







Acta Infologica

Research Article

Open Access

Enhancing Software Requirement Classification via Dataset Fusion and Machine Learning



Muhammad Owais Raza ¹  , Vajeiha Mir ² , Jawad Rasheed ^{1,3,4} , Mirsat Yeşiltepe ⁵  & Shtwai Alsubai ⁶ 

¹ İstanbul Sabahattin Zaim University, Department of Computer Engineering, İstanbul, Türkiye

² Mehran University of Engineering and Technology, Department of Software Engineering, Jamshoro, Pakistan

³ Applied Science Private University, Applied Science Research Center, Amman, Jordan

⁴ İstanbul Nisantaşı University, Department of Software Engineering, İstanbul, Türkiye

⁵ İstanbul University, Department of Intelligent Transportation System, İstanbul, Türkiye

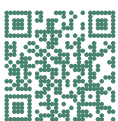
⁶ Department of Computer Science, College of Computer Engineering and Sciences in Al-Kharj, Prince Sattam Bin Abdulaziz University, Al-Kharj, Saudi Arabia

Abstract


Software engineering involves numerous steps; a successful software product follows these guidelines to the core. One such step is gathering requirements for a software product. This step is quite expensive in terms of time and money; a potential solution is to automate the requirement collection process. Automating the process of gathering software requirements requires separating requirements into types. An approach to predict the type of requirement is using text classification and machine learning; however, the problem with this approach is that it requires a large amount of data, which is not available for this use case. In this study, we perform dataset fusion to create a large dataset. We applied vertical fusion, which increased the number of instances in the dataset. Once a fusion-based dataset is created, machine learning algorithms are applied, and based on empirical results, the performance of the machine learning model after fusion drastically improved to 87.78% f1score with support vector machine (SVM). This improvement shows the efficacy of data fusion in improving the performance of a text classifier and demonstrates that it can overcome the limitations of small datasets by combining data from diverse sources. Our study demonstrated the robustness of our approach in software requirement classification by surpassing the highest recall scores from the previous four years, achieving 94.20% with fusion-based SVC and outperforming previous models even in non-fusion settings.

Keywords

Software Engineering · Machine Learning · Software Requirement Engineering · Hybrid Models · Ensemble Modeling



“ Citation: Raza, M. O., Mir, V., Rasheed, J., Yeşiltepe, M. & Alsubai, S. (2025). Enhancing software requirement classification via dataset fusion and machine learning. *Acta Infologica*, 9(1), 275-292. <https://doi.org/10.26650/acin.1634472>

© This work is licensed under Creative Commons Attribution-NonCommercial 4.0 International License. 

© 2025. Raza, M. O., Mir, V., Rasheed, J., Yeşiltepe, M. & Alsubai, S.

✉ Corresponding author: Muhammad Owais Raza 6210024002@std.izu.edu.tr



Introduction

A successful computer program (i.e. Software) is the result of robust software requirements engineering processes; thus, software requirements (SR) are the foundation of software systems (Laplante & Kassab, 2022). The requirement gathering step is one of the most critical and difficult parts of SDLC (The Software Development Life Cycle) (Okesola et al., 2020). Risks associated with software requirements can be predicted using classification algorithms during the requirement collection stage (Handa, Sharma, & Gupta, 2022; Luitel, Hassani, & Sabetzadeh, 2024; Mullis et al., 2024).

There are two types of software requirements: functional requirements (FRs) and nonfunctional requirements (NFRs) (Eckhardt, Vogelsang, & Fernandez, 2016). The FRs are considerably easier to grasp and extract from the user narratives at (requirement gathering process) RGP. Non-functional requirements (NFRs) are also vital; they play a key role in improving product quality and determining the acceptability of a proposed system. Overly high NFRs can rapidly increase costs, whilst low non-functional demands can result in poor user experiences. It has been noted that NFR corrections are far more costly and challenging to perform.

Classifying the SR for each group is challenging, although automated requirement extraction and categorization can save time. It can be achieved through the use of machine learning (ML) (Kaur & Kaur, 2023; Rahman et al., 2024; Rahimi, Eassa, & Elrefaei, 2021), and various ML algorithms find the best results by learning from such data (Textual data). Supervised learning is a suitable choice for categorization problems, such as determining which category the data belong to (Khatian et al., 2021). To predict risks in new software requirements, a dataset that includes software requirements requirements is necessary.

One of the key problems in SR prediction is limited data availability, and the available datasets are relatively small for text classification (Catal & Diri, 2009). To address this issue, a potential solution is dataset fusion i.e. (combining two datasets to either increase number of features or number of instances). Therefore, in this study we are proposing a data fusion technique and apply machine learning to improve the performance of ML models. The main contribution of this study are:

According to the study, data fusion can successfully address the drawbacks of small datasets by merging information from several sources. In this study the Support Vector Classifier (SVC) attains an F1-score of 87.78\% which is quite exceptional for this task.

A thorough comparative analysis was performed to assess how well the machine learning models performed on separate datasets as opposed to the fused datasets and with the literature.

This study outperformed prior research in both fusion and non-fusion contexts, demonstrating consistent superiority in software requirement classification using machine learning models with recall scores of 94.20\% (fusion) and 77.77\% (non-fusion).

Literature Review

. Supervised and unsupervised machine learning techniques have been effectively used to categorize the non-functional requirements of software applications (Handa, Sharma, & Gupta, 2022). They created a model to categorize NFRs from the input text documents, and they evaluated several machine learning techniques using various criteria, including accuracy, recall, performance, and precision (Younas et al., 2020). Suggested a method for recognizing and categorizing NFRs into 14 different groups using an unsupervised machine-learning strategy based on developer- and user-oriented quality factors. Their method demonstrates how unsupervised machine learning can be used to properly categorize NFRs. (Hey et al., 2020) Suggested the

Norbert transfer learning model, which is a developed form of the BERT model. Applying natural language processing techniques, Norbert has demonstrated promising results in categorizing non-functional needs by using the best aspects of the BERT model.

A classification approach based on ontologies was employed to categorize nonfunctional needs into their respective subclasses. This study produced a fresh classifier using a Support Vector Machine (SVM) and a gold standard corpus to automatically classify requirement words from Software Requirement Specifications (SRS). Individuals populate the later ontology by connecting sentences with the correct classes; this allows SPARQL to be used to obtain sentences of a certain class, such as security (Rashwan, Ormandjieva, & Witte, 2013).

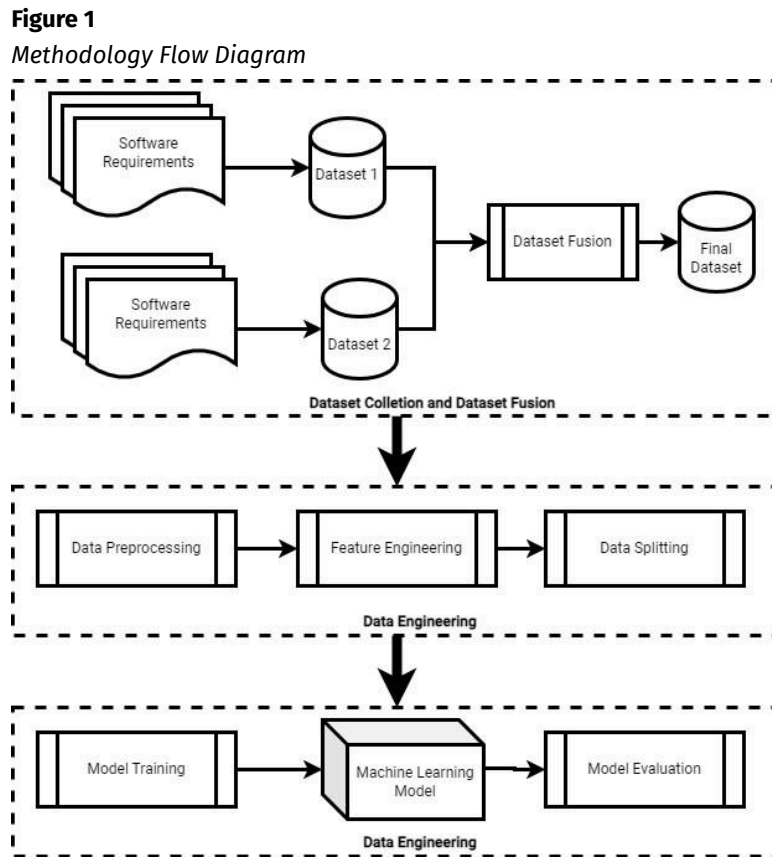
According to a method developed in (Haque, Rahman, & Siddik, 2019), user evaluations of a few app shops, such as Play Store, Apple Store, and others, were performed using 4 feature selection methods. These methods divided the requirements into NFR (Non Functional Requirements), functional requirements, and others. They performed this by combining the ML methods J48, Bagging, and Naive Bayes with the following four classification techniques: When comparing the F1 scores of various permutations of methods and algorithms, they found that the integration of AUR-BoW and bagging produced better results. In addition, they discovered that bagging is more appropriate for classifying NFRs from user feedback than the J48 and Naive Bayes results (Khatian et al., 2021). This research shows that supervised ML algorithms are suitable options for extracting and identifying NFRs in SR. To categorize NFR, empirical research contrasted 4 feature engineering techniques and multiple ML methods (Binkhonain & Zhao, 2019). In (Li & Nong, 2022), NFR from SRE papers was automatically categorized using a DNN model named NFRNet. The network is divided into two sections. To understand the context representations of the requirements descriptions, one method uses enhanced BERT based on ngram masking, and the other method uses a BILSTM classification network together with the context information. Paper (Khan et al., 2023) provides a unique methodology using transfer learning (TL) techniques to perform NFR detection and classification to reduce resource consumption and development time, which will eventually result in increased efficiency in software development. For this purpose, they assessed many cutting-edge pretrained models, such as XLNet, BERT, Distill Roberta, and Electra small (Devlin, Chang, & Toutanova, 2019; Khan et al., 2023).

The goal of this study (García, Fernández-y-Fernández, & Pérez, 2023) is to employ a convolutional neural network to enhance the performance of the categorization of non-functional requirements (NFR). In addition, it aims to demonstrate the significance of preprocessing, sampling strategy implementation, and the utilization of pre-trained matrices like Word2vec, Glove, and Fasttext (García, Fernández-y-Fernández, & Pérez, 2023). . An Agile methodology was explained and proposed for managing nonfunctional needs (Sherif, Helmy, & Galal-Edeen, 2023). The requirements gathering, assessment, documentation, and validation are among the main requirements engineering processes that the framework facilitates. The framework also manages recommendations that are not functional (Sherif, Helmy, & Galal-Edeen, 2023).

Based on the literature, very limited work is done for issue of limited data availability; therefore, in this study, data fusion was performed to create an augmented dataset. This dataset was trained on machine learning models, and the performance of machine learning models trained on individual and fused datasets was compared. Based on the comparison, the usefulness of data fusion for the use case of SR prediction was evaluated.

Methodology

A three-layered methodology was applied in this study. The first layer performs data collection and fusion, the second layer performs data engineering (i.e. cleaning, feature selection and splitting), and the third layer performs machine learning modeling. [Figure 1](#) shows the flow chart diagram of the methodology. In the following section, each step is discussed in detail.



Dataset Collection and Dataset Fusion

The first step in the methodology is the collection of dataset and performing data fusion. For this study, two publicly available datasets were used, and the fusion algorithm was applied to create a new large dataset for better performance. Details regarding datasets and fusion process are described in the following.

The datasets D_1 (Owais, 2025) and D_2 (Owais, 2025) both contain the software requirements in textual for and classification of those requirements into specific classes. D_1 contains 1005 rows, and D_2 has 978 rows. The independent variable in this study was the requirements from the datasets, and the dependent variable was the category or class of the requirement both dataset D_1 , and D_2 contain different numbers of classes, and each dataset is evaluated based on the number of classes it contains. [Table 1](#) lists the classes of requirement in each dataset and their number of instances.

Table 1
Class Distribution of Datasets D_1 and D_2

Requirements	Class Distribution	
	# of Instance D_1	# of instances D_2
F	265	209
SE	147	56
US	118	63
O	96	58
PE	89	54
LF	78	34
A	56	21
MN	51	17
SC	43	21
PO	24	2
FT	23	10
FR	N/A	312
NFR	N/A	110
L	15	10

The class distribution for D_1 and D_2 is shown in Table 1, with D_1 exhibiting a greater diversity of cases. The majority of classes, including PO ($D_1:24, D_2:2$), have fewer instances in D_2 even though F ($D_1:265, D_2:209$) is still the greatest in both classes. Indicating dataset differences, FR ($D_2:312$), NFR ($D_2:110$), and L ($D_2:10$) are unique to D_2 .

There are two primary ways to fuse the dataset, vertical and horizontal fusion. In horizontal dataset fusion, researchers merge two datasets side by side by aligning them on a common key or index. In horizontal dataset fusion, we merged two datasets side by side by aligning them on a common key or index.. The horizontal fusion assumption is This assumes that datasets D_1 and D_2 have the same number of rows and are aligned on a common key or index. As this type of fusion is not possible for this case, the dataset in this study do not coincide in terms of features but instances; thus, instances need to be merged not the features so that there is another type of fusion called vertical fusion, which is used in this study. In vertical data fusion, two datasets are combined by stacking one over another. The assumption in this type is that both datasets have similar columns. In this case, vertical data fusion is used. The mathematical representation of vertical data fusion is given as follows. D_1 and D_2 have m and n rows, respectively, and mathematically, it is defined as follows:

$$D_1 = \{(x_{11}, x_{11}, \dots, x_{1k}), \dots, (x_{m1}, x_{m2}, \dots, x_{mk})\} \tag{1}$$

$$D_2 = \{(y_{11}, y_{11}, \dots, y_{1k}), \dots, (y_{n1}, y_{n2}, \dots, y_{nk})\} \tag{2}$$

The vertical dataset fusion is performed by obtaining the union between the rows of both datasets, which is represented in this study as D_{vf} i.e. vertically fused dataset. Mathematically, it is expressed as follows.

$$D_{vf} = D_1 \cup D_2 \tag{3}$$

$$D_{vf} = \{(x_{11}, x_{11}, \dots, x_{1k}), \dots, (x_{m1}, x_{m2}, \dots, x_{mk})\}, \{(y_{11}, y_{11}, \dots, y_{1k}), \dots, (y_{n1}, y_{n2}, \dots, y_{nk})\} \tag{4}$$



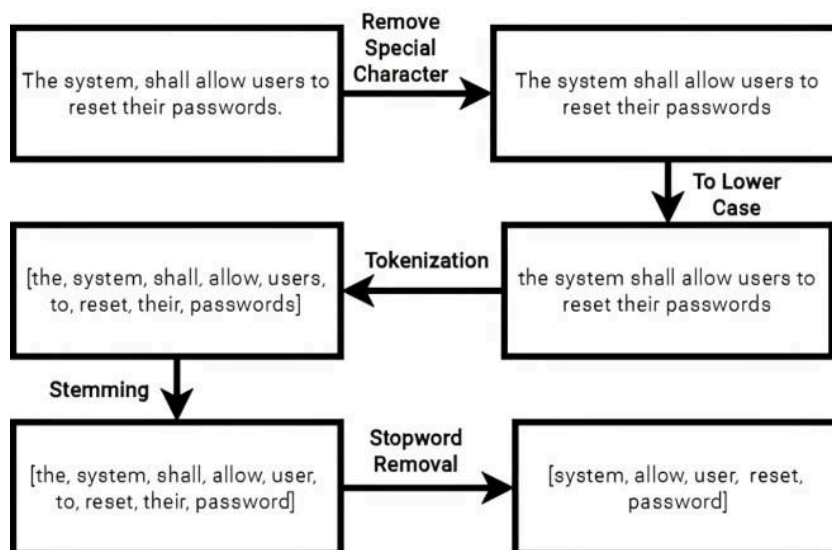
Vertical data fusion is only possible when different datasets have consistent and compatible column structures. In this study, vertical data fusion combined two datasets to satisfy software requirements to form a large dataset. This increases the sample size and improves the performance of the machine learning models.

In this study, before feature extraction and classification by vertically concatenating the datasets D_1 and D_2 . This method guarantees that both dataset's original feature space of both datasets is preserved while using all available data. This allows the model to learn intricate relationships between datasets. This research approach provides a deeper understanding of the best fusion strategies for enhancing model accuracy..

Data Engineering

The second layer of the methodology involves data preprocessing, feature engineering, and training test split. The following subsection elaborates each step in detail. In the data preprocessing stage, the first thing that occurs is that special characters are removed from the text; only letters are exits in the text from this step forward. Once only the English letter exists, the next step is to convert it into lower case letters for consistency. After making the text consistent throughout the dataset, the next step is tokenization, which involves breaking sentences into words. The next step is to perform stemming. Stemming converts all variation of words into root words, which makes the text consistent and reduces the dimensionality of the data. In the last preprocessing step all the stopwords from the text are removed. Stopwords are common words like 'a', 'the' which are in huge amounts but contribute very little during the classification process. **Figure 2** shows the complete life cycle of preprocessing a text.

Figure 2
Preprocessing Step During Data Engineering Process



Once data is preprocessed and next step is to perform feature engineering. This is a crucial step in the text classification machine learning pipeline. In this study TF-IDF (Term Frequency-Inverse Document Frequency) vectorization was used. It converts textual data (i.e software requirements) into numerical features that can be fed to machine learning algorithms. Mathematically TF-IDF is shown in equation 5.

$$TF - IDF(t, d) = TF(i, d) \times IDF(t) \tag{5}$$



$$TF(t, d) = \frac{\text{No of times term } t \text{ appears in document } d}{\text{Total No of terms in document } d} \quad (6)$$

$$TF(t, d) = \log \frac{\text{Total No documents}}{\text{No of documents containing term } t} \quad (7)$$

TF-IDF improves classification performance by incentivizing informative features and down-weighting common but insignificant ones. Because it captures the importance of features well, especially in text-based or high-dimensional datasets, this approach was selected.

For machine learning modeling and evaluation phase two sets of datasets are required, one for training and the other for testing or validation. In this study, a stratified split was used to break the dataset into training and testing sets. The split ratio used in this study is 80%-20%, 80% is the part of dataset used for model training, and the remaining 20% of the dataset is used for testing the model. The specific training and testing details are given in the next section.

Modeling and Evaluation

In this phase, models are developed using the training set, which comprises 80% of the dataset, and the remaining 20% of the dataset is used for testing the model. The models were tested on various performance metrics discussed as follows. We use a different of ML algorithms to build predictive engines during this phase. The training dataset was used to fit the models. To identify various patterns and relationships in the textual data (Software Requirements in this case). Python is the programming language used to implement the machine learning models in this study. Scikit-learn (sklearn) is a key library used for hyperparameter tuning and model building, training, and evaluation. Numpy manages array operations and numerical computations, and pandas handles data handling and preparation. The Matplotlib tool is used to build visualizations showing data distributions and model performance. The scikit-learn methods were used to compute the performance metrics, which include accuracy, precision, recall, and F1-score. The following are the algorithms used in this study:

Logistic Regression (LR): The LR model is a statistical machine learning model that uses logistic regression as its dependent variable. Using logistic regression, data and the relationship between one or more independent variables and dependent variables are described. Nominal, ordinal, or interval variables could be independent variables. Logistic regression is a term that comes from the concept of a logistic function. The logistic function is sometimes referred to as the sigmoid function. The range of values for this logistic function is 0–1. A logistic function is used by a binary classification linear model (LR) to predict the probability of a binary outcome (Dias Canedo and Cordeiro Mendes (2020).

Support Vector Machine (SVM): SVM is a popular ML algorithm. SVM algorithms find the best decision boundary to break the N-dimensional vector space into target labels or classes. This approach makes it easy for SVM to classify new data instances in the future. A hyperplane is a term used to describe this ideal decision boundary. SVM chose the extreme points and vectors in order to construct the hyperplane. According to (Wang, & Wang 2023), SVM works well for datasets with distinct margins of separation and in higher dimensions.

K-Nearest Neighbor (KNN): A KNN is a basic ML algorithm based on a supervised ML approach. The KNN algorithm takes the assumption that the new instances and previous data instances are similar for the existing cases, and it assigns the new instance to the label that is most similar to the existing labels. The KNN algorithm saves the available data and then employes similarity matrices to classify a new data instance.

This KNN algorithm simplifies the process of assigning recently discovered data to an appropriate category. When working with large datasets, a processor can demand substantial processing power despite being simple to use (Syriopoulos et al., 2023).

Decision Tree (DT): Regression problems can be solved using DT; however, classification problems are the main application of these supervised learning techniques. The features of a dataset are represented by internal nodes in this tree-structured classifier, the decision rules are represented by branches, and the outcome is represented by each leaf node. There are two nodes in a DT: the decision node and the leaf node. The result of these choices is a leaf node, which does not have further branches, unlike decision nodes that have multiple branches and are used to make specific decisions based on feature values. The proposed method manages categorical and numerical data and is easy to understand (Yuvaraj et al., 2021).

Random Forest (RF): DTs are used in RF, also called random decision forest, a supervised machine learning algorithm for classification, regression, and other applications. To generate a set of DTs, a random subset of the training set was selected. The votes from these decision trees are then tallied to determine the ultimate prediction. It reduces overfitting and enhances predictive performance by using majority voting (Yuvaraj et al., 2021).

Gaussian Naive Bayes (GB): A subset of Naive Bayes called GB accounts for continuous attributes and distributes data features across the dataset in a Gaussian manner. GB is a classification algorithm that uses continuous normally distributed features and is based on the Naive Bayes algorithm, according to the Sklearn library. It works effectively and efficiently on large datasets (Luo, 2021).

AdaBoosting (AB): AdaBoosting is an ML technique that employs an ensemble learning approach. Using the training dataset, it iteratively trains a weak classifier. AB's basic principle is that each succeeding classifier assigns more weight to incorrectly classified data points. The final AdaBoost model combines all of the training weak classifiers (Carreira-Perpiñán & Zharmagambetov, 2020).

Gradient Boosting (GRB): GRB is a very strong algorithm that transforms a number of weak classifiers or regressors into strong classifiers or regressors ones. GRB minimizes the loss function, for example the MSE (mean squared error or cross-entropy) of the previous models, and gradient descent is used to train each new model. The algorithm first calculates the gradient of the loss function in relation to the current ensemble's predictions for each epoch, after which it trains a new weak model to minimize the gradient. (Burhan & Ali, 2023).

Once the models were trained on the training set of the split test set, the next step was to evaluate the performance of the model in the test set. The instances in the test sets are new to the model i.e. data not seen by the model. Numerous performance metrics are available for classification problems; however, the selected metrics for this study are accuracy, precision, recall, and the F1 Score. Each of these metrics is described as follows.

Accuracy: Accuracy is the ratio of True Positive (TP) and True Negative TN to all instances (TP, TN, FP, FN). The result is shown in equation 8:

$$\text{Accuracy} = \frac{T_p + T_n}{T_p + T_n + F_p + F_n} \quad (8)$$

Precision: The ratio of True Positive (TP) to all positive instances (TP and FP) is referred to as precision. Equation 9 represents precision

$$\text{Precision} = \frac{T_p}{T_p + F_p} \tag{9}$$

Recall: Recall is the ratio of True positive to True positive and True negative; Equation 10 represents recall mathematically.

$$\text{Accuracy} = \frac{T_p}{T_p + F_n} \tag{10}$$

F1 Score: The F1 score is the harmonic mean of precision and recall and is considered a robust metric to evaluate a classification task. Equation 11 represents F1 Score.

$$\text{Accuracy} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{11}$$

Results

Table 2 shows the accuracy, precision, and recall for all 8 algorithms over the three datasets (D_f , D_1 , and D_f) considered in this study. For D_1 , the best performing algorithm was gradient boosting, which achieved accuracy, precision, and recall of 78.11%, 65.24%, and 76.87%, respectively, followed by RF and GB with accuracy of 78% and 76%. Gradient Boosting also provided the best results for D_2 with an accuracy of 70.92%. The second best model for D_2 is Naive Bayes with an accuracy of 70.92%. The top three algorithms for D_f were SVC, RF, and GRB with accuracy scores of 89.67%, 89.67%, and 89.17%. respectively. Overall, these accuracies were obtained for D_f because the overall number of instances of D_f is twice the number of rows in D_1 and D_2 .

Table 2
Performance measure of different datasets with different algorithms

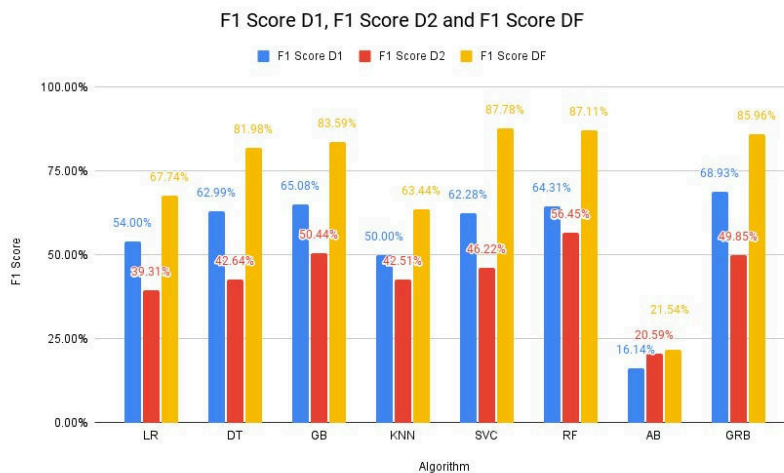
Accuracy-precision recall			
D1			
Algorithm	Accuracy	Precision	Recall
LR	70.65%	49.37%	64.70%
DT	71.14%	66.67%	63.11%
GB	76.62%	63.65%	69.52%
KNN	66.67%	50.15%	53.77%
SVC	73.63%	55.71%	77.77%
RF	78.11%	63.20%	74.88%
AB	33.83%	17.66%	23.57%
GRB	78.11%	65.24%	76.87%
D2			
LR	69.39%	37.53%	48.67%
DT	70.41%	40.11%	48.95%
GB	70.92%	49.30%	57.69%
KNN	70.41%	41.79%	54.27%
SVC	69.90%	41.07%	69.75%
RF	72.96%	50.84%	77.78%
AB	54.08%	22.45%	28.05%
GRB	70.92%	46.70%	57.18%



Accuracy–precision recall			
Df			
LR	79.60%	62.29%	82.73%
DT	85.64%	83.05%	82.13%
GB	84.38%	82.65%	85.27%
KNN	73.30%	63.36%	68.22%
SVC	89.67%	83.86%	94.20%
RF	89.67%	83.80%	92.35%
AB	40.55%	22.83%	29.76%
GRB	89.17%	83.61%	89.54%

Figure 3 shows a bar chart that compares the F1 Scores of all algorithms across datasets D_1 , D_2 , and D_f . For D_1 , GRB is able to achieve the best performance with an F1 score of 68.93%, although in case D_2 , RF outperformed the algorithm with an F1 score of 56.45%. For D_f , the best performing algorithm was SVC with a top score of 87.78%. AdaBoost consistently received the lowest scores in every scenario (D_1 , D_2 and D_f), indicating notable performance disparities among the algorithms. SVC, RF, and GRB consistently performed well across all datasets, especially on D_f . The superior performance of all algorithms on D_f was due to the increased number of instances due to data fusion.

Figure 3
F1 Score Comparison Between Different Datasets



The confusion matrix shows the number of correct predictions per class. This indicates how well or poorly a classifier performs in a given class. Figures 4, 5, and 6 show the confusion matrices for all the algorithms on dataset D_1 , D_2 , and D_f . The diagonal represents the correct prediction rest shows the incorrect predictions. The classes with smaller number of instances have poor performance. Classes with a large number of instances demonstrate better performance.

Based on Figure 4, the best performing algorithms for dataset D_1 are random forest and gradient boosting because they have the highest number of predictions along the diagonal and every small number of predictions outside the diagonal. The KNN shows poor performance across all classes for dataset D_1



Figure 4
Confusion Matrices For D_1

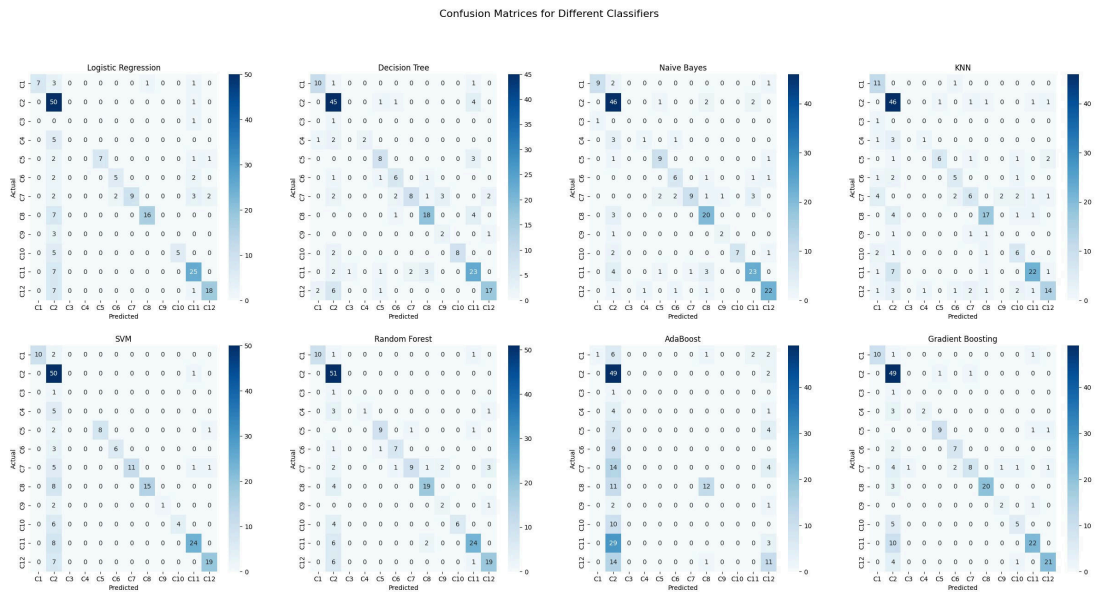


Figure 5 shows confusion matrices for all algorithms for D_2 . The results indicate that the Random Forest and Gradient Boosting classifiers perform fairly well, especially for classes 3 and 4, as these have higher values along the diagonal that point to higher classification accuracy for these classes. The SVC and logistic regression classifiers exhibit comparable performance and higher accuracy, especially regarding class 3 and class 6 predictions. Compared to the other models, the Decision Tree, Gaussian Naive Bayes, and AdaBoost models exhibit higher rates of misclassification across several classes.

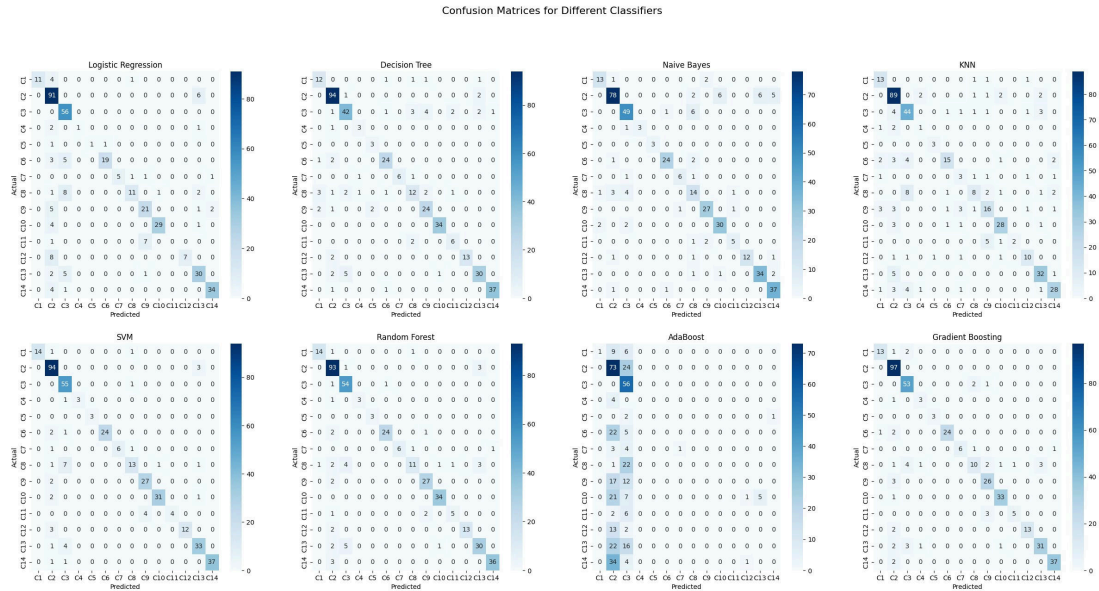
Figure 5
Confusion Matrices For D_2



Figure 6 shows the confusion matrices for the fused dataset D_f . As shown in the results, gradient boosting focuses more on correct predictions overall and for key classes such as 12. In addition to gradient boosting,

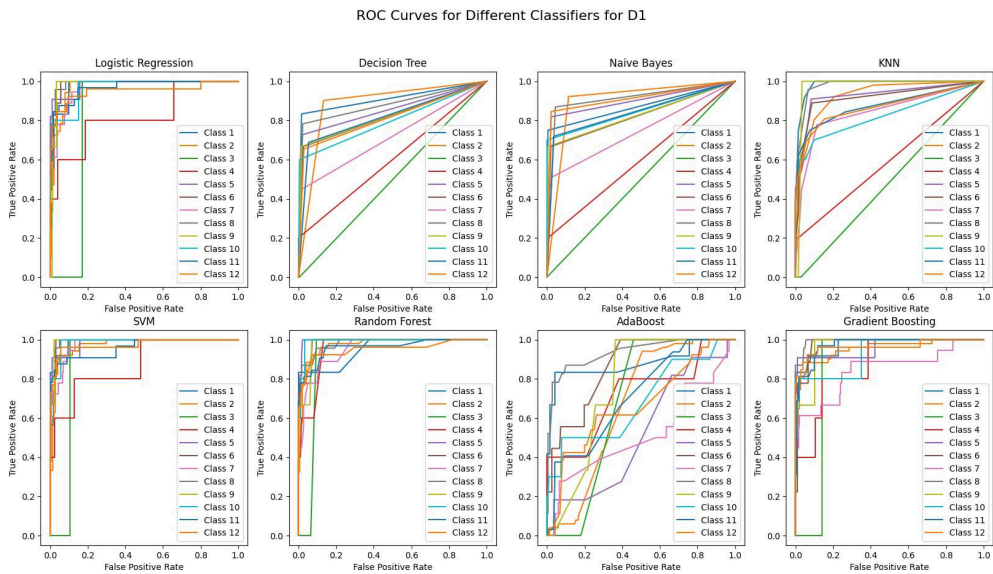
random forest, logistic regression, KNN, SVC, also show higher level of performance. But model such as DT and GB show misclassification and hence poor classification accuracy. Compared to D_1 , D_2 , and D_f provided superior accuracy for all classes.

Figure 6
Confusion Matrices For D_f



Figures 7, 8, and 9 show the ROC curve for all algorithms used in this study for all datasets D_1 , D_2 , and D_f . If the line on the ROC AUC curve follows a straight diagonal line, then the model has no ability to discriminate between right and wrong predictions. Results in **Figure 7** show that models such as GB, LR, and RF have good predictive ability as their curves follow a solid convex curve with a higher AUC score for all classes. Although model such as KNN lack the ability to predict the curve. Through out the algorithms, class 3 is difficult to predict because its curve in some models is closer to a diagonal line.

Figure 7
ROC curve for D_1



Similar to Figure 7 and Figure 8 follows the same convex curve pattern for most of the algorithms, including the LR, SVC, GB, and RF algorithms. The models with weaker ability to predict and differentiate among classes include KNN and GNB with lesser AUC scores and lower convex curves

Figure 8
ROC curve for D_2

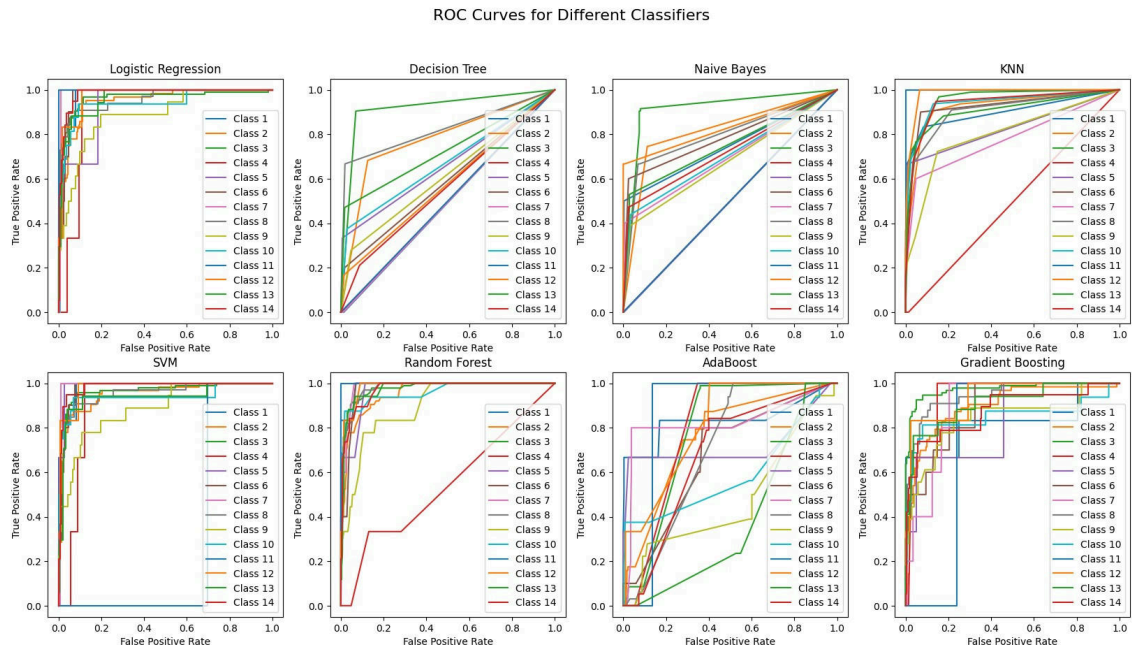
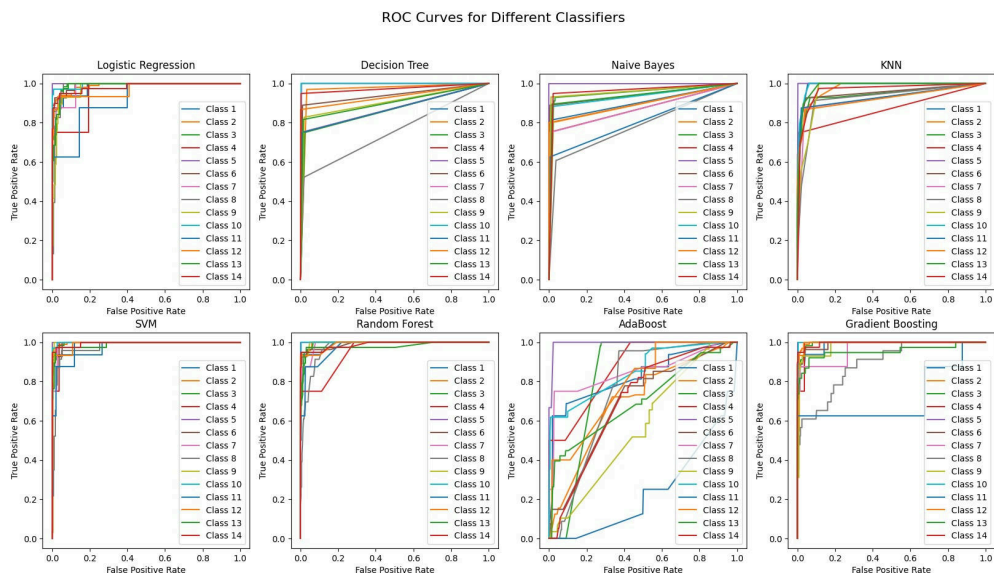


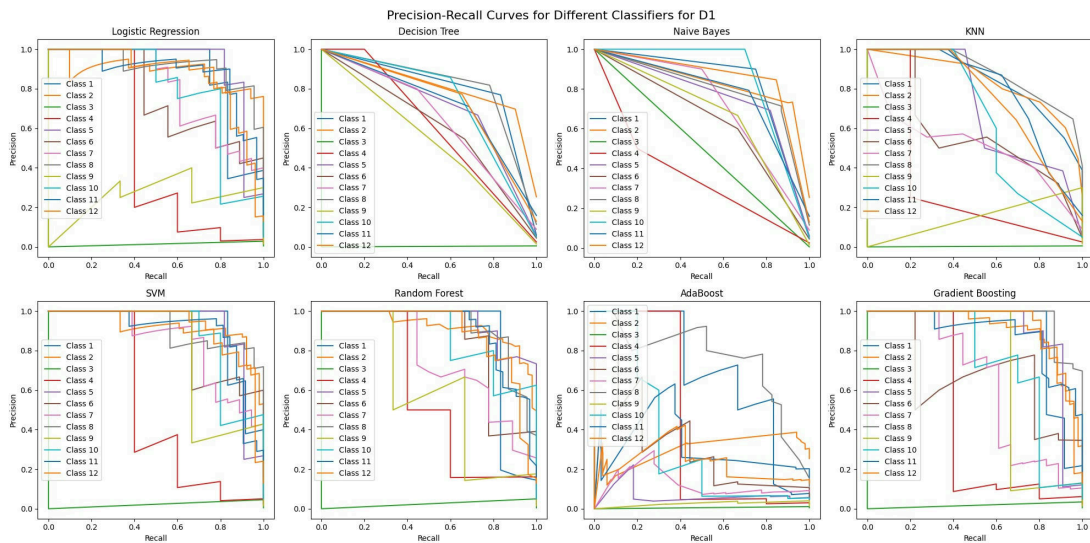
Figure 9 shows the models developed using D_f , which exhibit superior performance compared to models developed on D_1 and D_2 . The curve in Figure 11 show higher AUC score and a higher convex curve. These results are evident from the confusion matrix and other performance evaluation parameters.

Figure 9
ROC Curve For D_f



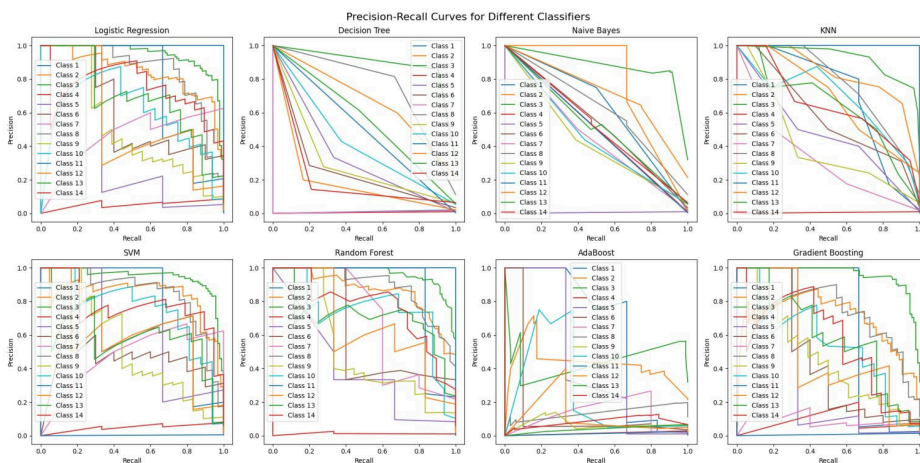
Figures 10, 11, and 12 show the precision recall curves for D_1 , D_2 and D_f using all the algorithms discussed in this study. A precision recall curve represents the trade-off between precision and recall i.e. sensitivity under varying thresholds. A good precision recall curve has higher precision at higher recall scores, which also results in a higher area under the curve i.e. AUC. The precision-recall curves in Figure 10 demonstrate the varying performance among the classifiers. Here, RF, LR, SVC, and GB exhibit the most consistent performance, and AB continues to perform the worst across classes. Class 3 has proved to be difficult to predict and exhibits extremely poor performance across multiple classifiers for which the AUC for class 3 is between 0.00 and 0.02. Most of the other classes continued to perform well, with an AUC greater than 0.80.

Figure 10
Precision recall curve for D_1



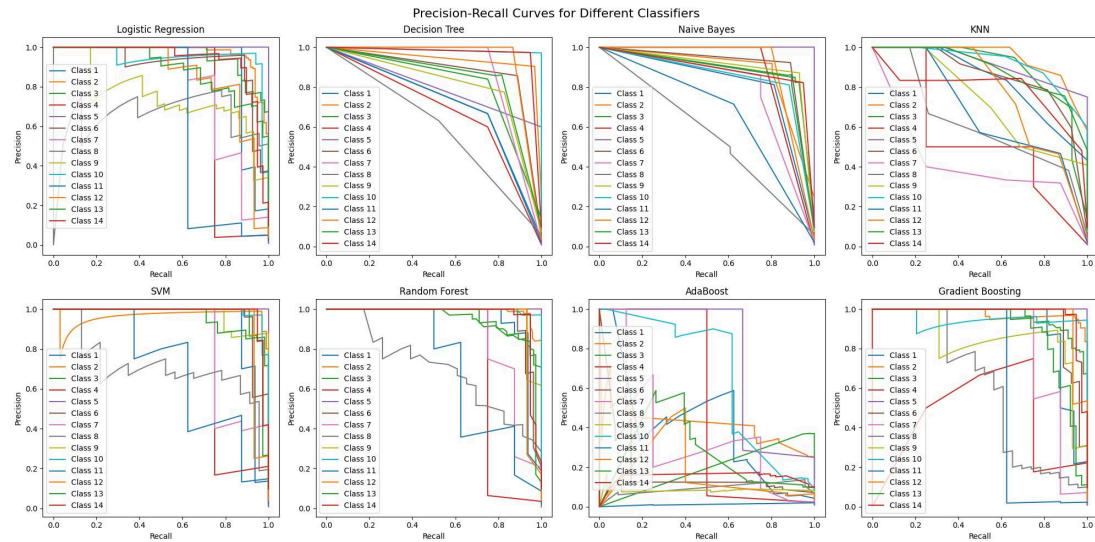
Charts in Figure 11 shows the precision-recall curve for D_2 . Based on the graphs, it shows a degradation in model performance compared to D_1 . The RF and GB algorithms maintained relatively better performance than the other algorithms on this dataset. For D_2 , different classes, such as 4, 5, and 11, showed lower AUC scores across all classifiers, some with an AUC score of 0.50. Overall, AB proved to be the worst performing algorithm.

Figure 11
Precision recall curve for D_2



Precision-recall curves in Figure 12 shows superior performance to those of both D_1 and D_2 . The curves in Figure 12 shows higher degree of performance across most classifiers, with particularly high AUC scores for GB, RF, LR, and SVC, with many classes achieving AUC values greater than 0.90. However, AB still demonstrated poor performance with several classes having very low AUC scores (less than 0.30).

Figure 12
Precision recall curve for D_f



The models evaluated in this study performed better than the highest recall scores documented literature from last four years, it is shown in Table 3. The recall metric was selected as the main evaluation criterion due to its crucial role in reducing false negatives, which is a prerequisite for accurately classifying software requirements. To ensure a consistent basis for comparison, only machine learning models were used in the previous studies and the current assessment. The efficacy of the proposed strategy is demonstrated by the improved performance of the proposed models in both fusion and nonfusion settings. Under unfusion, our SVC, GBR, and RF outperformed the existing models with recall of 77.77%, 76.87%, and 74.88%, respectively, and under fusion settings, the SVC outperformed all the models by a huge margin with a recall score of 94.20%.

Table 3
Performance measure of different datasets with different algorithms

Study	Approach	Recall Scores	
(Dias Canedo & Cordeiro Mendes, 2020)	SVM	73%	(Dias Canedo & Cordeiro Mendes, 2020)
Multinomial Naive Bayes (MNB)	77%	(Binkhonain & Zhao, 2023)	Hierarchical Classification for Requirements Classification (HC4RC)
64%	This Study	SVC	77.77%
This Study	GRB	74.88%	This Study
RF	76.87%	This Study	SVC (Fused)
94.24%			

Conclusion

Software systems are based on requirements. The requirements phase of the software development process sets reasonable objectives that must be achieved. The most important and challenging phase of the SDLC (Software Development Life Cycle) is gathering requirements. In the age of automation, the process of requirement gathering need to automatics as well. One of the key features of automation is the recognition of the type of requirements. The data available for this task are very limited. Therefore, in this study, we perform data fusion to create a new dataset from existing one with more instances. Based on the results, data fusion proved to be effective. Separately D_1 and D_2 had an F1 score of 68.93% and 56.45%, respectively, but after fusion, the F1 score improved to 87.78%. In both the fusion and non-fusion scenarios, the models proposed in this study performed better than previous research. SVC achieved 94.20% recall in fusion and 77.77% in non-fusion, demonstrating that our methodology improved the classification of software requirements. The major limitation of this study was the lack of resources available for performing the classification task; thus, performing fusion would not scale. One potential future work of this study to combine more datasets and determine if fusion is as effective as shown in this study, or adding more datasets will alter the improvement.



Peer Review	Externally peer-reviewed.
Author Contributions	Conception/Design of Study- M.O.R., V.M., J.R.; Data Acquisition- M.O.R., V.M., J.R.; Data Analysis/Interpretation- M.O.R., V.M., J.R.; Drafting Manuscript- M.Y., J.R., M.O.R.; Critical Revision of Manuscript- M.Y., J.R., M.O.R., S.A.; Final Approval and Accountability- J.R., S.A.; Technical or Material Support- M.O.R., V.M., J.R.; Supervision- M.Y., J.R., S.A.
Conflict of Interest	The authors have no conflict of interest to declare.
Grant Support	The authors declared that this study has received no financial support.

Author Details


Muhammad Owais Raza

¹ İstanbul Sabahattin Zaim University, Department of Computer Engineering, İstanbul, Türkiye

 0000-0002-3065-385X  6210024002@std.izu.edu.tr

Vajeaha Mir

² Mehran University of Engineering and Technology, Department of Software Engineering, Jamshoro, Pakistan


 0009-0009-4935-9042

Jawad Rasheed

¹ İstanbul Sabahattin Zaim University, Department of Computer Engineering, İstanbul, Türkiye


³ Applied Science Private University, Applied Science Research Center, Amman, Jordan

⁴ İstanbul Nisantası University, Department of Software Engineering, İstanbul, Türkiye

 0000-0003-3761-1641

Mirsat Yeşiltepe


⁵ İstanbul University, Department of Intelligent Transportation System, İstanbul, Türkiye

 0000-0003-4433-5606



Shtwai Alsubai

⁶ Department of Computer Science, College of Computer Engineering and Sciences in Al-Kharj, Prince Sattam Bin Abdulaziz University, Al-Kharj, Saudi Arabia

 0000-0002-6584-7400

References

- Ali, Z. H., & Burhan, A. M. (2023). Hybrid machine learning approach for construction cost estimation: Evaluation of the extreme gradient boosting model. *Asian Journal of Civil Engineering*, 24(7), 2427–2442, 2017.
- Binkhonain, M., & Zhao, L. (2023). A machine learning approach for hierarchical classification of software requirements. *Machine Learning with Applications*, 12, 100457. <https://doi.org/10.1016/j.mlwa.2023.100457>.
- Binkhonain, M., & Zhao, L. (2019). A review of machine learning algorithms for the identification and classification of non-functional requirements. *Expert Systems with Applications: X*, 1, 100001.
- Catal, C., & Diri, B. (2009). We investigate the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problems. *Information Sciences*, 179(8), 1040–1058.
- Carreira-Perpiñán, M. Á., & Zharmagambetov, A. (2020). *The ensembles of bagged TAO trees consistently improved over random forests, AdaBoost, and gradient boosting*. In Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference (pp. 35–46).
- Devlin, J., Chang, M.-W., & Toutanova, K. (2019). *BERT: Pre-training of deep bidirectional transformers for language understanding*. In Proceedings of NAACL-HLT (p. 2). Minneapolis, MN, USA.
- Dias Canedo, E., & Cordeiro Mendes, B. (2020). Software requirements classification using machine learning algorithms. *Entropy*, 22(9), 1057. <https://doi.org/10.3390/e22091057>.
- Eckhardt, J., Vogelsang, A., & Fernandez, D. M. (2016). *Are non-functional requirements really non-functional? An investigation of non-functional requirements in practice*. In Proceedings of the 38th International Conference on Software Engineering (pp. 832–842). Austin, TX, USA.
- García, S. M., Fernández-y-Fernández, C. A., & Pérez, E. R. (2023). Classification of non-functional requirements using convolutional neural networks. *Programming and Computer Software*, 49(8), 705–711.
- Handa, N., Sharma, A., & Gupta, A. (2022). Framework for prediction and classification of non-functional requirements: A novel vision. *Cluster Computing*, 25(2), 1155–1173.
- Haque, M. A., Rahman, M. A., & Siddik, M. S. (2019). *Non-functional requirements classification with feature extraction and machine learning: An empirical study*. In 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT) (pp. 1–5). Dhaka, Bangladesh.
- Hey, T., Keim, J., Koziol, A., & Tichy, W. F. (2020). *Norbert: Transfer learning for requirements classification*. In 2020 IEEE 28th International Requirements Engineering Conference (RE) (pp. 169–179). Zurich, Switzerland.
- Khan, M. A., Khan, M. S., Khan, I., Ahmad, S., & Huda, S. (2023). *Non-functional requirements identification and classification using transfer learning model*. IEEE Access.
- Khatian, V. M., Arain, Q. A., Alenezi, M., Raza, M. O., & Shaikh, F. (2021). *Comparative analysis for predicting non-functional requirements using supervised machine learning*. In 2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA) (pp. 7–12). Riyadh, Saudi Arabia.
- Kaur, K., & Kaur, P. (2023). *Improving the BERT model for requirements classification using a bidirectional LSTM-CNN deep model*. Computers and Electrical Engineering.
- Laplante, P. A., & Kassab, M. (2022). *Requirement engineering of software and systems*. Boca Raton, FL: Auerbach Publications.
- Li, B., & Nong, X. (2022). Automatically classifying non-functional requirements using a deep neural network. *Pattern Recognition*, 132, 108948.
- Luitel, D., Hassani, S., & Sabetzadeh, M. (2024). Improving requirements completeness: Automated assistance through large language models. *Requirements Engineering*, 29(1), 73–95.
- Luo, X. (2021). Efficient English text classification using selected machine learning techniques. *Alexandria Engineering Journal*, 60(3), 3401–3409.



- Mullis, J., Chen, C., Morkos, B., & Ferguson, S. (2024). Deep neural networks in natural language processing to classify requirements by origin and functionality: Application of BERT in system requirements. *Journal of Mechanical Design*, 146(4), 041401.
- Okesola, O. J., Adebisi, A. A., Owoade, A. A., Adeaga, O., Adeyemi, O., et al. (2020). Software requirement in iterative SDLC model. In *Intelligent algorithms in software engineering: Proceedings of the 9th computer science on-line conference 2020* (pp. 26–34). Cham, Switzerland: Springer; 2020.
- Raza, O. "Owais4321/software-requirement-dataset: Dataset for Software Requirement Engineering. dataset, *Zenodo*, Apr. 17, 2025, doi:10.5281/zenodo.15235904.
- Rahman, K., Ghani, A., Misra, S., & Rahman, A. U. (2024). A deep learning framework for non-functional requirement classification. *Scientific Reports*, 14(1), 3216.
- Rahimi, N., Eassa, F., & Elrefaei, L. (2021). One-and two-phase software requirement classification using ensemble deep learning. *Entropy*, 23(10), 1264.
- Rashwan, A., Ormandjieva, O., & Witte, R. (2013). *Ontology-based classification of non-functional requirements in software specifications: A new corpus and SVM-based classifier*. In 2013 IEEE 37th Annual Computer Software and Applications Conference (pp. 381–386). Kyoto, Japan.
- Sherif, E., Helmy, W., & Galal-Edeen, G. H. (2023). *Proposed framework to manage non-functional requirements in agile*. *IEEE Access*, 11, 53995–54005.
- Syriopoulos, P. K., Kalampalikis, N. G., Kotsiantis, S. B., & Vrahatis, M. N. (2023). kNN classification: A review. *Annals of Mathematics and Artificial Intelligence*, 1–33.
- Wang, H., Li, G., & Wang, Z. (2023). Fast SVM classifier for large-scale classification problems. *Information Sciences*, 642, 119136.
- Yuvaraj, N., Chang, V., Gobinathan, B., Pinagapani, A., Kannan, S., Dhiman, G., & Rajan, A. R. (2021). Automatic detection of cyberbullying using multifeature artificial intelligence with deep decision tree classification. *Computers and Electrical Engineering*, 92, 107186.
- Younas, M., Jawawi, D. N., Ghani, I., & Shah, M. A. (2020). Extraction of non-functional requirement using semantic similarity distance. *Neural Computing and Applications*, 32, 7383–7397.

