver demonstrated the highest macro-Recall, macro-Precision, and macro-F1 scores among the tested classifiers. TABLE II presents the best performance achieved by Machine learning algorithms when No stemmer is applied. In the following table, we show the best performance achieved

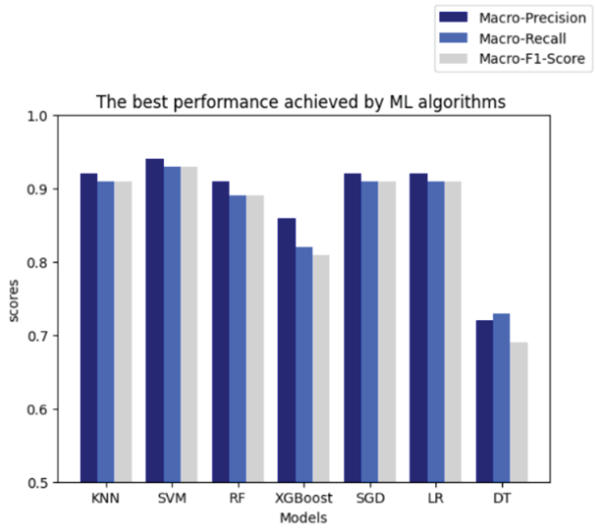

Fig. 2 Bar chart for the best performance achieved by Machine learning algorithms when no stemmer is applied.

by machine learning algorithms when no stemmer is applied.

Table 4 Performance Achieved by Machine Learning

Performance Measures	Macro-Precision	Macro-Recall	Macro-F1-Score
KNN	0.92	0.91	0.91
SVM	0.94	0.93	0.93
RF	0.91	0.89	0.89
XGBoost	0.86	0.82	0.81
SGD	0.92	0.91	0.91
LR	0.92	0.91	0.91
DT	0.72	0.73	0.69

In Figure 2, you can see a Bar chart for the best performance achieved by Machine learning algorithms when no stemmer is applied. Not only the classifier was evaluated, but we also examined how stemming techniques affect the classification outcomes. The initial approach involved the utilization of the ISRIStemmer algorithm, which is a frequently applied stemmer for Arabic texts. This algorithm worked by transforming words into their base forms, effectively minimizing word variations. We also tested the efficiency of the ArabicLightStemmer, which is another commonly used stemmer for Arabic. The main objective was to enhance the accuracy of the classification results [51]. Fig. 2. Bar chart for the best performance achieved by Machine learning algorithms when no stemmer is applied The results showed improved performance observed in multiple classifiers, such as KNN, SVM, Random Forest, XGBoost, Logistic Regression, and Decision Tree. Using ISRIStemmer, which assists in reducing the dimensionality of the text data and capturing root forms of words, led to better feature representation and improved classifiers' ability to differentiate between functional and non-functional requirements. To further enhance performance, we experimented with the ArabicLightStemmer algorithm and discovered that it had a particularly positive impact on the SGD classifier's performance. Similar to ISRIStemmer, ArabicLightStemmer aims to convert Arabic words into their base forms, resulting in a more standardized representation [35]. TABLE 5 illustrated the result of machine learning with the implementation of ISRIStemmer, TABLE 6 illustrated the result of machine learning with the implementation of ArabicLightStemmer.

Table 5 The result of machine learning with the implementation of ISRIStemmer

Performance Measures	Macro-Precision	Macro-Recall	Macro-F1-Score
KNN	0.93	0.92	0.92
SVM	0.96	0.95	0.95
RF	0.93	0.92	0.91
XGBoost	0.88	0.83	0.82
SGD	0.96	0.95	0.95
LR	0.95	0.94	0.94
DT	0.89	0.87	0.87

Table 6 THE RESULT OF MACHINE LEARNING WITH THE IMPLEMENTATION OF ARABICLIGHTSTEMMER

Performance Measures	Macro-Precision	Macro-Recall	Macro-F1-Score
KNN	0.92	0.90	0.90
SVM	0.95	0.94	0.94
RF	0.93	0.90	0.90
XGBoost	0.87	0.81	0.80
SGD	0.93	0.93	0.93
LR	0.94	0.94	0.93
DT	0.90	0.84	0.82

Figure 3 provides a bar chart for the best performance of ML algorithms with and without the implementation of stemming techniques, specifically focusing on the F1-score metric.

–Using deep learning

In the experiments involving deep learning, initially, we employed the pre-trained Arabert model in our deep learning experiments without any additional training. However, the achieved results were not as expected. To enhance the performance of the Arabert model, we proceeded to fine-tune it using the split method by training it on our specific dataset. Throughout the training process, we conducted experiments with the Arabert model, varying the random state values to observe the impact on the outcome with epochs 5. The tests included a range of random state values such as 32, 50, 52, 62, and 72. The variability in the random state values allowed us to assess the model's performance under diverse conditions by obtaining different splits of the data for training and testing. To evaluate the model's performance,

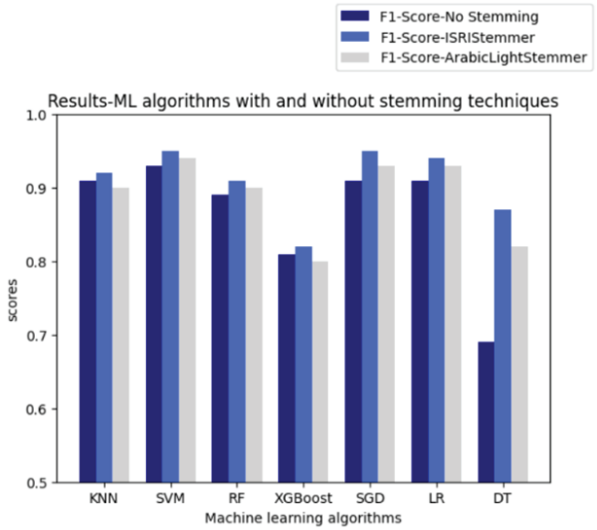

Fig. 3 Bar chart for the best performance of ML algorithms with and without the implementation of stemming techniques.

metrics such as accuracy, macro-precision, macro-recall, and macro-F1 score were utilized. The results were impressive, revealing significant improvements in classification accuracy and effectiveness. [52], [53] These findings demonstrate the significance of training deep learning models on data that is specific to their domain since this enables the models to adapt and become more specialized in the task at hand. The potential of deep learning was demonstrated in Arabic text classification through the combination of the pre-trained knowledge of the Arabert model and the additional training on our dataset, which resulted in excellent results. TABLE V illustrated the results obtained through the Arabert model.

Table 7 THE RESULTS OBTAINED THROUGH THE ARABERT MODEL

Performance Measures	Macro-Precision	Macro-Recall	Macro-F1-Score
random_state=32	0.97	0.97	0.97
random_state=50	0.94	0.94	0.94
random_state=52	0.97	0.97	0.97
random_state=62	0.94	0.94	0.94
random_state=72	0.97	0.98	0.97
Average	0.96	0.96	0.96

The Arabert model has emerged as a leading performer in the realm of deep learning, achieving an impressive F1-score of 0.96. This score demonstrates the model's effective grasp of the intricacies of the data and its ability to make precise predictions. The high F1-score showcases the Arabert model's proficiency in understanding and analyzing Arabic text, making it well-suited for various text analysis tasks. In terms of classifiers without stemming, the SVM classifier has proven to be exceptional with an F1-score of 0.93. This

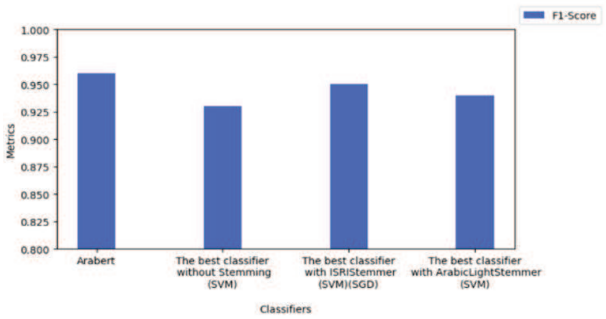

Fig. 4 Bar chart for the results of Deep Learning, Best ML Algorithms: No Stemming, ISRI Stemmer, and Arabic Light Stemmer.

underscores the SVM classifier's capability to accurately categorize data without relying on stemming techniques. The classifier's high F1-score indicates a strong balance between precision and recall, resulting in reliable predictions and overall impressive performance. In addition, the SVM classifier with the SGD classifier with the ISRI stemmer displayed impressive results among the classifiers that utilized stemming techniques. They achieved an F1-score of 0.95, showcasing their ability to effectively utilize stemming techniques. This improved their comprehension of the stemmed text, resulting in more precise and insightful predictions. TABLE VI illustrated the results of Deep Learning, Best ML Algorithms: No Stemming, ISRI Stemmer, and Arabic Light Stemmer.

Table 8 RESULTS OF DEEP LEARNING, BEST ML ALGORITHMS: NO STEMMING, ISRI STEMMER, AND ARABIC LIGHT STEMMER

Classifiers	F1- score
Arabert	0.96
The best classifier without Stemming (SVM)	0.93
The best classifier with ISRIStemmer (SVM), (SGD)	0.95
The best classifier with ArabicLightStemmer (SVM)	0.94

Figure 4 presents a comprehensive Bar chart that showcases the performance of deep learning. And best ML Algorithms in the context when no stemming is applied, as well as when utilizing the ISRI stemmer and the Arabic Light stemmer.

5 Conclusion

This study has made significant contributions to the field of Arabic requirement classification, addressing a critical research gap. An extensive and diverse dataset of Arabic requirements was successfully collected, filling the void of publicly available resources in this domain. This

dataset serves as a valuable asset for researchers and practitioners involved in Arabic requirement classification. The proposed method, which combines machine learning and deep learning algorithms, provides an efficient and accurate means of classifying Arabic requirements into Functional Requirements (FRs) and Non-Functional Requirements (NFRs). This automated approach reduces the time, effort, and costs associated with manual classification processes, enhancing productivity in software development projects. Additionally, this study lays the groundwork for future research in the field, offering a reference point for specialists to explore alternative methods and further streamline the manual classification process. In the context of related studies, this approach aligns with previous research on requirement classification, albeit with a focus on the Arabic language, which poses unique challenges due to its structural and semantic complexities. While some studies have attempted to classify English-written requirements, limited research has addressed Arabic requirements. This work not only addresses this research gap but also draws inspiration from the broader field of requirement classification, leveraging best practices and adapting them to the Arabic language. VI. RECOMMENDATIONS AND FUTURE WORK Regarding recommendations, several avenues for further research are suggested. Firstly, the performance of classifiers could be enhanced by incorporating more advanced deep learning techniques or leveraging transformer-based models specifically tailored for Arabic, such as BERT. Secondly, expanding the dataset with a larger volume of requirements from diverse domains could improve model generalization. Additionally, exploring the classification of other languages with similar structural challenges could broaden the applicability of this methodology. Lastly, collaborating with domain experts to refine the dataset and classification process further could yield even more accurate results. Overall, this study provides a solid foundation for advancing the field of Arabic requirement classification and offers valuable insights for future research directions.

Funding

The authors extend their appreciation to the Arab Open University for Funding this work through AOU research fund No. (AOURG-2023-021)

Acknowledgement

The authors would like to thank the Arab Open University and Al-Zaytoonah University for providing the necessary scientific research supplies to implement the research.

The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

References

[1] J. Klünder, M. Busch, N. Dehn, and O. Karras, Towards Shaping the Software Lifecycle with Methods and Practices. 2021.

[2] A. A. Thunibat, N. A. M. Zin, and N. Sahari, “Mobile Government User Requirements Model,” Journal of E-governance, vol. 34, no. 2, pp. 104–111, Jan. 2011.

[3] Y. Al-Kasabera, W. Alzyadat, A. Alhroob, S. Al Showarah and A. Thunibat, ”An automated approach to validate requirements specification”, Compusoft, vol. 9, no. 2, pp. 3578-3585, 2020.

[4] A. A. Thunibat, N. A. M. Zin, and N. Sahari, “Identifying User Requirements of Mobile Government Services in Malaysia Using Focus Group Methodch,” Journal of E-government Studies and Best Practices, pp. 1–14, Jan. 2011.

[5] T. A. Alrawashdeh, F. A. El-Qirem, A. Althunibat, and R. Alsoub, “A Prioritization Approach for Regression Test Cases Based on a Revised Genetic Algorithm,” Information Technology and Control, vol. 50, no. 3, pp. 443–457, Sep. 2021.

[6] H. M. S. A. Qaisi, G. Y. Quba, A. Althunibat, A. B. Abdallah, and S. AlZu’bi, An Intelligent Prototype for Requirements Validation Process Using Machine Learning Algorithms. 2021.

[7] D. Dave and V. Anu, Identifying Functional and Non-functional Software Requirements From User App Reviews. 2022.

[8] P. Talele and R. Phalnikar, Classification and Prioritisation of Software Requirements using Machine Learning – A Systematic Review. 2021.

[9] K. S. Shehadeh, N. Arman, and F. Khamayseh, Semi-Automated Classification of Arabic User Requirements into Functional and Non-Functional Requirements using NLP Tools. 2021.

[10] W. Fakhet, S. E. Khediri, and S. Zidi, Guided classification for Arabic Characters handwritten Recognition. 2022.

[11] G. Y. Quba, H. M. S. A. Qaisi, A. Althunibat, and S. AlZu’bi, Software Requirements Classification using Machine Learning algorithm’s. 2021.

[12] D. Dave, V. Anu, and A. S. Varde, Automating the Classification of Requirements Data. 2021.

[13] V. M. Khatian, Q. A. Arain, M. Alenezi, S. Ali, F. Shaikh, and I. Farah, Comparative Analysis for Predicting Non-Functional Requirements using Supervised Machine Learning. 2021.

[14] P. S. Shah, I. Ullah, and M. Shoaib, “Automatic Classification of Raw Software Requirements using Machine Learning,” Aug. 2021.

[15] R. Chatterjee, A. Ahmed, and P. Anish, “Identification and Classification of Architecturally Significant Functional Requirements,” In Proc. 2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering, Zurich, Switzerland, September 2020.

[16] A. K. M. A. Haque, A. Rahman, and Md. S. Siddik, Non-Functional Requirements Classification with Feature Extraction and Machine Learning: An Empirical Study. 2019.
[17] L. F. Li, N. C. Jin-An, Z. M. Kasirun, and C. Y. Piaw, "An Empirical Comparison of Machine Learning Algorithms for Classification of Software Requirements," International Journal of Advanced Computer Science and Applications, Jan. 2019.
[18] M. Binkhonain, and L. Zhao, "A Review of Machine Learning algorithms for Identification and Classification of Non-functional Requirements," Expert Systems with Applications X, March 2019.
[19] C. Baker, L. Deng, S. Chakraborty, J. Dehlinger, In Proc. 2019 IEEE 43rd Annual Computer Software and Applications Conference, WI, USA, July 2019.
[20] C. Wang, F. Zhang, P. Liang, M. Daneva, and M. Van Sinderen, Can app changelogs improve requirements classification from app reviews? 2018.
[21] M. Lu and P. Liang, Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. 2017.
[22] S. M. Alzanin, A. M. Azmi, and H. Aboalsamh, "Short text classification for Arabic social media tweets," Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 9, pp. 6595–6604, Oct. 2022.
[23] S. Al-Hagree and G. Al-Gaphari, Arabic Sentiment Analysis Based Machine Learning for Measuring User Satisfaction with Banking Services' Mobile Applications: Comparative Study. 2022.
[24] J. Atwan, M. Wedyan, Q. Bsoul, A. M. Hammadeen, and R. Alturki, "The use of stemming in the Arabic text and its impact on the accuracy of classification," Scientific Programming, vol. 2021, pp. 1–9, Nov. 2021
[25] M. A. Ibrahim, A. Alhakeem, and N. A. Fadhil, "Evaluation of Naïve Bayes Classification in Arabic Short Text Classification," Al-Mustansiriyah Journal of Science, vol. 32, no. 4, pp. 42–50, Nov. 2021.
[26] A.-K. Al-Tamimi, E. Bani-Isaa, and A. Al-Alami, Active Learning for Arabic Text Classification. 2021.
[27] M. Al-Smadi, B. Talafha, M. Al-Ayyoub, and Y. Jararweh, "Using long short-term memory deep neural networks for aspect-based sentiment analysis of Arabic reviews," International Journal of Machine Learning and Cybernetics, vol. 10, no. 8, pp. 2163–2175, Mar. 2018.
[28] A. Dahou, T. Seppänen, J. Zhou, and S. Xiong, "Arabic Sentiment Classification Using Convolutional Neural Network and Differential Evolution Algorithm," Computational Intelligence and Neuroscience, vol. 2019, pp. 1–16, Feb. 2019.
[29] I. Hmeidi, M. Al-Ayyoub, N. A. Abdulla, A. A. Almodawar, R. Abooraig, and N. A. Mahyoub, "Automatic Arabic text categorization: A comprehensive comparative study," Journal of Information Science, vol. 41, no. 1, pp. 114–124, Nov. 2014.
[30] S. Aftan and H. Shah, "Using the ARABERT model for customer satisfaction classification of telecom sectors in Saudi Arabia," Brain Sciences, vol. 13, no. 1, p. 147, Jan. 2023.
[31] A. Rai and S. Borah, "Study of various methods for tokenization," in Lecture notes in networks and systems, 2020, pp. 193–200.
[32] El Kah, A., & Zeroual, I. "The effects of Pre-Processing Techniques on Arabic Text Classification," International Journal of Advanced Trends in Computer Science and Engineering, vol. 10, no. 1, pp. 41–48, Feb. 2021, doi: 10.30534/ijatcse/2021/061012021.
[33] V. N. G. Raju, K. P. Lakshmi, V. K. Jain, A. Kalidindi, and V. Padma, Study the Influence of Normalization/Transformation process on the Accuracy of Supervised Classification. 2020.
[34] I. Gupta and N. Joshi, Tweet normalization: A knowledge based approach. 2017. doi: 10.1109/ictus.2017.8285996.
[35] D. H. Abd, W. Khan, K. A. Thamer, and A. Hussain, "Arabic Light stemmer based on ISRI stemmer," in Lecture Notes in Computer Science, 2021, pp. 32–45.
[36] D. Rani, R. Kumar, and N. Chauhan, Study and comparision of vectorization techniques used in text classification. 2022.
[37] Z. Jiang, B. Gao, Y.-L. He, Y. L. Han, P. Doyle, and Q. Zhu, "Text Classification Using Novel Term Weighting Scheme-Based Improved TF-IDF for Internet Media Reports," Mathematical Problems in Engineering, vol. 2021, pp. 1–30, Mar. 2021
[38] S. Alghowinem, T. Gedeon, R. Goecke, J. F. Cohn, and G. Parker, "Interpretation of depression detection models via feature selection methods," IEEE Transactions on Affective Computing, vol. 14, no. 1, pp. 133–152, Jan.
[39] P. J. M. Ali, "Investigating the Impact of Min-Max Data Normalization on the Regression Performance of K-Nearest Neighbor with Different Similarity Measurements," ARO. the Scientific Journal of Koya University, vol. 10, no. 1, pp. 85–91, Jun. 2022.
[40] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1007/bf00994018.
[41] W. A. Gardner, "Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique," Signal Processing, vol. 6, no. 2, pp. 113–133, Apr. 1984.
[42] Ho, T. K. Random decision forests. In IEEE Proceedings of 3rd international conference on document analysis and recognition (Vol. 1, pp. 278-282). Aug ,1995.
[43] J. S. Keller, M. R. Gray, and J. A. Givens, "A fuzzy K-nearest neighbor algorithm," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-15, no. 4, pp. 580–585, Jul. 1985.
[44] Pansare, J., Khande, M. S., Oswal, A., Munsiff, Z., Choudhary, S., & Kumbhar, V. 1900. Cricket score prediction using xgboost regression. URL: www. irjmets. com.
[45] E. B. Fowlkes, "Some diagnostics for binary logistic regression via smoothing," Biometrika, vol. 74, no. 3, pp. 503–515, Jan. 1987.
[46] W. A. Belson, "Matching and prediction on the principle of biological classification," Applied Statistics, vol. 8, no. 2, p. 65, Jun. 1959.
[47] R. Yacouby and D. Axman, Probabilistic extension of precision, recall, and F1 score for more thorough evaluation of classification models. 2020. doi: 10.18653/v1/2020.eval4nlp-1.9.
[48] P. Canbay and N. Bölücü, "A Language-Free hate speech identification on code-mixed conversational tweets," in Springer eBooks, 2023, pp. 102–108. doi: 10.1007/978-3-031-31956-3_8.

[49] M. Canesche, L. Braganca, O. P. V. Neto, J. A. M. Nacif, and R. B. Ferreira, Google CoLAB CAD4U: Hands-On Cloud Laboratories for Digital Design. 2021.
[50] T.-T. Wong and P. H. Yeh, "Reliable Accuracy Estimates from k-Fold Cross Validation," IEEE Transactions on Knowledge and Data Engineering, vol. 32, no. 8, pp. 1586–1594, Aug. 2020.
[51] K. Taghva, R. Elkhoury, and J. Coombs, Arabic stemming without a root dictionary. 2005. doi: 10.1109/itcc.2005.90.
[52] Jebril, I., Almaslmani, R., Jarah, B., Mugableh, M., & Zaqeeba, N. (2023). The impact of strategic intelligence and asset management on enhancing competitive advantage: The mediating role of cybersecurity. Uncertain Supply Chain Management, 11(3), 1041-1046.
[53] Alshehadeh, A. R., & Al-Khawaja, H. A. (2022). Financial Technology as a Basis for Financial Inclusion and its Impact on Profitability: Evidence from Commercial Banks. Int. J. Advance Soft Compu. Appl, 14(2).

Ahmad Althunibat serves as the Head of the Software Engineering department at Al-Zaytoonah University of Jordan, located in Amman, Jordan. He earned a PhD in software engineering from the National University of Malaysia in 2012 and his research interests include software testing, information systems acceptance, requirement engineering, and mobile technology

Reem Amro earned a Master's degree in Software Engineering from Al Zaytoonah University. Her research interests encompass requirement engineering, machine learning, deep learning, and healthcare. Notably, she has made valuable contributions, with research articles published in esteemed international conferences focused on software engineering.

Bilal Hawashin is an Associate Professor the Department of Artificial Intelligence, Al-Zaytoonah University of Jordan. He obtained his PhD degree in Computer Science from Wayne State University in Detroit, Michigan, with a specialty in Artificial Intelligence. He has worked as part of many artificial intelligence and machine learning-related projects during his academic career since 2003. These projects have resulted in journal and conference publications in various topics including text classification, text summarization, image classification, feature selection, clustering, multilabel text classification, collaborative filtering, recommender systems, dimensionality reduction, natural language processing, stemming, and fuzzy intelligent systems. Besides, he has taught various undergraduate and graduate courses in this field. He has been in the organizing committees of many ACM and IEEE conferences/workshops in this field.

Hend Al-Nuhait received the PhD degree in Computational Analysis and Modelling at Louisiana Tech University. Her research interests include Data Mining, Security Authentication, and Machine Learning. Dr. Al-Nuhait joined Arab Open University as an Assistant Professor of Information Technology and Computing, teaching undergraduate courses in programming, mathematics, and data structures.

Sally Almanasra is an associate professor at the faculty of computer studies, Arab Open University, Saudi Arabia. Her main teaching and research interests include AI, Image Processing, and E-Learning.

Haneen Al-Khawaja Research Assistant in Amman Arab University, Amman, Jordan from 2022-until now, Visiting Scholar (Prof. Thomas Puschmann team) in Swiss FinTech Innovation Lab, University of Zurich from 2021-until now, her research interests include Fintech, e-banking, Islamic banking, Cryptocurrencies.