Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 895–904
DOI: 10.15439/2022F140
ISSN 2300-5963 ACSIS, Vol. 30

Software Requirements Classification using Deep-learning Approach with Various Hidden Layers

Sanidhya Vijayvargiya[1]
Department of Computer Science & Information Systems
BITS Pilani Hyderabad Campus
f20202056@hyderabad.bits-pilani.ac.in

Lov Kumar[2]
Department of Computer Science & Information Systems
BITS Pilani Hyderabad Campus
lovkumar@hyderabad.bits-pilani.ac.in

Lalita Bhanu Murthy[3]
Department of Computer Science & Information Systems
BITS Pilani Hyderabad Campus
bhanu@hyderabad.bits-pilani.ac.in

Sanjay Misra[4]
Østfold University College, Halden, Norway
ssopam@gmail.com

***Abstract*—Software requirement classification is becoming increasingly crucial for the industry to keep up with the demand of growing project sizes. Based on client feedback or demand, software requirement classification is critical in segregating user needs into functional and quality requirements. However, because there are numerous machine learning (ML) and deep-learning (DL) models that require parameter tuning, the use of ML to facilitate decision-making across the software engineering pipeline is not well understood. Five distinct word embedding techniques were applied to the functional and quality software requirements in this study. The imbalanced classes in the dataset are balanced using Synthetic Minority Oversampling technique (SMOTE). Then, to reduce duplicate and unnecessary features, feature selection and dimensionality reduction techniques are used. Dimensionality reduction is accomplished with Principal Component Analysis (PCA), while feature selection is accomplished with the Rank-Sum Test (RST). For binary categorization into functional and non-functional needs, the generated vectors are provided as inputs to eight distinct Deep Learning classifiers. The findings of the research show that using a combination of word embedding and feature selection techniques in conjunction with various classifiers can accurately classify functional and quality software requirements.**

***Keywords*—Functional Requirements, Non-Functional Requirements, Deep Learning, Data Imbalance Methods, Feature Selection, Classification Techniques, Word Embedding.**

I. Introduction

SOFTWARE requirements classification deals with segregating the clients' requirements and demands found in the Software Requirements Specification (SRS) document into functional and non-functional requirements. It is a key step in the software development pipeline which can be automated using Machine Learning techniques. This allows the industry to save on labor expenses, as a domain expert is often required, while also optimizing the process and saving crucial time [1]. A key problem that needs to be addressed during requirements classification is that of the inconsistency in terminology used by the clients and the software engineers. This may lead to misclassification of the software requirements.

Functional requirements are the demands that the end-user defines as critical characteristics that the system should supply and that can be observed immediately in the finished result. This is how the input to the system, the action to be taken, and the intended output are all defined or stated. The system's basic quality standards, often known as non-functional requirements, include factors like reliability, maintainability, security, and portability [2].

Another problem faced during software requirements classification is the imbalance between the number of instances of functional and non-functional requirements classes. Data imbalance means that the number of instances of minority class are much lower than those of the majority class. Because of the unbalanced distribution of data, classifiers are misled while learning the minority class, resulting in biased and erroneous findings [3]. A good software requirements categorization model will be one that has been trained on a similar distribution of functional and non-functional requirement classes. In this study, this problem is addressed using oversampling through Synthetic Minority Oversampling Technique(SMOTE).

In this paper, we look to solve the above problem, and create highly accurate software requirements classification models which can be reliably used in the industry. The following are the research questions (RQs) that will be used to attain the aforementioned goals.

- RQ1: Which feature extraction technique can best capture the unstructured, textual data present in the SRS document, and convert it to structured data in the form of numerical vectors?
- RQ2: Which feature selection techniques are the best at getting rid of redundant and irrelevant features which may affect the performance of the classification models?
- RQ3: For what structure of the deep learning classifiers do the software requirements classification models achieve the best results?
- RQ4: How does the application of class balancing technique through oversampling affect the performance of the models?

The criteria used to evaluate the performance of the various models are F-measure, accuracy, and Area under the ROC curve (AUC). The Friedman test was used to determine whether an ML technique resulted in a substantial difference in performance. The PROMISE dataset[4] was used in this study, which contains 625 labeled criteria from 15 different projects. The contributions of the study are as follows:

- An extensive comparison of various word embedding techniques for the purpose of feature extraction is provided to analyze which technique is best suited for the SRS document.
- A thorough investigation on the effect of various feature selection techniques on the performance of classification models in software engineering is presented.
- Deep learning classifiers are employed to increase the accuracy of software requirements classification from previous studies, with variations in number of layers, and type of layer being analyzed to find the best deep learning model out of the eight distinct DL classifiers used in this study.
- The study evaluates and analyses the performance of requirements classification models using relevant performance metrics. The study includes a thorough statistical analysis to back up the findings with statistical testing, unlike previous studies.
- The effect of class-balancing techniques on software datasets to build more accurate models is examined.

The remainder of the paper is structured as detailed here: Section II presents a literature review on software requirement classification and various word embedding approaches that are used in this study. Section III describes the experimental dataset collection as well as the various machine learning algorithms used. The research methodology is described in Section IV using an architecture framework. In Section V, the results of the experiments, along with their analysis, are presented. Section VI shows a comparison of models created using various word-embedding approaches, sets of features, and machine learning models. Finally, Section VII summarizes the information provided and offers directions for further research.

II. Related Work

A. Software Requirements Word Embeddings

Navarro-Almanza et al. [5] explore Word2Vec using Skip-Gram to get structured representation for the textual software requirements dataset. The purpose of the Skip-gram model is to anticipate context words. The projection from Skip-Gram is a continuous vector space rendition of the word with a low dimensionality. The models developed achieved a maximum of 0.8 precision, 0.785 recall, and 0.77 F-measure.

To improve how well the word embeddings capture the content of the text, Marcacini et al. [6] analyze the impact of using contextual word embeddings for software requirements. They use the RE-BERT model to obtain the structured data to feed into their hierarchical clustering classifier. BERT is built on the Transformers architecture and uses a deep neural network. To address the sequence of occurrence of the tokens, a positional embedding is used. Static word embeddings, such as word2vec and FastText, on the other hand, have the issue of having the same embedding regardless of context, which makes structured modeling of software requirements difficult.

B. Classifying Software Requirements

Ott [7] employ two classifiers, Multinomial Naive Bayes, and Support Vector machine for software requirements classification. The classification techniques are applied on two datasets, out of which one is confidential and the other is public. The maximum recall achieved by the Naive Bayes classifier is 0.94, whereas the Support Vector machine achieves a precision of 0.86 in the best model. Baker et al. [8] work on classification of non-functional requirements into their subcategories. The authors compare the performance of the CNN model with that of an ANN model and results indicate that the CNN model outperforms the ANN on most performance metrics. The ANN consists of one hidden layer of 20 neurons. The ANN model has a precision of 82% to 90%, a recall of 78% to 85%, and the greatest F-score of 84%.With the maximum F-score of 92%, the CNN model obtains precision between 82% and 94%, and recall between 76% and 97%. Rahimi et al. [9] focus on further classifying the functional requirements into six different categories: policy, action constraint, solution, definition, attribute constraint, and enablement. The authors use the ensemble approach which combined five distinct classifiers: support vector classification, support vector machine, decision tree, logistic regression, and Naïve Bayes. For each class, accuracy per class as a weight is used to find the most optimal classifier. The best results are obtained using LR, SVC, SVM as the classifiers, which perform better than using all classifiers. An accuracy of 99.45% is achieved in classifying 600 functional requirements.

III. Study Design

This section presents the details regarding various design setting used for this research.

A. Experimental Dataset

Cleland-Huang and his team [4] used the same datasets to validate the proposed software requirement solution. Cleland-Huang and his team extracted this data with the help of MS students from DePaul University and rendered it for public research via the PROMISE repository. The functional and non-functional attributes are shown below in Figure 2. The first observation to be made about Figure 2 is that the PROMISE repository is uneven in number of functional and non-functional requirements, with quality requirements accounting for 382 of the 625 total.

B. Training of Models from Imbalanced Data Set:

Several ML algorithms have the issue of neglecting the minority class in unbalanced datasets, despite the reality that performance on those is often what matters. In order to use future ML techniques, it was essential to implement the SMOTE technique on the imbalanced classes in our dataset. SMOTE [10] (Synthetic Minority Oversampling Approach) is

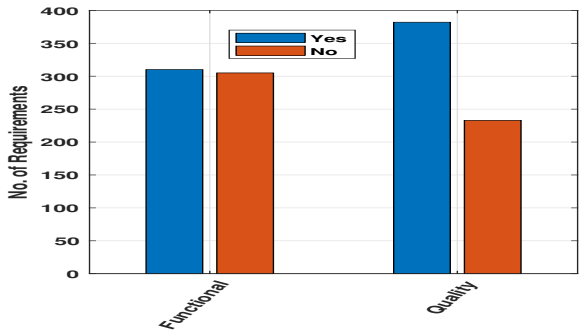

Fig. 2: Data-Sets

a data augmentation technique that duplicates existing minority class instances or generates new minority class instances. It solves the imbalance problem without adding any new data to the dataset, allowing us to employ machine learning techniques afterwards.

C. Word Embedding:

The dataset's textual data must be expressed as vectors in respect to one another. The dataset was subjected to five different word embedding techniques: Term Frequency and Inverse Document Frequency (TF-IDF), Continuous Bag of Words (CBOW)[1], Skip-Gram (SKG), Global Vectors for Word Representation (GLOVE)[2], and Google news Word to Vector (GW2V). The aim of these techniques, such as GLOVE, CBOW, etc., is to capture the semantic information in the text, which is not possible with other word-embedding techniques such as TF-IDF. The textual data was represented as a vector in an n-dimensional space using these techniques. Before applying the word embedding techniques, all characters in the requirements are converted to lowercase letters. We deleted any stopwords, bad symbols, and spaces. These will now be utilized to create models that will classify the requirements into functional and non-functional [11].

D. Feature Selection Techniques

We identify critical feature vectors that impact the performance of the models after doing word embedding on the data. To eliminate redundant and unnecessary features that may have a detrimental impact on the models' performance, the Rank Sum Test (RST) and Principal Component Analysis (PCA) are used. To see the difference, the performance of models with these features is compared to the performance of models without these features. This phase aids in the reduction of over-training and training time [12].

E. Classification Technique:

The dataset is divided into training and testing subsets and categorized using eight deep learning models using K-Fold Cross-validation with a k value of 5. The structure of the various models is presented below. All models have an input layer with number neurons equal to the number of features of input data. For each subsequent hidden layer, the number of neurons is halved. All layers involved in the Deep Learning models are either Dense layers or Dropout layers. In a Dense layer, each neuron in the layer receives input from all neurons of the previous layer. On the other hand, a Dropout layer randomly selects and omits a certain number of neurons in the layer while training the Deep Learning model. In this work, a dropout value of 0.2 is used. Dropout layers are used to solve the problem of overfitting models. Finally, the output layer consists of only one neuron which contains a binary value corresponding to the binary classification to either functional or non-functional requirements. The activation function for each layer is the rectified linear activation function or ReLU, except the output layer which uses a sigmoid activation function. Binary cross entropy is the loss function that is utilized to train the models with Adam as the optimizer. Figure 3 shows the architecture of the considered deep learning model1, model2, model3, and model4 (DL1, DL2, DL3, and DL4). Similarly, we

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

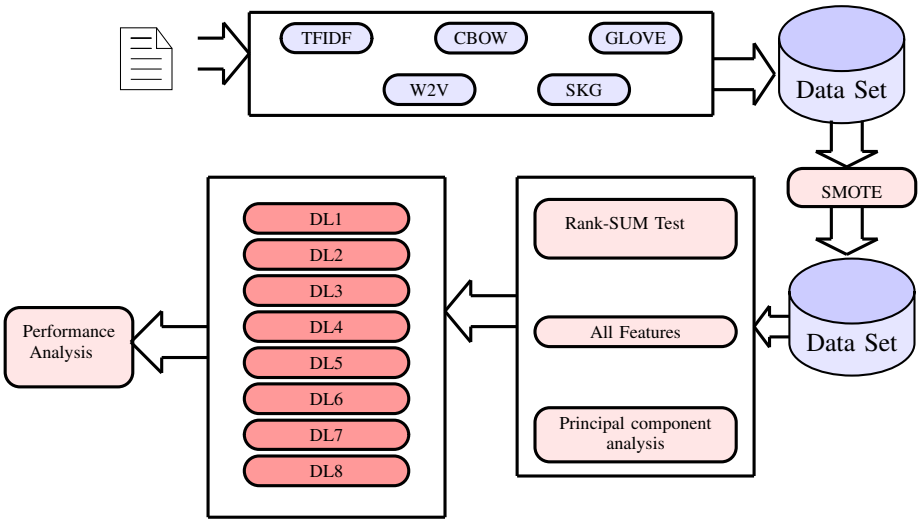

Fig. 1: Framework of proposed work

are increasing the number of hidden layers for four more deep learning models. The above considered models are validated using 5-fold cross-validation with batch size=30, epochs=100 , and Dropout=0.2.

IV. Research Methodology

Figure 1 provides a full outline of our proposed effort. We started by extracting the software requirements dataset from the PROMISE repository. This data contains labels of the software requirement category it falls under, i.e. functional or non-functional requirements. The data was then subjected to pre-processing techniques like converting all characters to lowercase, removing non-alphanumeric and non-symbol characters, eliminating frequently used words like 'the', and 'a', and other words with lengths less than or equal to two as they do not make a significant impact in the classification. Then, all of the phrases were tokenized to words.

To extract features from this unstructured, pre-processed data, five different word embedding techniques were applied to best capture the information. To account for the imbalance in classes due to 382 instances of quality requirements out of 625, a data oversampling technique in the form of SMOTE was used. Models were trained on both the balanced and imbalanced datasets to compare the effectiveness of the class-balancing. The features acquired after the word embeddings and class balancing needed to be refined to remove redundant and irrelevant features. Feature selection technique in the form of Rank-Sum test, and dimensionality reduction technique in the form of Principal Component Analysis were employed. These two sets of features as well as the set of original features were fed to the Deep Learning classifiers. This was done to help understand the importance of feature selection.

The eight different DL classifiers were trained using 5-fold cross validation. The classifiers have varying layer sizes, and layer types, but certain attributes like the optimizer, loss function, etc. remain constant across the different classifiers. These classifiers are named DL1, DL2, DL3, and so on till DL8. Finally, the performance of the various models developed was measured using accuracy, F-measure, and AUC. This performance was statistically analyzed using box-plots for visual representation, with statistics like mean, maximum, minimum, Q1, and Q3 for each performance metric. Further, any conclusions derived were statistically supported using the Friedman test.

V. Empirical Results and Analysis

To categorize software needs into functional or non-functional, we used five distinct word embedding approaches, a class balance strategy, two feature selection strategies, and eight different classification techniques. As a result, a total of 480 [5 word-embedding techniques $\times$ 2 requirements datasets(1 functional requirements dataset + 1 non-functional requirements dataset) $\times$ (1 imbalanced dataset + 1 balanced dataset) $\times$ 3 sets of features $\times$ 8 DL classifiers] models were generated. As shown in Tables I and II, the predictive performance of these trained models is assessed using the F-measure, accuracy and Area Under Curve (AUC) performance metrics.

- The high value of AUC confirms that the developed models have the ability to accurately classify the software requirements into functional and non-functional as almost all the performance values seen on the right side of Table I are greater than 0.75 AUC.
- The models developed using the Deep Learning structure of DL3 have better performance as compared to other classifiers.
- The models trained using neural network with ADAM (NNADAM) training algorithm have better predictive ability as compared LBFG, and SGD training algorithms.
- Simply by observing the values in Table I, we can see the difference SMOTE provides in improving the performance.

VI. Comparative Analysis

The various models created with using word embedding techniques, class balancing approaches, feature selection strategies, and different classifiers are compared in this section. The comparison is based on statistics such as the area under the ROC curve (AUC), F-measure, and accuracy, with box plots serving as a visualization of the comparative performance. The Friedman test was used in this research to verify the findings. The Friedman test is used to accept or reject the following hypothesis.

- ***Null Hypothesis***- There is no substantial difference in the predictive performance of software classification models constructed using different machine learning approaches.
- ***Alternate Hypothesis***- The prediction power of software classification models constructed using various ML approaches varies significantly.

With degrees of freedom of 4 for word embedding, 1 for class balancing, 2 for feature selection, and 7 for distinct DL classifier comparisons, the Friedman test was performed with a significance threshold of α = 0.05.

A. RQ1: Which feature extraction technique can best capture the unstructured, textual data present in the SRS document, and convert it to structured data in the form of numerical vectors?

In this work, five distinct word embedding approaches were utilized to compute the numerical vector of the functional and quality requirements: TF-IDF, Skip-Gram (SKG), Global Vectors for Word Representation (GloVe), W2V and Bag of Words (CBOW). To assess the prediction abilities of models generated using various word embedding techniques, the AUC, accuracy, and F-measure statistics were used.

1) Box-plot: Word-Embedding: Figure 4 illustrates the result of several word embedding algorithms. Models developed with the word embedding generated by TF-IDF are more reliable than other models, as shown in Figure 4. CBoW models exhibit poor prediction performance when compared to other methodologies. The mean AUC value of TF-IDF models is 0.91, with a maximum AUC of 0.98 and a Q3 accuracy of 0.96, implying that 25% of TF-IDF models have an AUC value

Model: "sequential_13"

Layer (type)	Output Shape	Param #
dense_36 (Dense)	(None, 1605)	2577630
dense_37 (Dense)	(None, 1)	1606

Total params: 2,579,236
Trainable params: 2,579,236
Non-trainable params: 0

(3.1) DL1

Model: "sequential_14"

Layer (type)	Output Shape	Param #
dense_38 (Dense)	(None, 1605)	2577630
dropout_8 (Dropout)	(None, 1605)	0
dense_39 (Dense)	(None, 1)	1606

Total params: 2,579,236
Trainable params: 2,579,236
Non-trainable params: 0

(3.2) DL2

Model: "sequential_16"

Layer (type)	Output Shape	Param #
dense_43 (Dense)	(None, 1605)	2577630
dropout_9 (Dropout)	(None, 1605)	0
dense_44 (Dense)	(None, 802)	1288012
dense_45 (Dense)	(None, 1)	803

Total params: 3,866,445
Trainable params: 3,866,445
Non-trainable params: 0

(3.3) DL4

Model: "sequential_12"

Layer (type)	Output Shape	Param #
dense_31 (Dense)	(None, 1605)	2577630
dense_32 (Dense)	(None, 802)	1288012
dense_33 (Dense)	(None, 401)	322003
dense_34 (Dense)	(None, 401)	161202
dense_35 (Dense)	(None, 1)	402

Total params: 4,349,249
Trainable params: 4,349,249
Non-trainable params: 0

(3.4) DL8

Fig. 3: Deep Learning Architecture

of more than 0.96. The accuracy data and F-measure of these models confirm these findings, revealing that classification techniques based on TF-IDF outperform classification models based on alternative word-embedding techniques.

2) Friedman Test: Word-Embedding: In this work, the Friedman Test is also utilized to examine the predictive power of the models created using different word embedding methods. The goal of the test is to see if the null hypothesis is correct. The null hypothesis asserts that "the various word embedding techniques have no discernible impact on the performance of the classification models." Table IV shows the mean ranks for the various word embedding techniques. The lower the mean rank, the better the models' performance. TF-IDF has the lowest mean rank of 1.88, while CBoW has the highest mean rank of 4.88. With a mean rank of 1.96, W2V provides comparable performance to TF-IDF. It's probable that the high performance of the TF-IDF is due to the fact that requirements papers contain numerous comparable phrases and terms. Because of its method of giving weights to each term, TF-IDF can effectively minimize frequent terms used in requirements from creating an effect on classification better than other word-embedding algorithms.

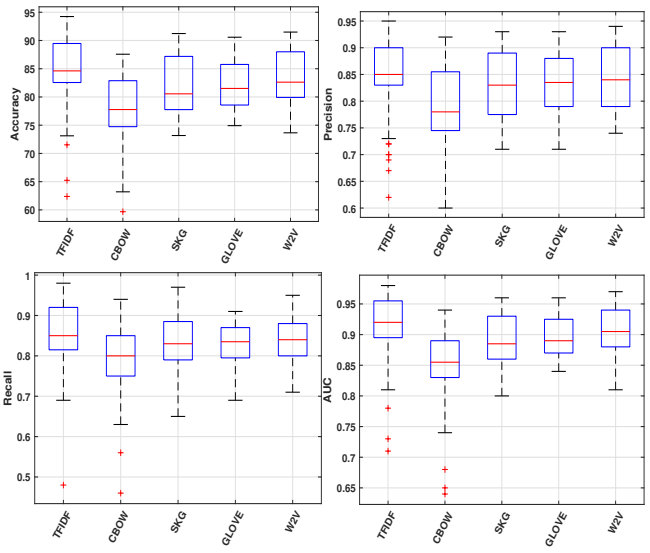

Fig. 4: Performance Box-Plot Diagram: Performance of Different Word Embedding

TABLE I: Precision and Recall

		Precision								Recall							
		OD(AF)															
		DL1	**DL2**	**DL3**	**DL4**	**DL5**	**DL6**	**DL7**	**DL8**	**DL1**	**DL2**	**DL3**	**DL4**	**DL5**	**DL6**	**DL7**	**DL8**
TFIDF	FUN	0.85	0.84	0.85	0.85	0.85	0.83	0.84	0.87	0.8	0.84	0.85	0.86	0.85	0.87	0.85	0.85
TFIDF	QUA	0.8	0.93	0.93	0.93	0.93	0.93	0.94	0.92	0.98	0.93	0.92	0.92	0.92	0.91	0.9	0.94
CBOW	FUN	0.74	0.76	0.78	0.76	0.75	0.72	0.76	0.8	0.73	0.81	0.76	0.8	0.76	0.8	0.79	0.73
CBOW	QUA	0.74	0.87	0.88	0.86	0.87	0.85	0.84	0.87	0.94	0.83	0.85	0.88	0.86	0.86	0.77	0.84
SKG	FUN	0.82	0.76	0.79	0.79	0.76	0.79	0.82	0.76	0.69	0.79	0.79	0.75	0.81	0.75	0.77	0.78
SKG	QUA	0.85	0.89	0.89	0.88	0.88	0.89	0.84	0.88	0.91	0.9	0.88	0.86	0.93	0.87	0.94	0.84
GLOVE	FUN	0.8	0.79	0.79	0.78	0.8	0.76	0.8	0.78	0.74	0.82	0.82	0.81	0.8	0.82	0.82	0.79
GLOVE	QUA	0.85	0.88	0.87	0.84	0.88	0.88	0.84	0.85	0.87	0.84	0.87	0.9	0.86	0.84	0.89	0.87
W2V	FUN	0.8	0.78	0.81	0.83	0.78	0.81	0.8	0.82	0.83	0.79	0.79	0.75	0.81	0.77	0.75	0.79
W2V	QUA	0.9	0.91	0.87	0.91	0.91	0.89	0.89	0.89	0.88	0.87	0.89	0.87	0.87	0.9	0.89	0.85
		OD(RST)															
TFIDF	FUN	0.88	0.87	0.84	0.85	0.86	0.84	0.86	0.85	0.75	0.82	0.82	0.81	0.8	0.81	0.8	0.8
TFIDF	QUA	0.81	0.91	0.9	0.9	0.9	0.9	0.9	0.88	0.98	0.92	0.91	0.93	0.93	0.92	0.92	0.93
CBOW	FUN	0.73	0.79	0.77	0.74	0.77	0.81	0.75	0.8	0.82	0.79	0.78	0.84	0.79	0.7	0.79	0.75
CBOW	QUA	0.77	0.87	0.82	0.86	0.84	0.84	0.87	0.82	0.92	0.86	0.92	0.87	0.88	0.88	0.85	0.92
SKG	FUN	0.78	0.76	0.75	0.79	0.78	0.75	0.82	0.77	0.77	0.83	0.83	0.82	0.78	0.8	0.77	0.79
SKG	QUA	0.84	0.89	0.86	0.88	0.89	0.88	0.9	0.89	0.9	0.9	0.91	0.91	0.91	0.89	0.9	0.88
GLOVE	FUN	0.78	0.79	0.81	0.8	0.83	0.79	0.79	0.79	0.79	0.83	0.8	0.8	0.76	0.81	0.79	0.82
GLOVE	QUA	0.84	0.85	0.9	0.9	0.88	0.87	0.88	0.87	0.87	0.89	0.85	0.87	0.89	0.9	0.88	0.88
W2V	FUN	0.81	0.81	0.75	0.85	0.83	0.81	0.79	0.79	0.83	0.82	0.84	0.75	0.77	0.8	0.82	0.82
W2V	QUA	0.87	0.88	0.9	0.9	0.9	0.88	0.88	0.9	0.89	0.93	0.9	0.91	0.88	0.88	0.91	0.9
		OD(PCA)															
TFIDF	FUN	0.69	0.77	0.75	0.78	0.7	0.76	0.76	0.7	0.79	0.75	0.77	0.7	0.82	0.78	0.72	0.79
TFIDF	QUA	0.74	0.88	0.83	0.9	0.84	0.85	0.85	0.86	0.94	0.83	0.9	0.85	0.88	0.85	0.88	0.85
CBOW	FUN	0.63	0.66	0.69	0.67	0.66	0.71	0.68	0.68	0.7	0.69	0.78	0.72	0.67	0.73	0.67	0.78
CBOW	QUA	0.64	0.76	0.82	0.78	0.85	0.82	0.76	0.81	0.91	0.82	0.85	0.84	0.67	0.74	0.85	0.83
SKG	FUN	0.76	0.79	0.75	0.75	0.81	0.74	0.8	0.74	0.68	0.79	0.78	0.85	0.79	0.82	0.75	0.82
SKG	QUA	0.71	0.89	0.9	0.9	0.85	0.89	0.89	0.89	0.97	0.87	0.88	0.86	0.9	0.85	0.85	0.86
GLOVE	FUN	0.8	0.77	0.77	0.8	0.75	0.8	0.77	0.8	0.69	0.78	0.77	0.78	0.8	0.79	0.79	0.75
GLOVE	QUA	0.75	0.85	0.88	0.87	0.86	0.87	0.84	0.87	0.91	0.87	0.86	0.85	0.85	0.88	0.91	0.86
W2V	FUN	0.78	0.78	0.76	0.77	0.77	0.76	0.81	0.77	0.71	0.8	0.83	0.81	0.79	0.78	0.74	0.77
W2V	QUA	0.79	0.88	0.88	0.88	0.88	0.87	0.87	0.88	0.95	0.86	0.86	0.86	0.87	0.86	0.87	0.83
		SMOTE(AF)															
TFIDF	FUN	0.84	0.84	0.84	0.84	0.85	0.83	0.84	0.81	0.8	0.84	0.85	0.85	0.85	0.84	0.84	0.84
TFIDF	QUA	0.88	0.94	0.94	0.95	0.95	0.95	0.95	0.94	0.92	0.92	0.92	0.93	0.93	0.93	0.92	0.93
CBOW	FUN	0.78	0.75	0.75	0.8	0.7	0.76	0.66	0.77	0.46	0.8	0.76	0.73	0.86	0.76	0.78	0.65
CBOW	QUA	0.86	0.89	0.89	0.77	0.86	0.89	0.85	0.89	0.69	0.85	0.85	0.86	0.85	0.84	0.8	0.81
SKG	FUN	0.75	0.8	0.77	0.71	0.78	0.73	0.77	0.73	0.84	0.71	0.74	0.82	0.8	0.81	0.83	0.81
SKG	QUA	0.87	0.9	0.92	0.85	0.93	0.89	0.91	0.87	0.87	0.85	0.89	0.9	0.82	0.87	0.91	0.93
GLOVE	FUN	0.81	0.81	0.78	0.79	0.77	0.81	0.81	0.79	0.78	0.82	0.85	0.77	0.8	0.81	0.73	0.82
GLOVE	QUA	0.88	0.92	0.92	0.9	0.92	0.92	0.93	0.87	0.84	0.85	0.86	0.88	0.86	0.86	0.81	0.89
W2V	FUN	0.79	0.83	0.83	0.82	0.75	0.81	0.79	0.79	0.73	0.82	0.78	0.81	0.89	0.81	0.8	0.81
W2V	QUA	0.9	0.92	0.92	0.88	0.91	0.91	0.91	0.8	0.87	0.88	0.87	0.89	0.91	0.93	0.87	0.92
		SMOTE(RST)															
TFIDF	FUN	0.85	0.85	0.85	0.82	0.83	0.83	0.85	0.85	0.73	0.83	0.82	0.84	0.83	0.83	0.84	0.82
TFIDF	QUA	0.85	0.92	0.91	0.92	0.92	0.9	0.93	0.92	0.92	0.9	0.91	0.9	0.91	0.91	0.91	0.93
CBOW	FUN	0.77	0.79	0.8	0.77	0.76	0.78	0.79	0.78	0.77	0.79	0.75	0.8	0.82	0.75	0.8	0.78
CBOW	QUA	0.79	0.86	0.84	0.86	0.88	0.92	0.89	0.86	0.88	0.87	0.87	0.9	0.86	0.83	0.83	0.86
SKG	FUN	0.75	0.79	0.8	0.83	0.78	0.79	0.77	0.75	0.81	0.81	0.83	0.75	0.8	0.81	0.82	0.84
SKG	QUA	0.86	0.91	0.92	0.93	0.88	0.87	0.9	0.93	0.86	0.85	0.83	0.79	0.9	0.89	0.85	0.83
GLOVE	FUN	0.78	0.81	0.83	0.79	0.84	0.79	0.77	0.71	0.79	0.83	0.79	0.78	0.78	0.81	0.82	0.85
GLOVE	QUA	0.86	0.9	0.91	0.91	0.9	0.91	0.87	0.91	0.85	0.88	0.86	0.85	0.87	0.9	0.89	0.86
W2V	FUN	0.79	0.8	0.79	0.79	0.85	0.82	0.83	0.78	0.83	0.84	0.83	0.81	0.78	0.84	0.77	0.8
W2V	QUA	0.88	0.92	0.93	0.93	0.93	0.92	0.9	0.88	0.88	0.85	0.87	0.88	0.87	0.88	0.88	0.91
		SMOTE(PCA)															
TFIDF	FUN	0.67	0.76	0.72	0.74	0.72	0.74	0.62	0.73	0.48	0.69	0.79	0.77	0.79	0.77	0.77	0.8
TFIDF	QUA	0.86	0.88	0.91	0.87	0.88	0.81	0.9	0.92	0.87	0.92	0.86	0.84	0.93	0.92	0.83	0.86
CBOW	FUN	0.6	0.7	0.71	0.74	0.73	0.7	0.68	0.77	0.56	0.69	0.85	0.78	0.71	0.79	0.69	0.75
CBOW	QUA	0.73	0.85	0.87	0.87	0.88	0.81	0.77	0.88	0.63	0.78	0.81	0.72	0.77	0.84	0.81	0.77
SKG	FUN	0.78	0.76	0.78	0.78	0.76	0.83	0.8	0.79	0.65	0.82	0.77	0.81	0.76	0.69	0.73	0.74
SKG	QUA	0.85	0.88	0.92	0.9	0.92	0.9	0.9	0.88	0.88	0.9	0.87	0.87	0.9	0.9	0.92	0.9
GLOVE	FUN	0.74	0.77	0.77	0.76	0.74	0.78	0.78	0.76	0.77	0.78	0.78	0.8	0.83	0.77	0.77	0.8
GLOVE	QUA	0.87	0.9	0.89	0.89	0.89	0.9	0.91	0.91	0.82	0.85	0.88	0.89	0.89	0.87	0.88	0.85
W2V	FUN	0.74	0.79	0.78	0.78	0.8	0.8	0.78	0.74	0.73	0.83	0.74	0.84	0.78	0.78	0.77	0.84
W2V	QUA	0.87	0.92	0.91	0.91	0.92	0.94	0.91	0.89	0.87	0.87	0.87	0.87	0.88	0.83	0.87	0.9

B. RQ2: Which feature selection techniques are the best at getting rid of redundant and irrelevant features which may affect the performance of the classification models?

In the proposed study, we use Rank Sum test and PCA as feature selection procedures, and we use all of the original features for developing predictive models for requirements categorization in a third set of models. These feature selection procedures were applied to both the functional and quality requirements datasets.

TABLE II: Accuracy and AUC

		Accuracy								AUC							
		DL1	**DL2**	**DL3**	**DL4**	**DL5**	**DL6**	**DL7**	**DL8**	**DL1**	**DL2**	**DL3**	**DL4**	**DL5**	**DL6**	**DL7**	**DL8**
		OD(AF)															
TFIDF	FUN	83.2	84.32	85.12	85.60	85.12	84.80	84.80	85.92	0.9	0.91	0.92	0.92	0.92	0.91	0.91	0.91
TFIDF	QUA	83.52	91.36	91.04	91.20	90.88	90.40	90.40	91.20	0.93	0.96	0.96	0.96	0.96	0.96	0.96	0.95
CBOW	FUN	73.6	77.44	77.12	77.44	76.00	74.88	76.80	77.44	0.8	0.85	0.84	0.85	0.84	0.83	0.84	0.84
CBOW	QUA	76	81.92	83.68	83.68	83.36	82.40	76.96	82.88	0.83	0.89	0.9	0.88	0.89	0.89	0.84	0.89
SKG	FUN	77.12	77.60	79.04	77.76	77.76	77.76	80.16	77.12	0.86	0.86	0.86	0.86	0.85	0.87	0.87	0.85
SKG	QUA	84.16	87.52	85.92	84.64	87.68	84.96	85.44	83.68	0.91	0.93	0.93	0.92	0.92	0.91	0.92	0.91
GLOVE	FUN	78.08	80.48	80.32	79.20	80.16	78.24	80.96	78.88	0.86	0.87	0.87	0.86	0.87	0.87	0.85	0.87
GLOVE	QUA	82.88	83.20	84.32	83.04	84.32	82.88	82.88	82.4	0.9	0.91	0.92	0.9	0.91	0.9	0.91	0.89
W2V	FUN	81.44	78.88	80.64	80.00	79.36	79.84	78.08	80.64	0.89	0.88	0.88	0.88	0.87	0.86	0.87	0.87
W2V	QUA	86.72	87.04	85.60	87.04	86.72	87.52	86.56	84.8	0.92	0.93	0.93	0.94	0.93	0.93	0.93	0.92
		OD(RST)															
TFIDF	FUN	82.56	84.80	83.20	83.36	83.52	82.88	83.84	83.36	0.89	0.92	0.92	0.92	0.91	0.91	0.91	0.91
TFIDF	QUA	84.64	89.28	88.80	89.12	89.28	88.64	89.28	88.16	0.93	0.95	0.95	0.95	0.95	0.94	0.95	0.93
CBOW	FUN	76	79.04	77.44	77.44	77.92	76.96	76.48	78.24	0.84	0.87	0.86	0.86	0.86	0.86	0.85	0.86
CBOW	QUA	78.08	83.68	82.88	82.88	82.72	82.24	83.20	82.88	0.86	0.9	0.9	0.89	0.91	0.85	0.9	0.91
SKG	FUN	77.60	78.72	77.92	80.32	78.08	76.8	80.32	77.60	0.85	0.88	0.86	0.86	0.86	0.85	0.86	0.85
SKG	QUA	83.04	86.88	85.60	87.20	87.36	85.92	87.20	86.08	0.9	0.92	0.92	0.92	0.92	0.91	0.92	0.91
GLOVE	FUN	78.24	80.32	80.80	80.32	80.48	79.84	79.20	80.48	0.86	0.88	0.88	0.88	0.88	0.87	0.87	0.88
GLOVE	QUA	81.92	83.84	84.80	86.24	85.76	85.76	85.28	84.80	0.89	0.91	0.91	0.92	0.92	0.92	0.92	0.9
W2V	FUN	81.44	81.76	78.24	81.12	80.96	80.64	80.16	80.16	0.89	0.9	0.89	0.9	0.9	0.89	0.88	0.89
W2V	QUA	85.12	88.16	87.84	88.32	86.72	85.76	86.88	87.36	0.93	0.94	0.94	0.94	0.93	0.93	0.93	0.93
		OD(PCA)															
TFIDF	FUN	71.52	76.80	75.84	75.04	73.60	76.64	74.56	73.12	0.78	0.85	0.84	0.83	0.82	0.84	0.81	0.82
TFIDF	QUA	76.16	82.56	82.56	85.12	82.24	81.60	83.36	82.72	0.82	0.91	0.9	0.92	0.89	0.89	0.85	0.89
CBOW	FUN	64.48	67.04	71.84	68.80	66.56	71.52	67.84	70.56	0.68	0.75	0.79	0.77	0.74	0.78	0.74	0.77
CBOW	QUA	63.2	73.60	79.36	76.16	72.64	74.24	74.24	78.08	0.65	0.81	0.85	0.83	0.81	0.82	0.8	0.85
SKG	FUN	73.6	79.04	76.64	78.40	80.00	76.96	78.56	76.48	0.8	0.86	0.85	0.87	0.86	0.86	0.87	0.86
SKG	QUA	74.56	85.44	86.40	86.08	84.64	84.16	84.32	84.48	0.89	0.92	0.93	0.92	0.91	0.91	0.91	0.92
GLOVE	FUN	76.16	77.12	76.96	79.36	76.80	79.68	77.76	78.08	0.84	0.85	0.84	0.85	0.85	0.86	0.84	0.85
GLOVE	QUA	76.48	82.40	84.32	83.04	82.24	84.80	84.00	83.52	0.86	0.9	0.91	0.9	0.9	0.91	0.9	0.89
W2V	FUN	75.84	78.72	78.72	78.40	77.92	77.12	78.40	77.12	0.84	0.85	0.86	0.86	0.86	0.86	0.84	0.84
W2V	QUA	81.6	84.80	84.16	84.64	84.96	83.68	83.84	82.72	0.9	0.92	0.92	0.92	0.92	0.92	0.9	0.91
		SMOTE(AF)															
TFIDF	FUN	82.22	84.13	84.29	84.60	85.08	83.65	84.29	82.22	0.89	0.91	0.91	0.91	0.91	0.91	0.9	0.89
TFIDF	QUA	90.05	93.32	93.19	93.98	94.11	94.24	93.59	93.32	0.96	0.98	0.98	0.98	0.98	0.98	0.98	0.98
CBOW	FUN	66.51	76.35	75.40	77.62	74.60	76.03	69.05	73.17	0.79	0.84	0.83	0.83	0.84	0.84	0.77	0.83
CBOW	QUA	78.93	87.17	87.30	79.97	85.60	86.91	83.12	85.47	0.9	0.92	0.92	0.88	0.92	0.92	0.89	0.92
SKG	FUN	77.62	76.83	76.03	74.44	78.73	75.71	79.21	75.71	0.85	0.85	0.83	0.84	0.86	0.84	0.85	0.83
SKG	QUA	87.17	87.83	90.31	87.04	88.35	88.22	90.71	89.66	0.93	0.95	0.96	0.93	0.96	0.94	0.95	0.95
GLOVE	FUN	80.16	81.43	80.63	78.25	77.94	80.95	78.25	80.16	0.87	0.89	0.88	0.87	0.87	0.89	0.87	0.87
GLOVE	QUA	86.39	88.61	89.14	88.87	89.14	88.87	87.83	88.09	0.93	0.95	0.96	0.95	0.95	0.94	0.94	0.94
W2V	FUN	76.67	82.54	80.95	81.43	79.84	80.79	79.21	79.37	0.85	0.9	0.89	0.9	0.9	0.88	0.88	0.87
W2V	QUA	88.48	89.92	89.79	88.48	90.97	91.49	89.01	84.16	0.95	0.96	0.96	0.95	0.96	0.96	0.96	0.94
		SMOTE(RST)															
TFIDF	FUN	80.32	83.81	84.13	82.86	83.49	83.17	84.60	83.81	0.9	0.92	0.92	0.92	0.92	0.91	0.92	0.91
TFIDF	QUA	87.43	90.97	91.10	91.23	91.62	90.71	92.02	92.41	0.94	0.97	0.97	0.97	0.97	0.97	0.97	0.97
CBOW	FUN	76.51	79.05	77.94	77.94	78.10	76.67	79.21	78.10	0.84	0.87	0.86	0.85	0.87	0.86	0.87	0.86
CBOW	QUA	82.2	86.39	85.47	87.57	87.30	87.57	86.13	86.26	0.89	0.93	0.93	0.93	0.93	0.94	0.92	0.92
SKG	FUN	76.83	79.68	80.79	79.52	78.41	79.52	78.89	78.41	0.85	0.88	0.88	0.87	0.87	0.88	0.87	0.86
SKG	QUA	86.13	88.35	88.09	86.65	89.27	88.22	87.43	88.35	0.92	0.95	0.96	0.95	0.95	0.95	0.95	0.95
GLOVE	FUN	78.57	81.59	81.43	78.57	81.27	79.52	78.57	75.56	0.86	0.88	0.89	0.88	0.88	0.87	0.87	0.84
GLOVE	QUA	85.47	89.01	88.74	88.22	88.61	90.58	87.96	88.87	0.93	0.94	0.95	0.94	0.95	0.95	0.94	0.93
W2V	FUN	80.16	81.75	80.48	80.00	82.06	82.70	80.63	78.57	0.89	0.9	0.91	0.9	0.91	0.9	0.9	0.88
W2V	QUA	88.22	88.48	90.05	90.97	90.18	90.31	89.14	89.27	0.94	0.96	0.97	0.97	0.97	0.97	0.96	0.95
		SMOTE(PCA)															
TFIDF	FUN	62.38	73.97	74.44	75.08	73.81	74.92	65.24	74.76	0.71	0.83	0.82	0.84	0.82	0.83	0.73	0.82
TFIDF	QUA	86.78	89.66	89.14	85.99	90.05	85.08	86.65	88.87	0.94	0.96	0.96	0.94	0.96	0.93	0.93	0.95
CBOW	FUN	59.68	69.52	75.56	75.40	72.22	72.54	68.25	76.19	0.64	0.77	0.84	0.83	0.8	0.81	0.76	0.81
CBOW	QUA	70.16	82.07	84.29	80.50	82.98	82.07	78.80	83.51	0.76	0.89	0.91	0.89	0.88	0.89	0.86	0.9
SKG	FUN	73.17	77.78	77.94	79.05	76.19	77.30	77.30	77.30	0.81	0.85	0.85	0.86	0.85	0.85	0.85	0.84
SKG	QUA	86.13	88.87	89.79	88.87	91.10	90.31	91.23	89.14	0.93	0.95	0.95	0.95	0.96	0.95	0.95	0.95
GLOVE	FUN	74.92	77.30	77.30	77.78	76.98	77.78	77.46	77.30	0.84	0.86	0.86	0.86	0.85	0.86	0.85	0.84
GLOVE	QUA	84.95	87.57	88.48	89.14	89.27	88.74	89.27	87.96	0.93	0.94	0.94	0.94	0.94	0.94	0.94	0.93
W2V	FUN	73.65	80.32	76.83	80.16	78.89	79.21	77.62	77.30	0.81	0.88	0.86	0.88	0.87	0.87	0.86	0.86
W2V	QUA	86.91	89.53	89.40	88.87	90.18	89.01	89.01	89.14	0.94	0.96	0.95	0.96	0.96	0.96	0.95	0.95

1) Box-plot: Feature Selection: RST seems to select a better subset of features than any other technique, per Figure 5. RST has an average AUC of 0.91, with a lowest of 0.84 and a peak of 0.97. The features created using PCA performed the worst of the three sets of features, with a mean AUC of 0.87.

2) Friedman Test: Feature Selection: We also utilized the Friedman test to evaluate the various feature selection procedures based on their ability to predict model performance metrics generated with three distinct sets of features. The null hypothesis, that must be evaluated based on the Friedman

TABLE III: Descriptive Statistics: Different Word Embedding

	Min	Max	Mean	Q2	Q1	Q3
Accuracy						
TFIDF	62.38	94.24	84.52	84.62	82.56	89.47
CBOW	59.68	87.57	77.80	77.77	74.74	82.88
SKG	73.17	91.23	82.34	80.56	77.76	87.19
GLOVE	74.92	90.58	82.42	81.51	78.57	85.76
W2V	73.65	91.49	83.50	82.62	79.92	88.00
Precision						
TFIDF	0.62	0.95	0.85	0.85	0.83	0.90
CBOW	0.60	0.92	0.79	0.78	0.75	0.86
SKG	0.71	0.93	0.83	0.83	0.78	0.89
GLOVE	0.71	0.93	0.83	0.84	0.79	0.88
W2V	0.74	0.94	0.84	0.84	0.79	0.90
Recall						
TFIDF	0.48	0.98	0.85	0.85	0.82	0.92
CBOW	0.46	0.94	0.79	0.80	0.75	0.85
SKG	0.65	0.97	0.83	0.83	0.79	0.89
GLOVE	0.69	0.91	0.83	0.84	0.80	0.87
W2V	0.71	0.95	0.84	0.84	0.80	0.88
AUC						
TFIDF	0.71	0.98	0.91	0.92	0.90	0.96
CBOW	0.64	0.94	0.85	0.86	0.83	0.89
SKG	0.80	0.96	0.89	0.89	0.86	0.93
GLOVE	0.84	0.96	0.89	0.89	0.87	0.93
W2V	0.81	0.97	0.91	0.91	0.88	0.94

TABLE IV: Friedman test : Mean Rank

	Accuracy	Precision	Recall	AUC
DL				
DL1	7.33	6.28	5.17	7.28
DL2	3.53	3.94	4.39	3.25
DL3	3.69	4.02	4.28	3.16
DL4	3.83	4.13	4.35	3.65
DL5	3.84	4.13	4.13	3.65
DL6	4.45	4.43	4.44	4.23
DL7	4.51	4.33	4.88	5.22
DL8	4.83	4.75	4.36	5.58
	P≤0.05	P≤0.05	P≤0.05	P≤0.05
Word-Embedding				
TFIDF	1.95	2.18	2.01	1.88
CBOW	4.82	4.51	4.09	4.88
SKG	2.96	3.02	2.98	3.05
GLOVE	3.02	2.99	3.11	3.24
W2V	2.25	2.30	2.81	1.96
	P≤0.05	P≤0.05	P≤0.05	P≤0.05
Feature Sets				
AF	1.74	1.71	1.83	1.81
RST	1.58	1.76	1.74	1.47
PCA	2.68	2.53	2.43	2.73
	P≤0.05	P≤0.05	P≤0.05	P≤0.05
OD and SMOTE				
OD	1.74	1.64	1.47	1.77
SMOTE	1.26	1.36	1.53	1.23
	P≤0.05	P≤0.05	P≤0.05	P≤0.05

test, is that "the requirements classification models developed using different feature sets do not have a significant difference in their prediction capacity." With two degrees of freedom and a significance threshold of α= 0.05, the Friedman test was conducted. Table IV shows the mean ranks of the three feature sets. The mean ranks of the Friedman test can be used to differentiate between different feature selection techniques. Lower mean ranks imply better achievement as compared to others. The models trained with the set of RST features had the lowest mean rank (1.47), followed by those trained with the original set of features (1.81), and finally PCA (2.73). The mean ranks show that models perform better when

TABLE V: Descriptive Statistics: Different Sets of Features

	Min	Max	Mean	Q2	Q1	Q3
Accuracy						
AF	66.51	94.24	83.14	83.20	78.91	87.17
RST	75.56	92.41	83.59	83.28	79.92	87.43
PCA	59.68	91.23	79.62	78.64	75.84	84.64
Precision						
AF	0.66	0.95	0.84	0.84	0.79	0.89
RST	0.71	0.93	0.84	0.85	0.79	0.89
PCA	0.60	0.94	0.80	0.80	0.76	0.88
Recall						
AF	0.46	0.98	0.84	0.84	0.80	0.88
RST	0.70	0.98	0.84	0.84	0.80	0.89
PCA	0.48	0.97	0.81	0.82	0.77	0.87
AUC						
AF	0.77	0.98	0.90	0.90	0.87	0.93
RST	0.84	0.97	0.91	0.91	0.88	0.93
PCA	0.64	0.96	0.87	0.86	0.84	0.92

RST features are included, whereas dimensionality reduction approaches like PCA just regress the models' performance.

C. RQ3: For what structure of the deep learning classifiers do the software requirements classification models achieve the best results?

The study used eight different Deep Learning classifiers to categorize the software requirements. These classifiers are employed in conjunction with various word-embedding techniques and feature selection techniques. The DL binary classification models include models with different layer sizes, and layer types. Hyperparameters such as optimizer, activation function for specific layers, loss function, etc. are kept consistent across different models.

1) Box-plot: Classification Techniques: Figure 6 depicts the accuracy, precision, recall, and AUC statistics for the

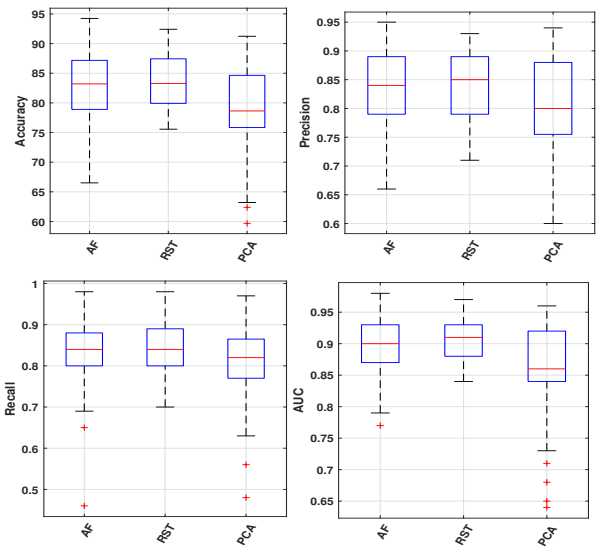

Fig. 5: Performance Box-Plot Diagram: Performance of Different Sets of Features

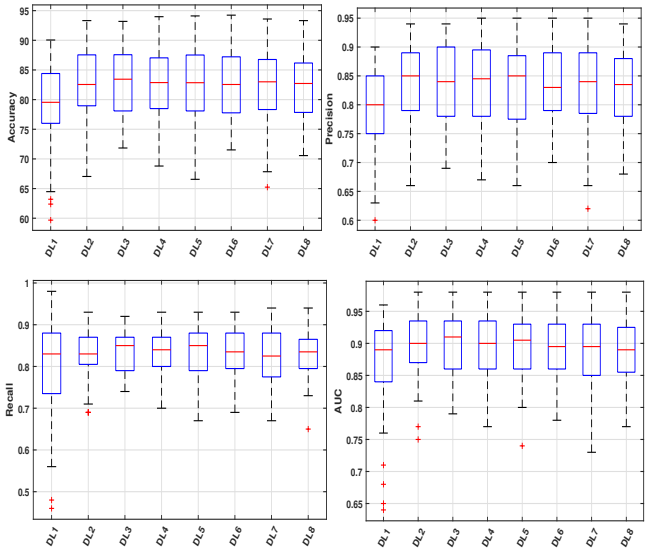

Fig. 6: Performance Box-Plot Diagram: Different Variants of Deep Learning

different classifiers used, including Mean, Median, Min, Max, Q1, and Q3. With a mean AUC of 0.9, the models trained with the DL2, DL3, DL4, and DL5 exhibited relatively similar performance and surpassed the others. The DL1 classifier performed the worst in comparison to other classifiers, with a mean AUC of 0.86. The DL3 classifier produces models with a maximum AUC of 0.98, a minimum of 0.79, a Q1 AUC of 0.86, and a Q3 AUC of 0.94. A box plot is insufficient to discriminate between the DL2, DL3, DL4, and DL5 classifiers.

2) Friedman Test: Classification Techniques: The Friedman test is also performed on the performance metrics of the various classifiers in order to statistically compare the models' performance and help differentiate the models where box-plot failed. The goal of the test is to see if the null hypothesis is correct. The null hypothesis for this test is that "the requirements classification models developed utilizing the different classifiers do not have a significant variation in their prediction abilities." With 7 degrees of freedom and a significance level of α= 0.05, the Friedman test was performed. The mean rank of various classifiers after the Friedman test is shown in Table IV. Table IV shows that DL3 has the lowest mean rank of 3.16, followed by DL2 with 3.25, DL4, and DL5 with 3.65. DL1 and DL8 performed significantly worse with mean ranks of 7.28, and 5.58, respectively.

D. RQ4: How does the application of class balancing technique through oversampling affect the performance of the models?

Based on the data utilized to train the models, there are two types of models presented in this study. One dataset contains imbalanced classes, while the class-balanced dataset is used to train the other set of models.

1) Box-plot: SMOTE: Figure 7 presents a visual representation of the predictive ability of models trained on a balanced

TABLE VI: Descriptive Statistics: Different Variants of DL

	Min	Max	Mean	Q2	Q1	Q3
Accuracy						
DL1	59.68	90.05	79.06	79.55	76.00	84.40
DL2	67.04	93.32	82.79	82.55	78.96	87.55
DL3	71.84	93.19	82.93	83.44	78.09	87.57
DL4	68.80	93.98	82.67	82.87	78.49	87.04
DL5	66.56	94.11	82.70	82.85	78.09	87.52
DL6	71.52	94.24	82.54	82.55	77.77	87.22
DL7	65.24	93.59	82.02	83.00	78.33	86.77
DL8	70.56	93.32	82.22	82.72	77.84	86.17
Precision						
DL1	0.60	0.90	0.80	0.80	0.75	0.85
DL2	0.66	0.94	0.84	0.85	0.79	0.89
DL3	0.69	0.94	0.84	0.84	0.78	0.90
DL4	0.67	0.95	0.84	0.85	0.78	0.90
DL5	0.66	0.95	0.83	0.85	0.78	0.89
DL6	0.70	0.95	0.83	0.83	0.79	0.89
DL7	0.62	0.95	0.83	0.84	0.79	0.89
DL8	0.68	0.94	0.83	0.84	0.78	0.88
Recall						
DL1	0.46	0.98	0.81	0.83	0.74	0.88
DL2	0.69	0.93	0.83	0.83	0.81	0.87
DL3	0.74	0.92	0.84	0.85	0.79	0.87
DL4	0.70	0.93	0.83	0.84	0.80	0.87
DL5	0.67	0.93	0.84	0.85	0.79	0.88
DL6	0.69	0.93	0.83	0.84	0.80	0.88
DL7	0.67	0.94	0.83	0.83	0.78	0.88
DL8	0.65	0.94	0.83	0.84	0.80	0.87
AUC						
DL1	0.64	0.96	0.86	0.89	0.84	0.92
DL2	0.75	0.98	0.90	0.90	0.87	0.94
DL3	0.79	0.98	0.90	0.91	0.86	0.94
DL4	0.77	0.98	0.90	0.90	0.86	0.94
DL5	0.74	0.98	0.90	0.91	0.86	0.93
DL6	0.78	0.98	0.90	0.90	0.86	0.93
DL7	0.73	0.98	0.89	0.90	0.85	0.93
DL8	0.77	0.98	0.89	0.89	0.86	0.93

dataset versus models learned on an imbalanced dataset. In most box plot measures, models trained with SMOTE outperformed models trained on the original dataset. SMOTE-trained models had a mean AUC of 0.9, a maximum of 0.98, a minimum of 0.64, and a Q3 of 0.95.

TABLE VII: Descriptive Statistics: OD and SMOTE

	Min	Max	Mean	Q2	Q1	Q3
Accuracy						
OD	63.20	91.36	81.20	81.84	77.76	84.80
SMOTE	59.68	94.24	83.03	83.50	78.18	88.68
Precision						
OD	0.63	0.94	0.82	0.83	0.78	0.88
SMOTE	0.60	0.95	0.84	0.84	0.78	0.90
Recall						
OD	0.67	0.98	0.83	0.84	0.79	0.88
SMOTE	0.46	0.93	0.83	0.84	0.79	0.87
AUC						
OD	0.65	0.96	0.88	0.89	0.86	0.92
SMOTE	0.64	0.98	0.90	0.91	0.86	0.95

2) Friedman Test: SMOTE: To evaluate the performance of the two sets of models, the Friedman test is applied to their performance measures on class-balanced and imbalanced datasets. The goal of this test is to accept or reject the null hypothesis, which states that "there is no substantial difference in performance between models trained with balanced or imbalanced classes." With a significance threshold of α = 0.05

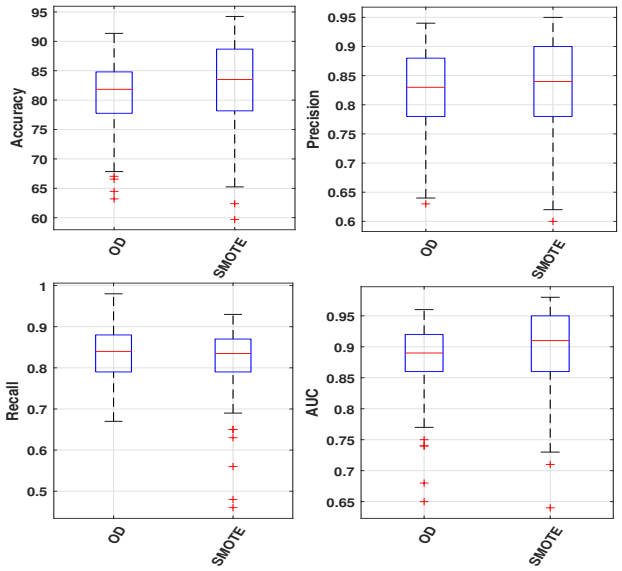

Fig. 7: Performance Box-Plot Diagram: Performance of Original Data and SMOTE

and degrees of freedom of 1, the Friedman test was performed. The models trained on the class-balanced dataset have the lower mean rank of 1.23, while those trained on the imbalanced dataset have the higher mean rank of 1.77.

VII. CONCLUSION

The Deep Learning classifiers used in this work, in conjunction with the word embedding, feature selection, and class balancing techniques have produced a very high accuracy for software requirements classification. The best models can be deployed in industry to do away with the manual classification, which is ailed by the problem of inconsistencies between client and software engineer terminologies [13]. Industry can benefit from the lower costs, and higher efficiency. The key conclusions that we arrived at were as follows:

- Models that utilized features extracted from TF-IDF word embedding performed considerably better than other models.
- Out of the eight DL classifiers used, the models trained with the DL3 classifier had the best results, with the DL2 and DL4 classifiers performing similarly.
- The Rank Sum test-selected features outperformed any other group of characteristics utilized in this work.
- SMOTE-based class balancing improved the performance of requirements categorization models.

According to the results of this study, DL classifiers are critical for more accurate software requirement classification. Future research can extend this work by classifying the requirements into the subcategories of functional and non-functional requirements. Researchers can also focus on other parts of the software development pipeline, and use the models developed in this work for the requirements classification step.

VIII. ACKNOWLEDGEMENTS

This research is funded by TestAIng Solutions Pvt. Ltd.

REFERENCES

[1] M. V. Mäntylä, F. Calefato, and M. Claes, "Natural language or not (nlon): A package for software engineering text analysis pipeline," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 387–391. [Online]. Available: https://doi.org/10.1145/3196398.3196444

[2] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements engineering*, vol. 12, no. 2, pp. 103–120, 2007.

[3] M. H. Osman and M. F. Zaharin, "Ambiguous software requirement specification detection: An automated approach," in *2018 IEEE/ACM 5th International Workshop on Requirements Engineering and Testing (RET)*, 2018, pp. 33–40.

[4] E. Knauss, D. Damian, G. Poo-Caamaño, and J. Cleland-Huang, "Detecting and classifying patterns of requirements clarifications," in *2012 20th IEEE International Requirements Engineering Conference (RE)*, 2012, pp. 251–260.

[5] 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)*, 2017, pp. 116–120.

[6] A. Araujo and R. Marcacini, "Hierarchical cluster labeling of software requirements using contextual word embeddings," in *Brazilian Symposium on Software Engineering*, 2021, pp. 297–302.

[7] D. Ott, "Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2013, pp. 50–64.

[8] C. Baker, L. Deng, S. Chakraborty, and J. Dehlinger, "Automatic multi-class non-functional software requirements classification using neural networks," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2019, pp. 610–615.

[9] N. Rahimi, F. Eassa, and L. Elrefaei, "An ensemble machine learning technique for functional requirement classification," *symmetry*, vol. 12, no. 10, p. 1601, 2020.

[10] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[11] S. Tiun, U. Mokhtar, S. Bakar, and S. Saad, "Classification of functional and non-functional requirement in software requirement using word2vec and fast text," in *journal of Physics: conference series*, vol. 1529, no. 4. IOP Publishing, 2020, p. 042077.

[12] J. Hassine, R. Dssouli, and J. Rilling, "Applying reduction techniques to software functional requirement specifications," in *System Analysis and Modeling*, D. Amyot and A. W. Williams, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 138–153.

[13] C. P. Guevara-Vega, E. D. Guzmán-Chamorro, V. A. Guevara-Vega, A. V. B. Andrade, and J. A. Quiña-Mera, "Functional requirement management automation and the impact on software projects: case study in ecuador," in *International Conference on Information Technology & Systems*. Springer, 2019, pp. 317–324.