Sistemasi: Jurnal Sistem Informasi
Volume 15, Nomor 10, 2026: 4142-4154
ISSN:2302-8149
e-ISSN:2540-9719

A Systematic Method for Classifying Software Requirements

[1]Asmaa Hadi Al Bayati*
[1]Department of Software, College of Computer Science and Mathematics, University of Mosul, Mosul 41002, Iraq
*e-mail: asmashade77@uomosul.edu.iq

(***received*****:** 21 September 2026, ***revised*****:** 18 October 2026, ***accepted*****:** 19 October 2026)

Abstract

This paper focuses on the classification of software requirements to functional and non-functional categories using ensemble learning algorithms. It relies on the PROMISE dataset. The first stage involves preprocessing the dataset, which includes removing irrelevant columns, performing text processing, and applying natural language processing (NLP) methods like eliminating redundant phrases and two-letter words, and finally converting text to numbers using NLP techniques like eliminating redundant phrases and two-letter words and finally converting text to numbers using TF-IDF. The most important step for classification is extracting the properties based on the statistical technique of Analysis of Variance (ANOVA) so that the data becomes ready as inputs to machine learning algorithms. The paper employs four types of boosting algorithms. The results indicated that the XGBoosting algorithm is the best algorithm, with an accuracy of .95, compared to the paper that used the same database.

Keywords: classification, ensemble learning, functional requirements, intelligent techniques, non-functional requirements.

1 Introduction

The System Development Life Cycle (SDLC) is a widely adopted framework for describing the systems development process. It comprises several stages that guide system design and analysis activities [1]. While the number and naming of these stages may vary by author or methodology, the Requirements Analysis (RA) phase is universally recognized as a critical component in any system development project. The project's success or failure is largely dependent on how the software requirements are categorized and analyzed. Properly identifying and documenting these requirements is fundamental, as it enables developers to define specifications that align with user needs. Therefore, accurately classifying software requirements during the SDLC is vital for minimizing errors and ensuring the effectiveness of later development stages [2].

The two primary categories of software requirements are functional requirements (FR) and non-functional requirements (NFR). The process of requirements engineering (RE) is essential to software development, focusing on identifying and documenting users' needs and expectations. Integrating artificial intelligence (AI) techniques into this process offers numerous advantages, such as enhanced data analysis, better requirements management, improved communication, early problem detection, automated requirements generation, and higher-quality requirements. Unlike FRs, which describe what the system should do, NFRs define how the system should perform. They cover aspects that affect overall system quality and user experience, including security, performance, reliability, usability, and scalability [3].

Manual methods for software requirement classification are time-consuming and error-prone [4]. As a result, artificial intelligence (AI) has become more popular, with branches like machine learning (ML) and ensemble learning (EL) that aim to reduce the amount of manual labor required of requirements engineers [5]. However, comparing these approaches to more conventional automated techniques has revealed encouraging outcomes. Therefore, classifying requirements with AI requires advanced techniques for accurate text analysis and categorization [6]. Adaboosting, CatBoost, GBM, and XGBoost are four ensemble learning algorithms used to classify software requirements. The best separation between classes is achieved by merging the results of multiple models and training a higher-level model to combine them.

The main contribution of the present paper is development of an integrated methodology for classifying software requirements that goes beyond the traditional application of machine learning

 ISSN:2302-8149
e-ISSN:2540-9719

algorithms. This paper is aimed at developing a highly accurate and reliable binary classifier to distinguish between FR and NFR. This precise binary classification is a crucial step and the cornerstone of intelligent requirements engineering systems, as any error at this preliminary stage will inevitably be compounded in subsequent analysis stages. The proposed methodology is based on an integrated pipeline that combines advanced natural language processing, TF-IDF-based text transformation, analysis of variance (ANOVA) to select the most distinctive features, and, finally, the application of four boosting algorithms.

Boosting techniques are widely used in machine learning to improve model performance and reduce error. These methods are mainly used in ensemble learning, a concept that combines multiple individual models to improve performance compared to the use of a single model [6].

Despite the use of ML and DL for the classification of software requirements to FR and NFR categories, most studies lack a unified framework that integrates NLP processing, TF-IDF representation, ANOVA-based feature selection, and boosting algorithms. Therefore, this study proposes an integrated framework comparing four boosting algorithms (AdaBoost, GBM, CatBoost, and XGBoost) and three traditional algorithms (Naive Bayes, logistic regression, and SVM) using TF-IDF and 5-fold cross-validation. The study's novelty lies in the systematic integration of these stages; XGBoost achieved an accuracy of 0.953 and an F1-score of 0.948 on the PROMISE data-set, outperforming previous results. The rest of the paper is in Section II; the related work is reviewed in Section 2; Section 3 presents the methodology for the proposed model; results are presented in Section 4; and the conclusion is in Section 5.

2 Related Works

Several prior studies have investigated the classification of specific requirements as FRs or NFRs. Several of these investigations also examined the effectiveness of applying AI and ML techniques to this purpose.

In 2020, Canedo and colleagues [7] displayed an innovative method to improve user classification of prerequisites by combining (TF-IDF) and linear regression (LR) techniques. The NFR was classified into eleven different types (subclasses of NFR) and twelve kinds of FR (in addition to subclasses of NFR). The researchers discovered that (LR) and (TF-IDF), when combined with machine learning algorithms, can provide a more effective solution. The overall classification of prerequisites achieved remarkable accuracy, with the F-measure, which reflects the balance between accuracy and sensitivity, reaching 91% for binary classification, 74% for classification of eleven NFR subtypes, and 78% for classification of twelve FR subtypes. The results reflect the effectiveness of the methods used in increasing classification precision and understanding of their fine details.

In 2021, Rahimi [8] presented a deep learning (DL) based study method (Binary Long Short-Term Memory (BiLSTM), Gated Recurrent Unit (GRU), Recurrent Neural Network (RNN), Convolutional Neural Network (CNN)) to develop single- and multi-stage classification systems for classifying SRs in various ways: binary classification and seventeen multi-class classifications for "NFRs" and "FRs". The two-stage categorization process obtained 93.40% in multi-class classification phase and 95.75% in the binary phase, while the single-phase system achieved 92.56%. The systems provide multi-class and binary classification with high precision. Consequently, it may be concluded that a two-phase system of classification is superior to a single-phase method. In the same year, Yquba [9] proposed an approach for automatic classification of software requirements via ML. They used the bag-of-words (BoW) technique to convert text into numerical representations based on word frequency, thereby facilitating processing by machine learning algorithms. Two main algorithms, the k-nearest neighbor (KNN) algorithm and support vector machines (SVMs) algorithm, were tested for classifying requirements to functional and non-functional categories. results showed that (SVM), when used with (BoW), achieved a classification accuracy of 90%, which significantly outperforms KNN. This success is due to SVM's ability to effectively identify boundaries between classes in text data, enabling more accurate classification.

In 2022, Khurshid et al. [10] presented several machine learning techniques for automatically categorizing NFR. They used (SVM), logistic regression (LR), (KNN), random forest (RF), multinomial naive Bayes (MNB), and hybrid rule-based KNN algorithms. Researchers have relied on 2 main feature extraction techniques, BoW and inverse term frequency (TF-IDF), for converting texts

 ISSN:2302-8149
e-ISSN:2540-9719

to numerical representations. After applying these methods, the researchers achieved a mean accuracy of 85.70%, which they consider excellent for classifying NFR requirements.

In 2023, Amin Khan et al. [1] proposed an approach to learning that uses a Word2Vec model trained on NFR datasets to detect and categorize NFRs in software development. By leveraging the semantic similarity between FRs and NFRs, the method effectively classifies NFRs. At the same time, evaluating multiple transfer learning models, like Distil Bert and BERT. Electra-base text encoding as discriminators, and eXtreme Language Network (XLNet), the proposed XLNet model was found to perform better than other models. Its precision, recall, accuracy, and F1 score were all exceptionally high, and its Matthews' correlation coefficient (MCC) was an amazing 0.9066. These findings clearly show how well XLNet performs in NFR identification and classification, yielding precise and reliable results. In the same year, Rahman et al. [11] categorized non-functional requirements into 11 categories by combining multiple machine learning algorithms with 4 text vectorization techniques. For requirements classification, the paper's “Chi-Squared” algorithm performed the worst, with hashing coming in second. BoW and TF-IDF reported precision (0.667), accuracy (0.667), and recall (0.539). At 81.5%, TF-IDF with Level of Service Requirements LSVC had the finest accuracy. They concluded that TF-IDF and LSVC are the most effective combination for classifying non-functional requirements. Passive Aggressive (PA) and LSVC were placed on top when using TF-IDF, with a recall of 0.815, and RF was 0.957, the highest value in the results.

In 2024, Rahman and colleagues [12] conducted an experiment using 914 NFR cases from two popular databases. Applied models underwent performance analysis, and several metrics were used to assess the results. Notably, DReqANN model outperforms other models in the classification of NFR, which achieved accuracy ranging from 81% to 99.80%, recall from 74% to 89%, and F1 score from 83% to 89%. Those important findings illustrate deep learning framework's exceptional effectiveness in handling NFR classification problems and show how it can advance the study of NFR classification and analysis. Recognizing the constraints of the suggested framework is crucial because it largely depends on sequential preprocessing steps, and any irregularities may affect the accuracy of NFR classification in these steps.

3 Proposed Methodology

One crucial phase in the SDLC is the use of AI to categorize software requirements, as traditional methods are often time-consuming and lack accuracy, potentially leading to project failure. This paper presents a classification approach using boosting ensemble learning implemented in Python 3.1. The main workflow of the suggested model is illustrated through the activity diagram shown in Figure 1.

In this paper, we depend on the Promise-exe dataset [13], which includes (3) columns and 970 rows for many projects, as shown in a sample of the dataset in Figure 2.

Data that support findings of the present study are available publicly from the PROMISE repository [13]. No new data were created or generated as part of this research. The activity diagram in Figure 1 outlines the stages and sequence of operations involved in classifying system requirements data. The key steps in the proposed model are as follows:

- Loading and reading the dataset.
- Preprocessing the data, which involves tasks such as data cleaning, text segmentation, removing unnecessary words and characters, eliminating irrelevant columns, stemming words to their root form, and vectorizing the data using two methods to produce a TF-IDF dataset.
- Selecting features using the ANOVA technique.
- Splitting the data-set to two subsets: (80%) for training and (20%) for testing, with k-fold equal to 5.
- Selecting suitable algorithms and adjusting coefficients, and performing both training and testing.
- Evaluating performance using specific measures and the confusion matrix.
- Display the final results.

 ISSN:2302-8149
e-ISSN:2540-9719

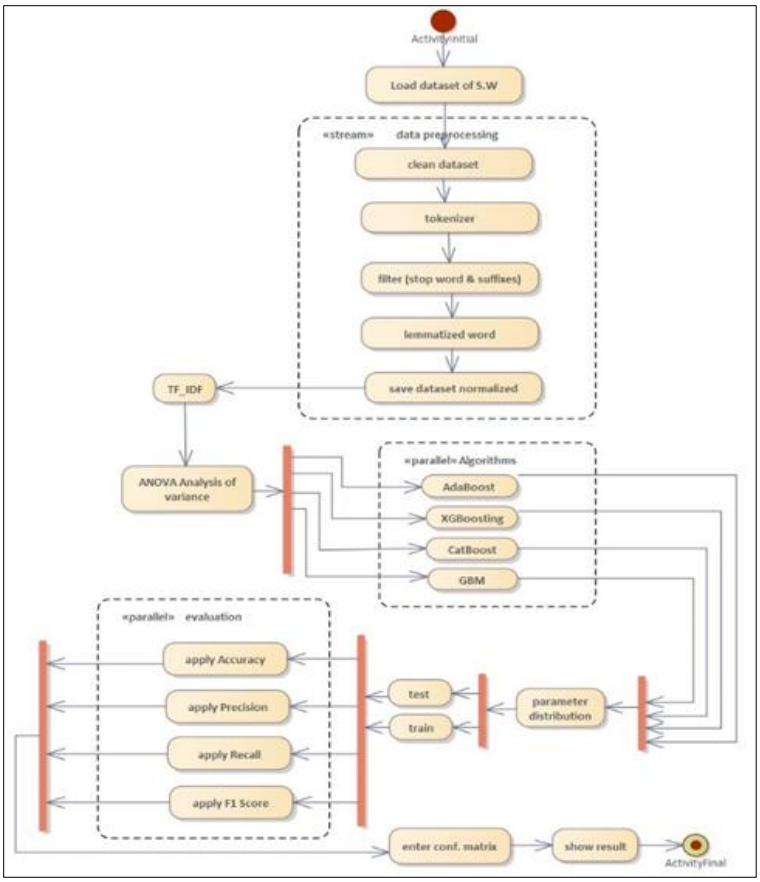

Figure 1 Activity Diagram of the Proposed Model

	A	B	C
1	ProjectID	RequirementText	_class_
2	1	The system shall refresh the display every 60 seconds.	PE
3	1	The application shall match the color of the schema set forth by Department of Homeland Security	LF
4	1	If projected the data must be readable. On a 10x10 projection screen 90% of viewers must be able	US
5	1	The product shall be available during normal business hours. As long as the user has access to the clien	A
6	1	If projected the data must be understandable. On a 10x10 projection screen 90% of viewers must be	US
7	1	The product shall ensure that it can only be accessed by authorized users. The product will be able to	SE
8	1	The product shall be intuitive and self-explanatory. 90% of new users shall be able to start the display	US
9	1	The product shall respond fast to keep up-to-date data in the display.	PE
10	1	The system shall have a MDI form that allows for the viewing of the graph and the data table.	F
11	1	The system shall display Events in a vertical table by time.	F
12	1	The system shall display the Events in a graph by time.	F
13	1	The system shall display Events or Activities.	F
14	1	The display shall have two regions: left 2/3 of the display is graphical right 1/3 of the display is a data table	F

Figure 2 Sample of Promise-Exe Dataset

3.1 Preprocessing

Since the PROMISE_exp dataset is in raw text format, it requires extensive cleaning and transformation before use in ML. The steps of preprocessing were implemented in Python using libraries such as “NLTK”, re (modal expressions), and “scikit-learn”. The following sequential steps were applied:

- Data cleaning: delete columns that do not affect the classification (such as project name), and duplicate rows were removed. All non-alphabetic characters (special characters [^, $, @, &], numbers, and punctuation marks) have also been removed from text. The text has then been converted to lowercase for consistency.
- Deleting common words and Tokenization: Common English words (such as "the," "is," and "at") and words of two or fewer letters were filtered because they typically have little significance for classification purposes. Cleaned text was segmented into individual words using the “word_tokenize” tool from the NLTK library.

 ISSN:2302-8149
e-ISSN:2540-9719

- Lemmatization: lemmatized words down to their root using the “WordNetLemmatizer” tool from the NLTK library to standardize variations in wording syntax (e.g., " going", "went", and "goes" all become " go"). This reduces feature space dimensionality.
- Text Vectorization (TF-IDF): The pre-processed text codes were converted into a digital feature array using TF-IDF technology. It used “TfidfVectorize” from the scikit-learn library with parameters set to max_features = 5,000 and ngram_range = (1,2) (to include both single as well as binary words).
- Target variable coding: grouped the original requirements into two categories: "F" for functional requirements and "NF" for all types of non-functional requirements. Figure 3 illustrates this binary target classification.

name	wmc	dit	noc	cbo	rfc	lcom	ca	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
org.apach	2	2	0	3	4	1	0	0.666667	0.666667	0	0	4	1	0.5	0
org.apach	10	1	0	4	37	0	0	0	0.34	0	0	26.6	5	1.6	0
org.apach	27	1	0	1	43	13	0	0	0.253086	0	0	42.18519	26	5.7407	0
org.apach	1	1	0	21	1	0	0	0	1	0	0	0	1	1	1
org.apach	3	1	2	19	4	3	0	0	1	0	0	1.333333	1	0.6667	1
org.apach	2	3	0	8	8	1	0	0.952381	0.583333	1	1	29.5	1	0.5	1
org.apach	7	2	0	6	30	1	0	0.142857	0.314286	1	1	51.57143	12	3.5714	0
org.apach	1	1	0	5	2	0	1	0	1	0	0	12	0	0	1
org.apach	9	2	3	4	14	0	0	0.619048	0.416667	1	4	21.11111	2	1	0
org.apach	6	1	0	8	29	0	5	0	0.479167	0	0	61	1	0.6667	0
org.apach	15	2	1	19	50	0	1	0.611111	0.251852	1	5	35.93333	6	1.2667	1
org.apach	5	4	0	2	18	0	1	0.866667	0.4	1	1	74	14	4.8	1
org.apach	17	2	0	12	55	12	0	0.428571	0.158824	2	4	22.52941	7	2.0588	1
org.apach	14	1	1	18	17	91	0	0	0.297619	0	0	12.5	1	0.9286	1
org.apach	2	2	0	6	4	1	0	0.916667	0.666667	1	3	18.5	4	2	0
org.apach	6	2	0	4	10	15	0	0.615385	0.666667	1	3	4.166667	1	0.8333	0
org.apach	1	2	0	14	2	0	1	1	1	0	0	9	0	0	0

Figure 3 Data After Preprocessing

3.2 Feature selection

Feature Selection is performed with the use of “SelectKBest" function with ANOVA as the criterion of evaluation. A maximum of 180 was used to determine the appropriate number of features to retain, which was eventually selected as it provided the optimal trade-off between high classification accuracy and computational efficiency. This method helps identify and preserve the dataset's most essential features strongly associated with the target variable, thereby enhancing the training of machine learning algorithms.

Create a classifier using algorithms such as AdaBoost, CatBoost, GBM, and XGBoost to be used in the training and evaluation process. The data is divided into (20%) for the test set and (80%) for the training set. For more dependable performance estimates, we used 5-fold cross-validation. The data were split to 5 equal parts; in every round, four parts were utilized in order to train the model, and the remaining part has been utilized in order to test it. This was repeated ten times, and the overall performance is the average across all rounds. The feature selection process here is done using the (SelectKBest) function with the test criterion [14].

The main step of ANOVA: Suppose you have k groups, each with several observations:

1. Divide the data into groups based on the target classes.
2. Calculate Group Means, which include "mean of each group and mean of all data".
3. Calculate Variance Components, including Between-group variability (SSB) and Within-group variability (SSW).
4. Calculate “F” function within the equation:

$$F = \frac{SSBL(k-1)}{SSWL(N-k)} \quad (1)$$

Which N represents total number of observations.

5. Select features with low p-values, where F is significantly large (or p-value $< \alpha$, commonly 0.05), not all means are equal, as important features for the classification model.
6. Repeat steps 2-3 and make adjustments to the features or data to improve performance.

The number of features to retain is determined by "$k_$ features”, with a maximum limit of 180. This method aims to identify features most closely related to the target, i.e., to assess how strongly

 ISSN:2302-8149
e-ISSN:2540-9719

each feature and the target are related, which helps in improving the model's performance. Features with higher values indicate a greater correlation, and the best k features are retained based on this test [14].

3.3 Boosting algorithms

Boosting is an ensemble technique that converts weak learners into strong classifiers by repeatedly applying a base learning algorithm to modified training data versions [15]. Weak learners are models that perform only a little bit better when compared to the random guessing, whereas the strong learners achieve a high level of accuracy. In this paper, we apply boosting algorithms, which include the AdaBoost, CatBoost, GBM, and XGBoost. In boosting, each iteration emphasizes examples misclassified in previous rounds—these cases are given higher weight and used to train the next weak learner [16, 17]. After much iteration, the individual models' outputs are combined (often by weighted voting or averaging [18, 19]. This process continues iteratively, and their results are pooled using weighted voting or averaging after a certain number of weak learners have been attained [20]. The main goal of boosting is to reduce bias rather than variance, and it works especially well with simple base learners like decision stumps, since increasing the weight on misclassified samples helps correct systematic bias in the base classifier [21, 22]. To ensure fair comparison and reproducibility, the hyperparameters of the implemented algorithms were carefully tuned using grid search. Table 1 summarizes the main hyperparameter configurations used in this study.

Table 1 The Hyperparameters of the Boosting Algorithm

Algorithm	Hyperparameters			
	n_ estimators	Learning rate	Max_ depth	Subsample
AdaBoost	100	0.01	-	-
GBM	200	0.1	3	-
CatBoost	300	0.2	6	-
XGBoost	300	0.2	6	0.8

- **Ada-boosting algorithm:** Ada-Boost is a boosting method that creates a strong classifier by combining several weak learners. It was the first boosting algorithm to be successful, and it typically uses Decision stumps, which are single-split decision trees, as its base learners. In the Ada-Boost training process, a base classifier – often a decision tree - is trained on the data. After each round, the classifier's performance is used to update the sample weights: instances that are correctly classified are given lower weights, while misclassified instances are given higher weights. This modification guarantees that later classifiers focus more on examples that are more challenging to classify [22]. A summary of the Ada-Boost algorithm is presented in Figure 4.
- **Categorical Boosting (CatBoost):** Yandex introduced the CatBoost method in 2018. It is believed to be a sophisticated gradient-boosting library created especially to handle categorical data in machine learning. CatBoost is an open-source method that has rapidly gained popularity because it doesn't require extensive data preparation to produce high-performance models. CatBoost is so good at handling categorical data natively that it is superior to alternative gradient boosting techniques for problems involving complex, real-world datasets [23, 24]. The CatBoost algorithm is summarized in Figure 5.

 ISSN:2302-8149
e-ISSN:2540-9719

Input : DN, weak classier WeakLearn ,
set of classiers ,
number of iterations T
Procedure:
Initialize sample weights:
For t = 1 : T
- Train the basic classifier $h_t(x)$ by minimizing
- Calculate the weight α_t of the classifier using:

$$h_t \leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{n=1}^{N} w^{(n)} \mathbb{I}\{h(\mathrm{x}^{(n)}) \neq t^{(n)}\}$$

- Update the sample weight using :

$$w^{(n} \leftarrow w^{(n)} \exp\left(-\alpha_t t^{(n)} h_t(\mathrm{x}^{(n)})\right)$$

$$\left[\equiv w^{(n)} \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathrm{x}^{(n)}) \neq t^{(n)}\}\right)\right]$$

End for
Classifier H(x) is :

$$H(\mathrm{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathrm{x})\right)$$

Output: Storng classifier H(x)

Figure 4 The AdaBoost Algorithm

input : $M, \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}, \alpha, L, \{\sigma_i\}_{i=1}^{s}, Mode$
$grad \leftarrow CalcGradient(L, M, y)$;
$r \leftarrow random(1, s)$;
if $Mode = Plain$ **then**
  $G \leftarrow (grad_r(1), \ldots, grad_r(n))$;
if $Mode = Ordered$ **then**
  $G \leftarrow (grad_{r,\sigma_r(i)-1}(i)$ for $i = 1..n)$;
$T \leftarrow$ empty tree;
foreach *step of top-down procedure* **do**
  foreach *candidate split* c **do**
    $T_c \leftarrow$ add split c to T;
    if $Mode = Plain$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_r(p)$ for
       $p : leaf_r(p) = leaf_r(i))$ for $i = 1..n$;
    if $Mode = Ordered$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_{r,\sigma_r(i)-1}(p)$ for
       $p : leaf_r(p) = leaf_r(i), \sigma_r(p) < \sigma_r(i))$
       for $i = 1..n$;
    $loss(T_c) \leftarrow ||\Delta - G||_2$
  $T \leftarrow \text{argmin}_{T_c}(loss(T_c))$
if $Mode = Plain$ **then**
  $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha\,\text{avg}(grad_{r'}(p)$ for
   $p : leaf_{r'}(p) = leaf_{r'}(i))$ for $r' = 1..s, i = 1..n$;
if $Mode = Ordered$ **then**
  $M_{r',j}(i) \leftarrow M_{r',j}(i) - \alpha\,\text{avg}(grad_{r',j}(p)$ for
   $p : leaf_{r'}(p) = leaf_{r'}(i), \sigma_{r'}(p) \leq j$ for $r' = 1..s$,
   $i = 1..n, j = 1..n$;
return T, M

Figure 5 The CatBoost Algorithm

- **Gradient Boosting Machine (GBM):** The GBM, which is a strong supervised learning approach, is widely employed in real-world situations. By focusing on the error at every one of the iterations, the Gradient Boosting Machine aggregates a set of weak learners and produces a strong learner as the sum of the subsequent weak learners. It is an additive model because the current classifier's residual is used as input to the subsequent classifier, forming the foundation on which the trees are constructed [25]. To capture the highest variance in the data, the classifiers introduce a learning rate, which allows them to update the residuals in a step-by-step manner. In keeping with Gradient Descent, which uses a similar logic, this technique is named Gradient Boosting because it allows us to gradually go in the right direction toward improved prediction by finding negative gradients and moving in the opposite direction to minimize loss. We therefore obtain a predicted value that is extremely

 ISSN:2302-8149
e-ISSN:2540-9719

close to the observed value, given the number of classifiers [26]. Figure 6 summarizes the gradient boosting machine algorithm.

Inputs:

- input data $(x, y)_{i=1}^{N}$
- number of iterations M
- choice of the loss-function $\Psi(y, f)$
- choice of the base-learner model $h(x, \theta)$

Algorithm:

1: initialize $\widehat{f_0}$ with a constant
2: **for** $t = 1$ to M **do**
3: compute the negative gradient $g_t(x)$
4: fit a new base-learner function $h(x, \theta_t)$
5: find the best gradient descent step-size ρ_t:

$$\rho_t = \arg\min_{\rho} \sum_{i=1}^{N} \Psi\left[y_i, \widehat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)\right]$$

6: update the function estimate:

$$\widehat{f}_t \leftarrow \widehat{f}_{t-1} + \rho_t h(x, \theta_t)$$

7: **end for**

Figure 6 The Gradient Boosting Machine Algorithm

- **eXtreme Gradient Boosting (XGBoosting):** XGBoost is a gradient-boosting ensemble machine learning technique that relies on decision trees. It relied on sequential decision trees built with an enhanced gradient-tree boosting method. It can complete related calculations quickly across all computational environments. Due to its ability for classifying marks and modeling more recent qualities, which is why it is used frequently. The XGBoost approach was implemented on structured and tabular data-sets; its popularity has skyrocketed [27]. The remarkable capacity of the algorithm to model complicated systems, along with its higher prediction accuracy, interpretability, classification variety, and adaptability, distinguishes it from other machine learning algorithms. It is an improvement to parallel trees that accurately and quickly resolve various data science problems [28]. The XGBoosting algorithm is summarized in Figure 7.

Input : training data S=(x1,y1),....(x2,y2),....,(xm,ym)
Adifferential loss function L(y,F(x))
Number of iteration T

Procedure :

- Initialize the model with constant value using :

$$F_0 = argmin_{\alpha} \sum_{i=1}^{N} L(y_t, F(x))$$

- For m=1 : T
  Calculate the false residuals :

$$r_{mi} = \left[\frac{\delta L(y_i, F(x))}{\delta F(x)}\right] F(x) = F_{m-1}(x)$$

  Train basi learner using the training set:

$$D = x_i, r_{mi}$$

  Obtain *Pm* by performing the linear search optimization :

$$p_m h_m(x) = argmin_{p,h} \sum_{i=1}^{N} L(y_i, F_{m-1}(x)) + ph(x_i)$$

  Update the model :

$$F_m(x) = F_{m-1}(x) + p_m h_m(x)$$

- End for

Output: Return the Final Model F(x)

Figure 7 The XGBoosting Algorithm

 ISSN:2302-8149
e-ISSN:2540-9719

3.4 Experimental Setup and Parameter Configuration

Experiments have been conducted with the use of the PROMISE dataset, which contains textual software requirements classified to FR and NFR categories. The data underwent preprocessing and transformation into a TF-IDF representation—utilizing a maximum of 5,000 features and an n-gram range of (1–2)—followed by feature selection via ANOVA. A maximum of 180 features was retained based on preliminary empirical evaluation to strike a balance between performance and computational efficiency.

The data-set has been split to 80% for training and 20% for testing; the training set has been evaluated using five-fold cross-validation to ensure a more reliable estimate and reduce reliance on a single data split.

Four boosting algorithms—AdaBoost, GBM, CatBoost, and XGBoost—were evaluated under identical conditions. Hyperparameters (number of estimators, maximum tree depth, learning rate, and subsample ratio) have been tuned using grid search, following the settings outlined in Table 1.
These methods were selected for their complementary nature: TF-IDF provides an effective representation of term importance, while ANOVA reduces dimensionality by retaining features most statistically correlated with the categories. Meanwhile, boosting algorithms enable the integration of weak learners to iteratively improve classification accuracy; evaluating them side-by-side allows for a fair comparison of their effectiveness in classifying software requirements.

4 Results

After applying ensemble learning methods to the t dataset using four techniques, we also implemented three baseline classifiers—Logistic Regression, SVMs, and Multinomial Naïve Bayesian—trained on the same TF-IDF feature representation and evaluated with 5-fold cross-validation. Results have been shown in Table2.

The results that have been listed in Table 2 demonstrate the effectiveness of the suggested pre-processing and feature selection pipeline, particularly after addressing class imbalance with SMOTE. The high F1-scores across all boosting algorithms, especially for XGBoost (0.953), are higher than those reported in previous studies using the same database (PROMISE). For example, the study by Quba et al. [9] achieved 90% accuracy using the SVM algorithm, while the study by Canedo and Mendes [7] achieved 91% accuracy using linear regression (LR). indicate a balanced performance between precision and recall. Which suggests that the model isn't simply predicting the majority class but is effectively learning to distinguish both FR and NFR. The XGBoost algorithm likely outperforms others due to its built-in regularization, which helps prevent overfitting on the minority class, and its efficient handling of sparse data, which is common after TF-IDF vectorization.

Table 2 Final Results of the Algorithms with the (TF-IDF Bow) Data Set

Algorithm	Accuracy	Precision	Recall	F1 Score
Adaboosting	0.925	0.930	0.916	0.922
GBM	0.944	0.941	0.946	0.948
CatBoost	0.941	0.948	0.933	0.939
XGBoosting	0.953	0.955	0.943	0.948
Naive Bayes	0.899	0.901	0.879	0.890
Logistic Regression	0.919	0.920	0.90	0.911
SVM	0.928	0.927	0.912	0.920

The histogram in Figure8 depicts the accuracy of the boosting algorithms, indicating that the XGBoost algorithm performs better than the others. The yellow color represents XGBoosting, the orange color represents the GBM algorithm, the gray color represents the CatBoost algorithm, and the blue color represents Adaboosting.

 ISSN:2302-8149
e-ISSN:2540-9719

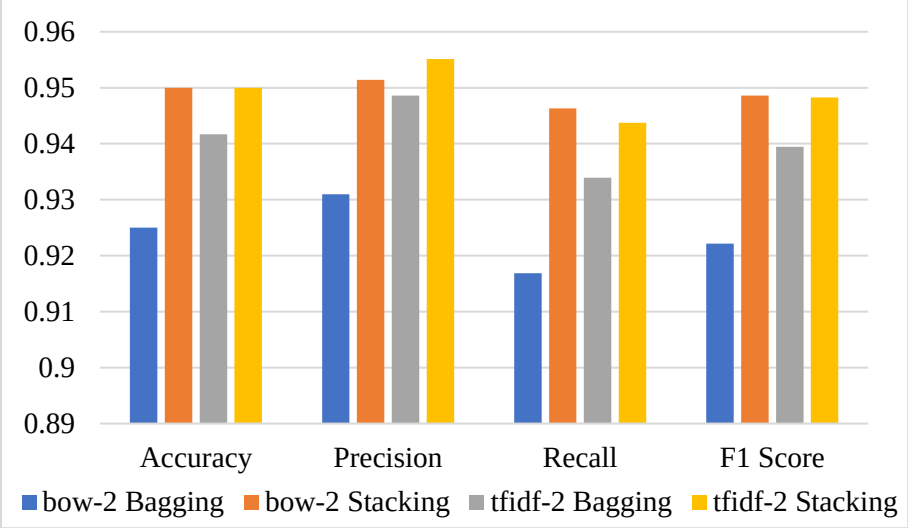

Figure 8 The Results of All Boosting Algorithms

Notation that the XGBoosting algorithm achieved the best results and outperformed the other algorithms on the two data sets, achieving Accuracy (0.953), Recall (0.943), Precision (0.955), and F1 Score (0.948). GBM has the second-best results, Accuracy (0.944), Recall (0.946), Precision (0.941), and F1 Score (0.948) using the (TF-IDF) technique.

To analyze the classifiers' performance in greater depth, Figure 9 presents the confusion matrices for the four algorithms studied. The XGBoost correlation matrix shows that the model misclassified X from the Functional requirements and Y from the Non-Functional requirements in a sample of Z from the test set.

It is worth noting that while the present study focuses on boosting algorithms with TF-IDF representation, recent advances in NLP such as BERT and RoBERTa have shown promising results in capturing contextual semantics. However, these transformer-based models were not included in the current comparison due to their higher computational cost and the study's aim to evaluate the effectiveness of boosting algorithms under a unified TF-IDF pipeline. Future work will address this limitation by incorporating BERT and RoBERTa for a more comprehensive comparison.

As seen in the confusion matrices in Figure 9, there are clear misclassification tendencies. For XGBoost," the most accurate model," most errors came from NFRs being labeled as FRs. Reviewing those PROMISE dataset examples detects that many NFRs (e.g., usability or operational constraints) use action-oriented wording like "system shall provide the user with the ability to...," which closely mirrors the phrasing of functional requirements. Fewer FRs were wrongly marked as NFRs, often when they described complex business rules and included quality-related adjectives or phrases. Overall, although boosting methods (notably XGBoost) perform well, the textual distinction between FR and NFR can be ambiguous, suggesting the need for further work on context-aware models.

 ISSN:2302-8149
e-ISSN:2540-9719

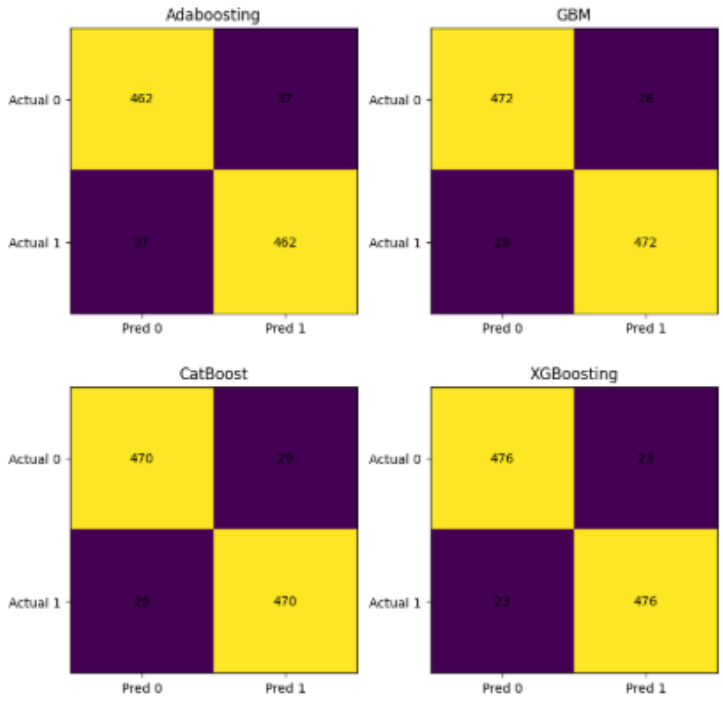

Figure 9 Confusion Matrix for Boosting Algorithms

5 Conclusion

The article provides a method for classifying software requirements using boosting algorithms. The model applied to the PROMISE_exp dataset, which contains textual data, was employed for this purpose. Various preprocessing steps were applied, including natural language processing, data cleaning, text segmentation, and the removal of irrelevant text segments. The TF-IDF algorithm was applied to transform the text into numerical form.

The paper unified eleven distinct types into one category, non-functional requirements (NFR), alongside functional requirements (FR). Feature selection was performed using the ANOVA algorithm to identify the most important features for classifying requirements.

The dataset has been divided to two parts: 20% for testing and 80% for training, with k-fold equal to five. Ensemble learning techniques were used for classification, with XGBoost algorithm delivering the optimal performance, achieving an accuracy of 0.95 and outperforming other boosting methods.

In future work, this research acknowledges the outstanding results achieved by the boosting algorithms, particularly XGBoost, while noting some limitations that open avenues for future work. Aims to expand the proposed framework by investigating Transformer models such as BERT and RoBERTa to capture contextual and semantic relationships within software requirements. Also hybrid architectures, evaluating them using diverse datasets, precisely classifying non-functional requirements, and examining computational cost, complexity, and interpretability

References

[1] M. A. Khan, M. S. Khan, I. Khan, S. Ahmad, and S. Huda, "*Non-Functional Requirements Identification and Classification using Transfer Learning Model*," *IEEE Access*, Vol. 11, pp. 74997-75005, 2023, DOI: 10.1109/ACCESS.2023.3295238.

[2] A. A. Fadhil and A. H. Albayati, "*Automatic Evaluation of the Student's Activity Diagrams in the Educational Process*," *Information Systems Engineering*, Vol. 30, No. 8, pp. 2149-2156, 2025, DOI: 10.18280/isi.300820.

[3] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, "*Towards Supporting Software Engineering using Deep Learning: A Case of Software Requirements Classification*," in *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*,

 ISSN:2302-8149
e-ISSN:2540-9719

Merida, Mexico, 25-27 Oct. 2017 2017: IEEE, pp. 116-120, DOI: 10.1109/CONISOFT.2017.00021.
[4] S. N. Nemat and S. R. Ghanem, "*Classification of Software Engineering Documents based on a Hybrid*," *Al-Rafidain Journal of Computer Science and Hybrid Model Mathematics*, Vol. 12, No. 2, pp. 61-87, 2018.
[5] A. Shaheen, "*A Machine Learning Approach for User Behavior Analysis in Developing Websites*," *AL-Rafidain Journal of Computer Sciences and Mathematics*, Vol. 18, No. 2, pp. 9-19, 2024, DOI: 10.33899/csmj.2024.146537.1106.
[6] N. Rahimi, F. Eassa, and L. Elrefaei, "*An Ensemble Machine Learning Technique for Functional Requirement Classification*," *Symmetry*, Vol. 12, No. 10, p. 1601, 2020, DOI: 10.3390/sym12101601.
[7] E. Dias Canedo and B. Cordeiro Mendes, "*Software Requirements Classification using Machine Learning Algorithms*," *Entropy*, Vol. 22, No. 9, p. 1057, 2020, DOI: 10.3390/e22091057.
[8] N. Rahimi, F. Eassa, and L. Elrefaei, "*One-and Two-Phase Software Requirement Classification using Ensemble Deep Learning*," *Entropy*, Vol. 23, No. 10, p. 1264, 2021, DOI: 10.3390/e23101264.
[9] G. Quba, H. Qaisi, A. Thunibat, and S. AlZu'bi, "*Software Requirements Classification using Machine Learning Algorithm's*," in *2021 International Conference on Information Technology (ICIT)*, 2021, pp. 685-690, DOI: 10.1109/ICIT52682.2021.9491688.
[10] I. Khurshid *et al*., "*Classification of Non-Functional Requirements from IoT Oriented Healthcare Requirement Document*," *Frontiers in Public Health*, Vol. 10, p. 860536, 2022, DOI: 10.3389/fpubh.2022.860536.
[11] A. Rahman, A. Nayem, and S. Siddik, "*Non-Functional Requirements Classification using Machine Learning Algorithms*," *International Journal of Intelligent Systems and Applications*, Vol. 15, No. 3, pp. 56-69, 2023, DOI: 10.5815/ijisa.2023.03.05.
[12] K. Rahman, A. Ghani, S. Misra, and A. U. Rahman, "*A Deep Learning Framework for Non-Functional Requirement Classification*," *Scientific Reports*, Vol. 14, p. 3216, 2024, DOI: 10.1038/s41598-024-52802-0.
[13] Raulvilla, "*Software Cost Estimation from PROMISE*," Kaggle, 2025. [Online]. Available: https://www.kaggle.com/datasets/raulvilla/software-cost-estimation-from-promise.
[14] P. Dugard, J. Todman, and H. Staines, "*Multivariate Analysis of Variance (MANOVA)*," in *Approaching Multivariate Analysis, 2nd Edition*: Routledge, 2022, pp. 55-82.
[15] Y. Sun, Z. Li, X. Li, and J. Zhang, "*Classifier Selection and Ensemble Model for Multi-Class Imbalance Learning in Education Grants Prediction*," *Applied Artificial Intelligence*, Vol. 35, No. 4, pp. 290-303, 2021, DOI: 10.1080/08839514.2021.1877481.
[16] L. F. Jarallah, "*Medical Decision Support Systems for Diagnosing Diseases based on Ensemble Learning Algorithms*," *Journal of Computer Sciences and Mathematics*, Vol. 18, No. 2, pp. 115-120, 2024.
[17] I. D. Mienye and Y. Sun, "*A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects*," *IEEE Access*, Vol. 10, pp. 99129-99149, 2022, DOI: 10.1109/ACCESS.2022.3207287.
[18] I. D. Mienye, Y. Sun, and Z. Wang, "*Improved Predictive Sparse Decomposition Method with Densenet for Prediction of Lung Cancer*," *International Journal of Computing*, Vol. 19, No. 4, pp. 533-541, 2020.
[19] I. D. Mienye, G. Obaido, K. Aruleba, and O. A. Dada, "*Enhanced Prediction of Chronic Kidney Disease using Feature Selection and Boosted Classifiers*," in *International Conference on Intelligent Systems Design and Applications*, 2021: Springer, Cham, pp. 527-537, DOI: 10.1007/978-3-030-96308-8_49.
[20] Q.-F. Li and Z.-M. Song, "*High-Performance Concrete Strength Prediction based on Ensemble Learning*," *Construction and Building Materials*, Vol. 324, p. 126694, 2022, DOI: 10.1016/j.conbuildmat.2022.126694.
[21] N. C. Oza and K. Tumer, "*Classifier Ensembles: Select Real-World Applications*," Inf*ormation Fusion*, Vol. 9, No. 1, pp. 4-20, 2008, DOI: 10.1016/j.inffus.2007.07.002.
[22] T. N. AL-Hadidi and S. O. Hasoon, "*Software Defect Prediction using Extreme Gradient Boosting (XGBoost) with Optimization Hyperparameter*," *Al-Rafidain Journal of Computer*

ISSN:2302-8149
e-ISSN:2540-9719

Sciences and Mathematics, Vol. 18, No. 1, pp. 22-29, 2024, DOI: 10.33899/csmj.2023.142739.1081.

[23] A. A. Ibrahim, R. L. Ridwan, M. M. Muhammed, R. O. Abdulaziz, and G. A. Saheed, "*Comparison of the CatBoost Classifier with Other Machine Learning Methods*," *International Journal of Advanced Computer Science and Applications*, Vol. 11, No. 11, pp. 738-748, 2020, DOI: 10.14569/IJACSA.2020.0111190.

[24] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "*CatBoost: Unbiased boosting with Categorical Features*," *Advances in Neural Information Processing Systems*, Vol. 31, 2018.

[25] M. Nagassou, R. W. Mwangi, and E. Nyarige, "*A Hybrid Ensemble Learning Approach Utilizing Light Gradient Boosting Machine and Category Boosting Model for Lifestyle-based Prediction of Type-II Diabetes Mellitus*," *Journal of Data Analysis and Information Processing*, Vol. 11, pp. 480-511, 2023, DOI: 10.4236/jdaip.2023.114025.

[26] A. F. L. Ptr, M. M. Siregar, and I. Daniel, "*Analysis of Gradient Boosting, XGBoost, and CatBoost on Mobile Phone Classification*," *Journal of Computer Networks, Architecture and High Performance Computing*, Vol. 6, No. 2, pp. 661-670, 2024, DOI: 10.47709/cnahpc.v6i2.3790.

[27] D. Uenoyama, H. Yoshiura, and M. Ichino, "*Personal Authentication of Iris and Periocular Recognition using XGBoost*," in *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, Osaka, Japan, 2019: IEEE, pp. 186-187, DOI: 10.1109/GCCE46687.2019.9015469.

[28] Z. A. Ali, Z. H. Abduljabbar, H. A. Tahir, A. B. Sallow, and S. M. Almufti, *"eXtreme Gradient Boosting Algorithm with Machine Learning: A review*," *Academic Journal of Nawroz University*, Vol. 12, No. 2, pp. 320-334, 2023, DOI: 10.25007/ajnu.v12n2a1612.