

Software Functional Requirements Classification Using Ensemble Learning

Sanidhya Vijayvargiya[1(✉)], Lov Kumar[1], Aruna Malapati[1], Lalita Bhanu Murthy[1], and Sanjay Misra[2]

[1] BITS-Pilani Hyderabad, Hyderabad, India
{f20202056,lovkumar,arunam,bhanu}@hyderabad.bits-pilani.ac.in
[2] Østfold University College, Halden, Norway
sanjay.misra@covenantuniversity.edu.ng

Abstract. Software requirement classification is crucial in segregating the user requirements into functional and quality requirements, based on their feedback or client demand. Doing so manually is time-consuming and not feasible. This can lead to delays in satisfying the requirements which in turn can lead to unhappier clients and users. Thus, machine learning techniques are used to optimize this task. In this work, five different word embedding techniques have been applied to the functional and non-functional (quality) software requirements. SMOTE is used to balance the numerical data obtained after word embedding. Dimensionality reduction and feature selection techniques are then employed to eliminate redundant and irrelevant features. Principal Component Analysis (PCA) is used for dimensionality reduction, and Rank-Sum test (RST) is used for feature selection. The resulting vectors are fed as inputs to eight different classifiers- Bagged k-Nearest Neighbors, Bagged Decision Tree, Bagged Naive-Bayes, Random Forest, Extra Tree, Adaptive Boost, Gradient Boosting, and a Majority Voting ensemble classifier, with Decision Tree, k-Nearest Neighbors, and Gaussian Naive Bayes. The experimental results suggest that the combination of word embedding and feature selection techniques with the various classifiers are successful in accurately classifying functional and quality software requirements.

Keywords: Functional requirements · Non-functional requirements · Ensemble learning

1 Introduction

Software requirements are the conditions that a system must meet in order to fulfill a contract or meet a standard. They can be classified into two types- functional and non-functional. Functional requirements are the demands that the end-user specifies as essential features that the system should provide. These are described as operation to be conducted, input to be delivered to the system, and expected output. Non-functional or quality requirements are the basic quality criteria that the system must meet, which include aspects such as portability,

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13381, pp. 678–691, 2022.
https://doi.org/10.1007/978-3-031-10548-7_49

security, maintainability, reliability, etc. As the software requirements increase for larger projects, manual classification becomes a tedious task requiring domain experts. Using natural language processing to classify the requirements helps streamline this process. A critical challenge that needs to be addressed is the inconsistency between terminologies used by stakeholders and requirements engineers, which leads to misclassifications [1]. Our aim in this paper is to create highly accurate software requirements classification models while analyzing-

- The word embedding techniques that best suit the task of software requirements classification using Word2Vec (w2v), Term Frequency-Inverse Document Frequency (TF-IDF), Global vectors for word representation (GloVe), Skip-Gram (SKG), and Continuous bag of words (CBOW).
- The impact of class balancing technique, SMOTE, on the classification compared to models trained without SMOTE.
- How feature selection and dimensionality reduction techniques like Rank-Sum Test (RST) and Principal Component Analysis (PCA) respectively impact the accuracy of the classification models compared to the set of the original features.
- The performance of eight different classifiers: Bagged k-Nearest Neighbors (Bag-kNN), Bagged Decision Tree (Bag-DT), Bagged Naive-Bayes (Bag-NB), Random Forest (RF), Extra Tree (EXTR), Adaptive Boost (AdaB), Gradient Boosting (GraB), and Majority Voting ensemble classifier (MVE).

We perform the above analysis by comparing the performance of models using accuracy, F-measure, and Area under ROC curve (AUC). We use the Friedman test to evaluate if the ML techniques used have a significant effect. The PROMISE dataset, which contains 625 labeled requirements taken from a group of 15 projects, is used in this investigation.

The primary contribution of this paper is to provide a thorough comparison of various ML techniques and algorithms for each step of the software requirements classification pipeline, starting from the feature extraction step, to the classification step. These comparisons, which are performed using AUC, F-measure, and accuracy as the performance metrics, help analyze the impact of the most commonly used ML techniques on the performance and single out those that work best for software requirements. Statistical testing in the form of Friedman test is used to further validate the findings.

The rest of the paper is organized as follows: Literature Review on software requirements classification and various word embedding techniques is presented in Sect. 2. The experimental dataset collection, as well as the various ML techniques used, are described in Sect. 3. Section 4 details the research methodology through an architecture framework. The findings of the experiments, as well as their analysis, are reported in Sect. 5. The comparative study of models generated using various word-embedding techniques, sets of features, and machine learning models is described in Sect. 5. Finally, Sect. 6 wraps up the material presented and suggests research directions for future studies.

2 Related Work

Researchers have previously explored the effectiveness of different classifiers in segregating functional and non-functional requirements. The impact of various word embedding techniques has also been studied, and the key conclusions of the research are discussed in this section. Tiun et al. [8] proposed models based on classifiers like Logistic Regression, Support Vector Machine (SVM) and Convolutional Neural Network (CNN) to classify the software requirements into functional and non-functional, and explored the impact of word embedding techniques on the performance. They concluded that Doc2Vec was a worse choice than fastText for feature selection as it faced the issue of sparsity due to a small vocabulary. Word2Vec performed at par with fastText, giving a highest F1-score of 91.6%. TF-IDF with SVM gave the worst performance with an F1-score of 38.42% due to the same reasons for the poor performance as Doc2Vec.

Rahimi et al. [6] attempted to improve the accuracy of software requirement classification using ensemble learning techniques with SVM, SVC, and Logistic Regression as classifiers. TF-IDF and CountVectorizer were used for feature extraction. CountVectorizer outperformed TF-IDF in all aspects and was used for word embedding. SVM, Support Vector Classifier (SVC), and Logistic Regression produced accuracies of 99.0%, 97.0%, and 99.0%, respectively, and were used to monitor the accuracy of the ensemble approach. On the constructed dataset, the proposed strategy was compared to other existing ensemble approaches, and their technique had the highest accuracy, with 99.45%.

Improving on their previous work, Rahimi et al. [7] proposed three different ensemble techniques to enhance the accuracy: mean ensemble, accuracy per class as a weight ensemble, and accuracy as a weight ensemble, with a combination of four different Deep Learning models- LSTM, BiLSTM, GRU, and CNN. The DL models achieved accuracies of 93.22%, 94.49%, 92.37%, and 93.22%, respectively. Ensemble learning helped boost these results by achieving 95.7% accuracy per class as a weight and 94.9% accuracy as a weight. The accuracy per class weight ensemble approach was used to reach 0.96 for the other performance evaluation parameters.

Abad et al. [1] made a key contribution to software requirements classification by first classifying the requirements into functional and non-functional and then further classifying the non functional requirements into their sub-categories. They used three methods for the latter. Topic modeling with Latent Dirichlet Allocation (LDA) algorithm and Biterm Topic Model (BTM) method in which highly correlated words are grouped into topics was employed. BTM generated better models and topics but performed much worse than LDA as BTM does all of its modeling at the corpus level, and biterms are generated independently of topics. On evaluating the clusterings using Hopkins statistic on the non-functional requirements it was shown that the dataset under examination is unstructured, and sub-categories are not effectively segregated. Thus, unsupervised approaches are not ideal. On the other hand, a Binarized Naive-Bayes approach resulted in accuracy of 90%.

3 Study Design

This section contains information on the various design settings used in this study.

3.1 Experimental Dataset

The proposed software requirement solution is validated using the same datasets as Cleland-Huang and his team [2,6]. With the support of MS students from DePaul University, Cleland-Huang and his team retrieved the data and made it available for public research via the PROMISE repository. Figure 1 depicts the functional and quality characteristics of this dataset. The first point to note about Fig. 1 is that the PROMISE repository is unbalanced in terms of functional and non-functional requirements, with 382 out of 625 being quality requirements.

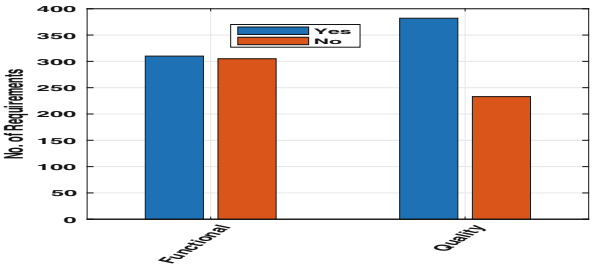

Fig. 1. Data-Sets

3.2 Data Normalization

Prior to feeding data into the model, data preprocessing is critical since it aids in data normalization [5]. To normalize the dataset, the following stages were followed in order:

- In the requirements specifications, all characters are transformed to lowercase letters.
- All alphanumeric and symbol characters are maintained, but others are removed.
- Stop-words are often used terms like 'a' and 'the,' and words with lengths less than two are deleted due to their insignificance.
- Finally, the corpus's phrases are tokenized into words.

3.3 Word Embedding

Vectorization techniques aid in efficient information retrieval and relevance ranking, which helps ML models perform better. In the classification phase, this

research examines the performance of five word embedding techniques: Skip-Gram (SKG), Word2Vec, TF-IDF, Global Vectors for Word Representation (GloVe)[1], and Bag of Words (CBOW)[2]. After being normalized, each requirement is vectorized using the aforementioned vector space models, resulting in a two-dimensional array with dimensions.

3.4 Feature Selection and Dimensionality Reduction Techniques

After performing word embedding on the data, we select essential feature vectors that impact the performance of the models. Rank Sum Test (RST) and Principal Component Analysis (PCA) are used to get rid of redundant and irrelevant features that might negatively impact the performance of the models. The performance of models with these features is compared with the original features to see the difference. This step helps reduce overfitting and training time [3]. **PCA** is a method for reducing the dimensionality of such datasets, improving interpretability, and minimizing data loss. To obtain lower dimensional data, we merely select the top principal components. While some information is lost, we are sacrificing accuracy for simplicity. The variables are sometimes so highly connected that they include redundant information. **Rank Sum test** - When the data consists of independent samples from two populations, the rank-sum test can be used to test the null hypothesis that their distributions are identical. When the value of the test statistic is either considerably large or significantly small, the rank-sum test is used to reject the null hypothesis.

3.5 SMOTE

Many machine learning algorithms have the flaw of ignoring the minority class in imbalanced datasets, despite the fact that it is often the performance on those that matters. It is critical to use the SMOTE approach on the imbalanced data in our dataset in order to apply subsequent Machine Learning techniques. SMOTE (Synthetic Minority Oversampling Technique) is a data augmentation technique in which current minority class instances are duplicated, or new minority class examples are generated. Although it does not provide any new data to the dataset, it overcomes the imbalance problem, allowing us to use machine learning techniques later [4].

3.6 Classifiers

To prevent the bias produced by distinct features measured at different scales, Min-Max Scaling is conducted prior to training the Machine Learning model. Using K-Fold Cross-validation with a k value of 10, the dataset is separated

[1] https://nlp.stanford.edu/projects/glove/.
[2] https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b.

into training and testing subsets and categorized using eight machine learning classifiers: Bagged k-Nearest Neighbors, Bagged Decision Tree, Bagged Naive-Bayes, Random Forest, Extra Tree, Adaptive Boost, Gradient Boosting, and Majority Voting ensemble classifier [3].

4 Research Methodology

In this work, we apply five different word embedding methods to extract features from the software requirements in natural language, balanced the classes using SMOTE, selected features from both the balanced and imbalanced datasets, and use them as input to develop models for classification. These models are trained with eight different classifiers, with one of them being a soft voting ensemble classifier. The proposed methodology entails extracting features from text data using word embedding techniques, class balancing, deleting irrelevant and redundant features, and lastly, creating prediction models utilizing eight distinct classifiers. In this experiment, we employ the Rank-Sum test to identify features that may distinguish between functional and non-functional requirements, as well as PCA to reduce the variance of the numerical features produced. Finally, eight distinct classifiers, including an ensemble classifier, are used to create the required classifications, which are then confirmed using 10-fold cross-validation. The accuracy, F-measure, and AUC of the developed requirement categorization models are extracted and compared using the Box-plot and Friedman test.

5 Empirical Results and Analysis

In this work, we apply five different word embedding techniques, a class balancing technique, two feature selection techniques, and eight different classification techniques to categorize the software requirements into functional or non-functional. Therefore, a total of 480 [5 word-embedding techniques * 2 requirements datasets (1 functional requirements dataset + 1 quality requirements dataset) * (1 balanced dataset + 1 imbalanced dataset)* 3 sets of features * 8 classifiers] distinct prediction models are built in this study. The predictive ability of these trained models is evaluated with Area Under Curve (AUC), F-measure, and accuracy performance values, as shown in Table 1. The first noteworthy finding is that all of the models trained for software requirement classification have an AUC value larger than 0.7, indicating that the trained models can classify functional and non-functional requirements. The second noteworthy finding is that software requirement classification models trained on SMOTE balanced data perform better than their imbalanced counterparts in most cases.

Table 1. AUC: SMOTE and Original Data

	BAG-KNN	BAG-DT	BAG-NB	RF	EXTR	ADaB	GRaB	MVE	BAG-KNN	BAG-DT	BAG-NB	RF	EXTR	ADaB	GRaB	MVE
	OD								SMOTE							
TFIDF																
FUN																
AF	0.75	0.84	0.86	0.86	0.90	0.74	0.71	0.89	0.76	0.85	0.85	0.85	0.89	0.68	0.70	0.88
PCA	0.87	0.85	0.88	0.85	0.86	0.70	0.70	0.88	0.87	0.84	0.91	0.84	0.85	0.73	0.71	0.89
RST	0.78	0.77	0.73	0.76	0.71	0.76	0.79	0.80	0.79	0.77	0.72	0.75	0.70	0.79	0.82	0.81
QUA																
AF	0.85	0.87	0.91	0.90	0.91	0.73	0.73	0.92	0.91	0.91	0.92	0.95	0.95	0.80	0.78	0.95
PCA	0.92	0.91	0.93	0.91	0.91	0.76	0.70	0.94	0.93	0.93	0.93	0.94	0.94	0.77	0.77	0.96
RST	0.84	0.84	0.75	0.78	0.77	0.84	0.82	0.85	0.84	0.88	0.80	0.88	0.88	0.87	0.85	0.88
CBOW																
FUN																
AF	0.80	0.72	0.72	0.71	0.71	0.75	0.76	0.78	0.83	0.72	0.70	0.72	0.73	0.78	0.79	0.79
RST	0.86	0.76	0.80	0.78	0.76	0.79	0.79	0.83	0.85	0.75	0.80	0.78	0.76	0.79	0.78	0.82
PCA	0.69	0.70	0.66	0.68	0.66	0.73	0.71	0.72	0.72	0.72	0.67	0.72	0.71	0.70	0.70	0.76
CBOW																
QUA																
AF	0.87	0.79	0.70	0.76	0.78	0.77	0.79	0.81	0.91	0.85	0.71	0.83	0.86	0.80	0.80	0.86
RST	0.89	0.79	0.81	0.83	0.81	0.80	0.80	0.87	0.93	0.87	0.81	0.89	0.89	0.81	0.83	0.89
PCA	0.79	0.77	0.73	0.77	0.72	0.78	0.80	0.80	0.83	0.82	0.71	0.83	0.87	0.79	0.79	0.86
SKG																
FUN																
AF	0.81	0.73	0.73	0.78	0.79	0.73	0.75	0.79	0.82	0.77	0.73	0.77	0.76	0.75	0.73	0.79
RST	0.83	0.76	0.75	0.78	0.80	0.75	0.74	0.78	0.83	0.76	0.77	0.78	0.77	0.76	0.77	0.79
PCA	0.81	0.75	0.72	0.73	0.78	0.79	0.80	0.80	0.85	0.78	0.72	0.76	0.77	0.76	0.78	0.81
SKG																
QUA																
AF	0.87	0.84	0.80	0.81	0.83	0.82	0.81	0.85	0.91	0.89	0.82	0.90	0.91	0.87	0.86	0.89
RST	0.89	0.85	0.82	0.81	0.85	0.82	0.81	0.86	0.92	0.90	0.84	0.90	0.92	0.84	0.86	0.90
PCA	0.86	0.79	0.75	0.80	0.80	0.80	0.77	0.86	0.92	0.89	0.78	0.90	0.90	0.83	0.83	0.91
GLOVE																
FUN																
AF	0.86	0.79	0.81	0.77	0.77	0.73	0.73	0.85	0.86	0.75	0.80	0.78	0.80	0.76	0.75	0.84
RST	0.87	0.77	0.83	0.80	0.80	0.75	0.77	0.87	0.86	0.78	0.82	0.79	0.80	0.74	0.75	0.85
PCA	0.82	0.70	0.75	0.71	0.74	0.71	0.71	0.81	0.81	0.73	0.71	0.71	0.66	0.69	0.71	0.81
GLOVE																
QUA																
AF	0.90	0.79	0.84	0.81	0.84	0.82	0.80	0.88	0.93	0.89	0.86	0.91	0.94	0.82	0.83	0.93
RST	0.90	0.81	0.86	0.82	0.85	0.79	0.77	0.88	0.94	0.89	0.88	0.91	0.91	0.83	0.82	0.92
PCA	0.85	0.75	0.76	0.73	0.75	0.75	0.71	0.85	0.91	0.85	0.79	0.82	0.85	0.80	0.77	0.89
W2V																
FUN																
AF	0.88	0.80	0.82	0.79	0.79	0.77	0.76	0.86	0.88	0.82	0.83	0.77	0.81	0.79	0.80	0.88
RST	0.88	0.82	0.84	0.81	0.79	0.80	0.79	0.87	0.89	0.81	0.85	0.83	0.81	0.81	0.80	0.86
PCA	0.82	0.73	0.75	0.68	0.70	0.74	0.76	0.83	0.85	0.74	0.78	0.73	0.72	0.76	0.76	0.83
W2V																
QUA																
AF	0.90	0.85	0.86	0.83	0.83	0.81	0.82	0.89	0.95	0.89	0.89	0.90	0.91	0.85	0.85	0.93
RST	0.92	0.85	0.88	0.85	0.86	0.82	0.80	0.90	0.95	0.89	0.90	0.92	0.92	0.85	0.84	0.93
PCA	0.87	0.79	0.82	0.75	0.77	0.83	0.82	0.89	0.91	0.83	0.82	0.86	0.87	0.85	0.84	0.90

5.1 Comparative Analysis

In this section, the performance of different software requirement classification models trained using different ensemble learning on original data and balanced data with reduced sets of vectors of embedding techniques are compared using

Area under ROC curve (AUC), simulated F-measure, and accuracy. We have used box plots acting as a visual representation of the comparative performance and Friedman test has been employed in this paper to corroborate the results. The below is tested using Friedman test:

- ***Null Hypothesis***- The predictive ability of software requirements classification models do not depends on different ML techniques, balancing techniques, embedding techniques, and feature selection techniques.
- ***Alternate Hypothesis***- The predictive ability of software requirements classification models depends on different ML techniques, balancing techniques, embedding techniques, and feature selection techniques.

The above null hypothesis is accepted if calculated p-value of Friedman test is less than 0.05 with degrees of freedom as 4 for word embedding, 1 for class balancing, 2 for feature selection, and 7 for different classifier comparisons.

5.2 Word-Embedding

Initially, we have used five different embedding techniques to extract numerical value of software requirement text. The quality of finding best numerical vectors using these techniques are compared using AUC Accuracy, Recall, Precision, and F-measure.

Box-Plot: Word-Embedding. The performance value of various word embedding approaches is visually depicted in Fig. 2. The information present in Fig. 2 suggested that the models trained by taking numeric vectors computed by W2V and TF-IDF more better ability to classify requirements as compared to other techniques. Similarly, the models built with CBoW have a low predictive performance. W2V-created models have an average accuracy of 75.64%, a maximum accuracy of 86.26%, and a Q3 accuracy of 78.97%, which implies that 25% of W2V-created models have an accuracy of higher than 78.97%. These conclusions are supported by the AUC data and F-measure of these models, which reveal that classification models based on W2V and TF-IDF outperform classification models based on other word-embedding techniques.

Friedman Test: Word-Embedding. The Friedman Test is also used to compare the predictive ability of the models constructed using various word embedding techniques in this study. The test's purpose is to determine if the null hypothesis is true or false. "The various word embedding strategies have no meaningful impact on the performance of the classification models," asserts the null hypothesis. The mean ranks for the various word embedding approaches are

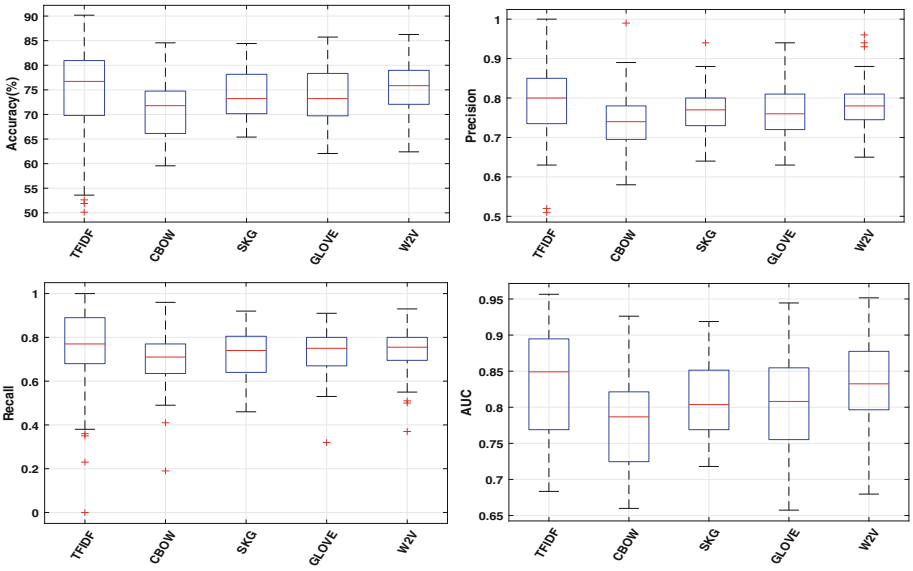

Fig. 2. Box-plot: word embedding approach

listed in Table 2. The better is the models' performance, the lower is the mean rank. The lowest mean rank is 2.07 for W2V, while the highest mean rank is 4.28 for CBoW. W2V slightly outperforms TF-IDF, which has a mean rank of 2.30. It's possible that TF-IDF's strong performance is related to the fact that requirements documents have many similar phrases and terms. TF-IDF can successfully prevent frequent terms used in articles from having an impact on the categorization better than other word-embedding algorithms because of its way of allocating weights to each term.

Table 2. Friedman test: word-embedding

	TFIDF	CBOW	SKG	GLOVE	W2V	Rank
TFIDF	1.00	0.00	0.03	0.02	0.71	2.30
CBOW	0.00	1.00	0.00	0.00	0.00	4.28
SKG	0.03	0.00	1.00	0.85	0.01	3.02
GLOVE	0.02	0.00	0.85	1.00	0.01	3.32
W2V	0.71	0.00	0.01	0.01	1.00	2.07

5.3 Feature Selection

We employ two different forms of feature selection strategies in the proposed research: Rank Sum test and PCA, and we use all of the original features for

training predictive models for requirements classification in a third set of models. Both the functional and non-functional needs datasets are subjected to these feature selection strategies.

Box-Plot: Feature Selection. RST appears to produce a better set of features than any other feature selection technique as shown in Fig. 3. The average accuracy of RST is 76.52%, with a minimum of 63.36% and a maximum of 89.14%. With an average accuracy of 70.56%, the features generated using PCA had the weakest performance among the three sets of features.

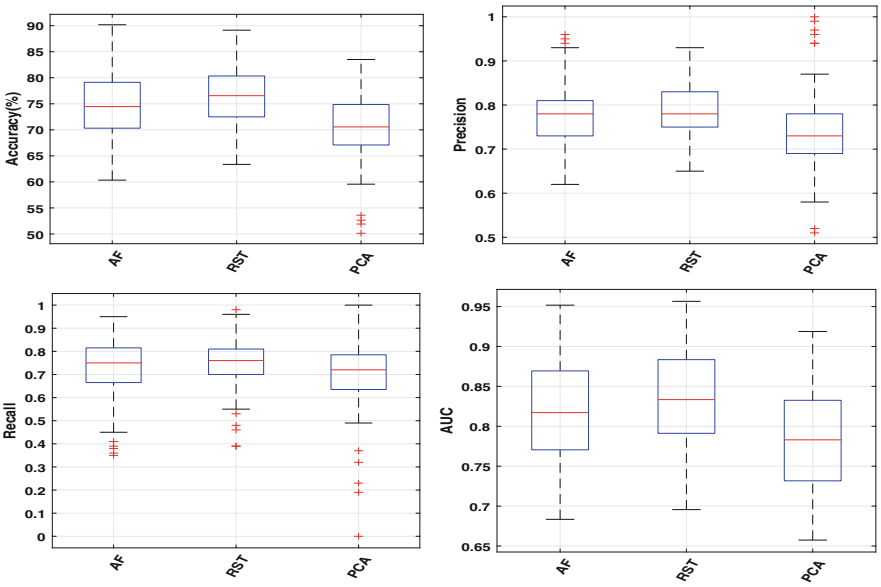

Fig. 3. Box-plot: feature selection

Friedman Test: Feature Selection. We also use the Friedman test to assess the different feature selection strategies based on their predictive ability on the performance metrics of the models, which are generated using three different sets of features. "The requirements classification models created using different sets of features do not have a substantial difference in their predictive ability" is the null hypothesis, which must be accepted or rejected based on the Friedman test. The Friedman test was performed with degrees of freedom of 2 and a significance threshold of = 0.05. The mean ranks of the three sets of features are shown in Table 3. The Friedman test's mean ranks can be used to compare the

performance of various techniques under investigation. In comparison to others, lower mean ranks indicate higher performance. The models trained with the set of RST features had the lowest mean rank (1.40), followed by the original set of features (1.97), and lastly, PCA (2.63). The mean ranks show that when RST features are included, the models perform better, while using dimensionality reduction techniques like PCA only regresses the models' performance.

Table 3. Friedman test: feature selection

	AF	RST	PCA	Rank
AF	1.00	0.03	0.00	1.97
RST	0.03	1.00	0.00	1.40
PCA	0.00	0.00	1.00	2.63

5.4 Classification Techniques

To categorize the software requirements, the study uses eight distinct classifiers. These classifiers are used with a variety of word-embedding approaches and on datasets with both functional and quality requirements. We employ Bagged k-Nearest Neighbors (Bag-kNN), Bagged Decision Tree (Bag-DT), Bagged Naive-Bayes (Bag-NB), Random Forest (RF), Extra Tree (EXTR), Adaptive Boost (AdaB), Gradient Boosting (GraB), and Majority Voting ensemble (MVE) classifiers in this study.

Box-Plot: Classification Techniques. Figure 4 helps show accuracy, precision, recall, and AUC statistics for the various classifiers employed, such as Mean, Median, Min, Max, Q1, and Q3. The model that was trained using the MVE had the best overall performance, with a mean accuracy of 78.29%. In comparison to other classifiers, the GraB and AdaB classifiers performed the worst, with mean accuracy of 71.76% and 71.67%, respectively. The models created with the ensemble classifier have a maximum accuracy of 89.14%, a minimum accuracy of 65.44%, a Q1 accuracy of 73.73%, and a Q3 accuracy of 82.49%. The Bag-kNN classifier performs marginally worse than the ensemble classifier but significantly better than the others.

Friedman Test: Classification Techniques. The Friedman test is also applied to the performance metrics of the various classifiers in order to compare the performance of the models statistically. The test's purpose is to determine if the null hypothesis is true or false. "The requirements classification models constructed using the different classifiers do not have a substantial difference in their prediction ability" is the null hypothesis for this test. The Friedman test

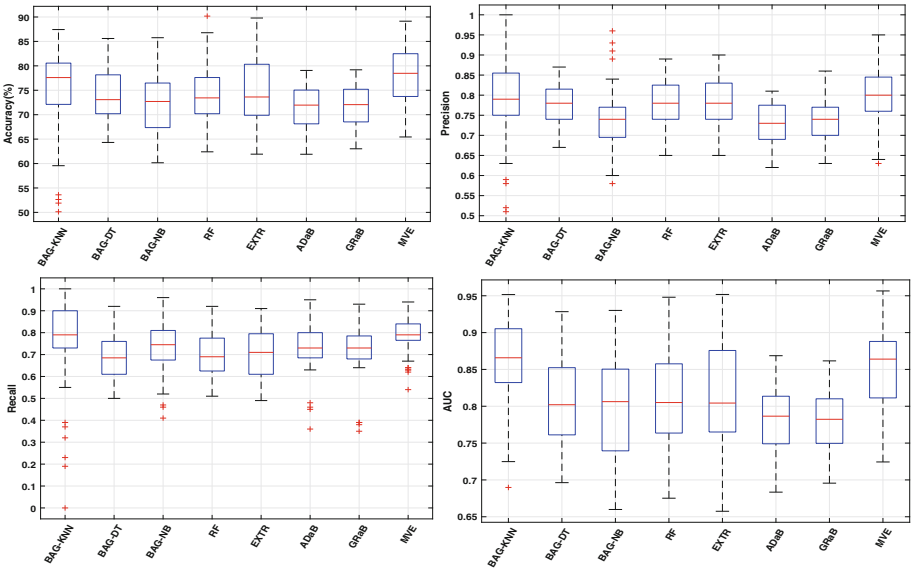

Fig. 4. Box-plot: classification techniques

was performed with degrees of freedom as 7 and a significance threshold of = 0.05. Table 4 shows the mean rank of various classifiers after applying the Friedman test. From Table 4, we can observe that the mean rank of 1.92 and 1.98 of Bag-kNN and ensemble classifier (MVE) is the lowest among all. The GraB classifier with a mean rank of 6.13 is the highest and thus symbolizes that the GraB classifier is the worst performer out of the eight.

Table 4. Friedman test: classification techniques

	BAG-KNN	BAG-DT	BAG-NB	RF	EXTR	ADaB	GRaB	MVE	Rank
BAG-KNN	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.55	1.92
BAG-DT	0.00	1.00	0.50	0.99	0.55	0.01	0.01	0.00	5.22
BAG-NB	0.00	0.50	1.00	0.53	0.34	0.10	0.04	0.00	5.25
RF	0.00	0.99	0.53	1.00	0.62	0.02	0.01	0.00	5.02
EXTR	0.00	0.55	0.34	0.62	1.00	0.01	0.00	0.00	4.45
ADaB	0.00	0.01	0.10	0.02	0.01	1.00	0.75	0.00	6.03
GRaB	0.00	0.01	0.04	0.01	0.00	0.75	1.00	0.00	6.13
MVE	0.55	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.98

5.5 SMOTE

In this paper, we use two different types of models based on the data used for training the models. One set is class-balanced using SMOTE, and the other set contains the imbalance classes.

Box-Plot: SMOTE. Figure 5 shows the visual depiction of the predictive ability of the models trained on the balanced dataset compared to that of the models trained on the imbalanced dataset. The models trained using SMOTE outperformed the models trained on the original dataset in most box plot metrics. The models trained with SMOTE had a mean accuracy of 74.44%, maximum of 90.18%, minimum of 50.13%, and 79.13% Q3.

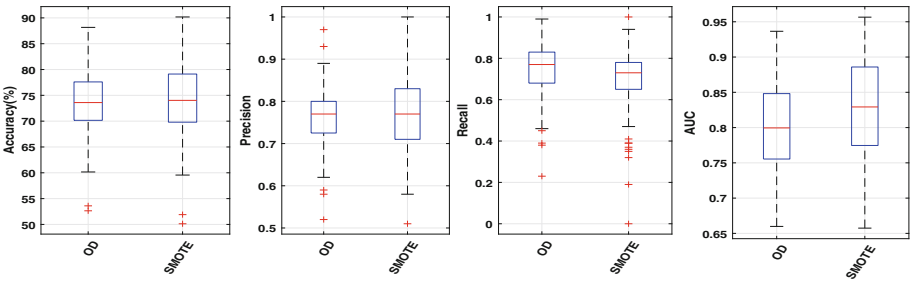

Fig. 5. Box-plot: SMOTE

Friedman Test: SMOTE. Finally, we have also applied Friedman test on the models trained using original data and sampled data. Here, this test was carried out on 0.05 significance level with 1 degrees of freedom. The information computed using this test suggested that the use of sampled data for training models significantly help in improving the performance of the models.

6 Conclusion

Automated classification of functional and non-functional requirements is a crucial task that helps overcome the inconsistencies between the software engineers and user/client demands in significantly less time than manual classification. In this work, various word embedding techniques, class balancing techniques, feature selection techniques, multiple classifiers have been used. The results are expected to help future researchers pick the ideal combination for requirements classification. The key conclusions that we arrived at were as follows:

– The models with features extracted using w2v and TF-IDF performed markedly better than other word embedding techniques.

- The models trained using the Majority Voting Ensemble classifier and bagged k-Nearest Neighbors classifier gave the highest performance out of the eight classifiers used.
- The features selected using the Rank Sum test were superior to any other subset of features used in this study.
- Class balancing using SMOTE helped improve the performance of requirements classification models.

Based on the findings of this experiment, ensemble classifiers are key to more accurate software requirements classification. More work is needed to analyze the best ML techniques to classify the functional and non-functional or quality requirements into their sub-categories, which would provide increased benefits to software developers.

Acknowledgements. This research is funded by TestAIng Solutions Pvt. Ltd.

References

1. Abad, Z.S.H., Karras, O., Ghazi, P., Glinz, M., Ruhe, G., Schneider, K.: What works better? A study of classifying requirements. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pp. 496–501. IEEE (2017)
2. Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: Automated classification of non-functional requirements. Requir. Eng. **12**(2), 103–120 (2007)
3. Kumar, L., Sripada, S.K., Sureka, A., Rath, S.K.: Effective fault prediction model developed using least square support vector machine (LSSVM). J. Syst. Softw. **137**, 686–712 (2018)
4. Kumar, L., Sureka, A.: Application of LSSVM and smote on seven open source projects for predicting refactoring at class level. In: 2017 24th Asia-Pacific Software Engineering Conference (APSEC), pp. 90–99. IEEE (2017)
5. Kumar, R., Singh, J., Kaur, A.: An empirical study of bad smell in code on maintenance effort. Int. J. Comput. Sci. Eng. **5**(6), 294–304 (2016)
6. Rahimi, N., Eassa, F., Elrefaei, L.: An ensemble machine learning technique for functional requirement classification. Symmetry **12**(10), 1601 (2020)
7. Rahimi, N., Eassa, F., Elrefaei, L.: One-and two-phase software requirement classification using ensemble deep learning. Entropy **23**(10), 1264 (2021)
8. Tiun, S., Mokhtar, U., Bakar, S., Saad, S.: Classification of functional and non-functional requirement in software requirement using word2vec and fast text. In: Iournal of Physics: Conference Series, vol. 1529, p. 042077. IOP Publishing (2020)